MACHINE LEARNING

Dr. U. Hemamalini

Mrs. V. Varalakshmi

Mrs. B. Ilakkiya Arasi

Mrs. Shina. M. K

Imaginex Inks Publication

71A 71B First Street, RKV Avenue,
Old Pallavaram, Chennai 600117, India.
Phone: +919962991087
e-mail: info@imaginexinkspublication.com
https://www.imaginexinkspublication.com/

Machine Learning

Authored by

Dr. U. Hemamalini, Mrs. V. Varalakshmi, Mrs. B. Ilakkiya Arasi and Mrs. Shina. M. K

18th May, 2026

Price: Rs. 450/-

ISBN: 978-93-47966-01-9

Published by and copies can be had from:

Imaginex Inks Publication

71A 71B First Street, RKV Avenue,

Old Pallavaram, Chennai 600117, India.

Phone:9750663871, 9962991057

e-mail: info@imaginexinkspublication.com

https://www.imaginexinkspublication.com/

Preface

Machine learning has become one of the most important technologies shaping modern science, industry, and society. This book, Machine Learning: Foundations, Algorithms, and Applications, is written to give readers a clear understanding of machine learning from basic mathematical principles to practical real-world use.

The book begins with essential foundations such as linear algebra, probability, statistics, optimisation, and learning theory. It then explains supervised learning, unsupervised learning, deep learning, model evaluation, deployment, MLOps, and responsible machine learning. Special attention is also given to fairness, interpretability, privacy, and emerging areas such as foundation models and causal learning.

This book is intended for students, researchers, teachers, and professionals who wish to learn machine learning in a structured and practical way. It aims to build not only technical knowledge but also the ability to apply machine learning responsibly and effectively in real-world problems.

Acknowledgments

We express our sincere gratitude to **Vels Institute of Science, Technology and Advanced Studies (VISTAS), Chennai**, for providing the academic environment, encouragement, and support for the preparation of this book.

We are deeply thankful to the management, authorities, and academic leadership of VISTAS for their continuous motivation and guidance. We also extend our heartfelt thanks to our colleagues, faculty members, research scholars, and students for their valuable suggestions, cooperation, and encouragement throughout the development of this work.

We are grateful to our families and well-wishers for their constant support and inspiration. Finally, we thank the publisher for their assistance and cooperation in bringing out this book successfully.

Table of Contents

Part I — Mathematical Foundations

Chapter 1: The Language of Learning — Linear Algebra, Probability, and Optimisation

Chapter 2: Learning Theory — Generalisation, Bias-Variance, and the PAC Framework

The Language of Learning

Linear Algebra, Probability, and Optimisation

Chapter 1

The Language of Learning — Linear Algebra, Probability, and Optimisation

In the year 1895, a British statistician named Karl Pearson introduced a method he called the method of moments for fitting theoretical distributions to observed data. He had, in effect, solved an optimisation problem — minimising the discrepancy between a model and observations — using only algebraic manipulation and physical intuition, because the differential calculus of optimisation he needed did not yet exist in the form required. The method worked, imperfectly, and Pearson could only guess at why.

One hundred and thirty years later, a student at any one of India's National Institutes of Technology can open a Python interpreter, load a dataset of, say, daily rainfall measurements from the India Meteorological Department, and fit a mixture of Gaussian distributions to that data in fewer than ten lines of code. The fitting algorithm — the Expectation-Maximisation algorithm — solves an optimisation problem of a kind that would have taken Pearson months of manual calculation. It runs in seconds. The student need not understand why it works.

The purpose of the present chapter is to ensure that the reader does understand why — not as an abstract exercise in mathematical appreciation, but because the understanding of mathematical foundations is what separates a practitioner who can apply machine learning tools from a researcher who can improve them, diagnose their failures, and invent new ones. The present chapter makes no apology for its mathematical content; it is, rather, an invitation to discover that the mathematics of machine learning is not difficult so much as it is beautiful, and that beauty, once perceived, is the

most reliable aid to memory and understanding that the discipline of study has yet produced.

1.1 Vectors and Matrices: The Geometry of Data

At the most fundamental level, machine learning is the study of patterns in collections of numbers. Every observation that a machine learning system processes — a patient's laboratory test results, a satellite image of a field in Punjab, a sentence in Tamil, the closing price of a stock on the Bombay Stock Exchange — must, before any algorithm can operate upon it, be represented as a collection of numbers. The branch of mathematics that provides the language and tools for reasoning about collections of numbers and the transformations between them is linear algebra, and it is with linear algebra that the present chapter begins.

A vector, in the context of machine learning, is an ordered list of real numbers. If a patient's medical record contains measurements of haemoglobin level, white blood cell count, platelet count, and ten other laboratory values, then that patient's record can be represented as a vector of fourteen numbers — a point in a fourteen-dimensional space. Each dimension of this space corresponds to one measurement; the position of the point within the space encodes the complete pattern of that patient's laboratory values. This geometric interpretation of data as points in a high-dimensional space is not merely a notational convenience; it is the conceptual foundation upon which virtually every geometric and algebraic insight in machine learning rests.

The inner product of two vectors x and y, denoted $x^{\mathsf{T}}y$ or $\langle x, y \rangle$, is defined as the sum of the products of corresponding components. It has a geometric interpretation of considerable importance: it equals the product of the lengths of the two vectors and the cosine of the angle between them. Two vectors whose inner

product is zero are orthogonal — they point in directions that are, in a precise mathematical sense, completely independent of one another. The inner product is the mathematical operation that underlies virtually every notion of similarity in machine learning: two data points whose feature vectors have a large inner product are, in a meaningful sense, similar.

A matrix is a rectangular array of numbers, and it may be understood geometrically as a linear transformation — a function that takes a vector as input and produces another vector as output. The action of a matrix on a vector rotates, scales, projects, and shears the vector in ways determined by the matrix's entries. Understanding matrix operations geometrically — not merely as rules for combining arrays of numbers but as descriptions of transformations of space — is one of the most productive shifts in perspective available to the student of machine learning. A weight matrix in a neural network is, geometrically, a transformation that maps the activation space of one layer into the activation space of the next; gradient descent is a procedure for finding the transformation that maps inputs most usefully to outputs.

The eigendecomposition of a square matrix A expresses it as a product of the form $A = Q\Lambda Q^{-1}$, where Q is a matrix whose columns are the eigenvectors of A and Λ is a diagonal matrix whose entries are the corresponding eigenvalues. The eigenvectors of a matrix are the directions in space that the matrix preserves under transformation — the vectors that are merely scaled, not rotated, by the matrix's action. For a symmetric positive definite matrix — the kind that arises as a covariance matrix of a data distribution — all eigenvalues are positive, and the eigenvectors form an orthogonal basis. The eigenvector corresponding to the largest eigenvalue points in the direction of greatest variance in the data; the eigenvector corresponding to the second-largest

eigenvalue points in the direction of second-greatest variance, and so on. This observation is the geometric foundation of Principal Component Analysis, which is treated in Chapter 6 of the present volume.

The Singular Value Decomposition (SVD) extends the eigendecomposition to non-square matrices. Any matrix A of dimensions $m \times n$ can be decomposed as $A = U\Sigma V^{T}$, where U is an $m \times m$ orthogonal matrix, Σ is an $m \times n$ diagonal matrix with non-negative entries (the singular values), and V is an $n \times n$ orthogonal matrix. The SVD reveals the fundamental structure of any linear transformation, and it is the computational workhorse of dimensionality reduction, matrix factorisation for recommender systems, and the analysis of the representations learned by neural networks. When the Indian Space Research Organisation's Earth observation scientists apply low-rank approximations to satellite imagery for compression and noise reduction, the SVD is the underlying mathematical operation.

1.2 Probability and Statistics: Reasoning Under Uncertainty

Machine learning algorithms do not operate on the world directly; they operate on data, which is an incomplete, noisy, and finite sample from a larger reality. The mathematical framework for reasoning rigorously about incomplete information, uncertainty, and the relationship between samples and populations is probability theory and mathematical statistics, and no serious engagement with machine learning is possible without a working understanding of its fundamental concepts.

A probability distribution is a function that assigns a probability — a number between zero and one — to each possible outcome of a random phenomenon, in such a way that the

probabilities of all possible outcomes sum to one. Probability distributions are the language in which machine learning models describe their uncertainty about the world. A classifier that assigns probabilities to class labels is not merely providing a numerical score; it is asserting a probability distribution over the possible classes, and the quality of that distribution — not merely its most probable outcome — is what ultimately determines the classifier's usefulness in decision-making contexts.

Of the many probability distributions that arise in machine learning, the Gaussian distribution — also called the normal distribution — occupies a position of particular centrality. Its probability density function is characterised by two parameters: the mean μ, which determines where the distribution is centred, and the variance σ^2, which determines how spread out it is. The Gaussian distribution is the unique maximum entropy distribution on the real line subject to fixed mean and variance, which means it is, in a precise information-theoretic sense, the least informative distribution consistent with those two moments. This property, together with the Central Limit Theorem — which establishes that the sum of many independent random variables converges in distribution to a Gaussian regardless of the individual variables' distributions — explains why Gaussian models appear so frequently as baseline assumptions in machine learning and statistics.

Bayes' theorem is the rule for updating probability distributions in the light of new evidence. In its most general form, it states that the posterior probability of a hypothesis H given observed evidence E is proportional to the product of the prior probability of H and the likelihood of observing E given H. In machine learning, Bayesian inference provides a principled framework for learning from data: the prior distribution encodes

our beliefs about the parameters of a model before observing any data, the likelihood function describes the probability of the observed data under each possible set of parameter values, and the posterior distribution updates our beliefs after observing the data. Maximum Likelihood Estimation (MLE) — perhaps the most widely used parameter estimation technique in machine learning — corresponds to the special case of Bayesian inference with a uniform prior, in which the posterior is proportional to the likelihood alone.

Information theory, developed by Claude Shannon in 1948, provides a third perspective on probability that is essential for understanding many machine learning algorithms. The entropy of a probability distribution p(x), defined as $H(p) = -\Sigma\, p(x) \log p(x)$, measures the average uncertainty or information content of random samples drawn from that distribution. A distribution concentrated on a single outcome has zero entropy; a uniform distribution over n outcomes has maximum entropy of log n bits. The Kullback-Leibler (KL) divergence between two distributions p and q, defined as $KL(p\|q) = \Sigma\, p(x) \log(p(x)/q(x))$, measures the information lost when q is used to approximate p. Minimising KL divergence is equivalent to maximum likelihood estimation, which reveals that these two seemingly distinct estimation frameworks are, in fact, the same thing viewed from different vantage points.

Table 1.1 Common Probability Distributions — Properties and Machine Learning Roles

Distribution	Parameters	Support	Key Property	ML Application
Gaussian (Normal)	μ (mean), σ^2 (variance)	$(-\infty, +\infty)$	Maximum entropy for fixed mean and variance; characterised by CLT	Linear regression noise model; VAEs;

				Gaussian processes; generative models
Bernoulli	$p \in (0,1)$	{0, 1}	Models a single binary trial	Binary classification output; dropout masks; latent variable sampling
Categorical / Multinoulli	$p = (p_1,...,p_k)$, $\Sigma p_i=1$	{1,...,K}	Generalises Bernoulli to K outcomes	Softmax classifier output; discrete latent variables; language model tokens
Multivariate Gaussian	$\mu \in \mathbb{R}^d$, $\Sigma \in \mathbb{R}^{dxd}$ (PSD)	$\mathbb{R}^d$	Covariance matrix captures feature correlations	Gaussian discriminant analysis; Gaussian mixtures; Kalman filtering
Beta	$\alpha > 0, \beta > 0$	[0, 1]	Conjugate prior for Bernoulli/binomial likelihood	Bayesian proportion estimation; click-through rate modelling
Dirichlet	$\alpha = (\alpha_1,...,\alpha_k) > 0$	K-simplex	Conjugate prior for Categorical/Multinomial	Topic modelling (LDA); Bayesian neural networks; mixture proportion priors

Laplace	μ (location), b (scale)	(−∞, +∞)	Heavier tails than Gaussian; MAP estimation yields L1 regularisation	Sparse Bayesian regression; robust regression; lasso equivalence
Exponential / Gamma	λ > 0 / (k, θ)	(0, +∞)	Models positive quantities; Gamma is conjugate for Poisson rate	Survival analysis; time-between-events; Poisson rate Bayesian inference

PSD = Positive Semi-Definite. CLT = Central Limit Theorem. LDA = Latent Dirichlet Allocation. VAE = Variational Autoencoder. Sources: Bishop (2006); Murphy (2012); Goodfellow et al. (2016).

1.3 Optimisation: Finding the Best Parameters

Every supervised machine learning algorithm, at its computational core, solves an optimisation problem: it seeks the values of a set of model parameters that minimise some measure of the discrepancy between the model's predictions and the observed data. The mathematical field of optimisation — the study of methods for finding the inputs that minimise or maximise a function — is therefore not a peripheral concern of machine learning but its computational engine.

The loss function $L(\theta)$, also called the objective function or cost function, quantifies the discrepancy between the model's predictions $f(x; \theta)$ and the true labels y for the training data. The choice of loss function is not merely a computational detail; it encodes a statistical assumption about the data-generating process. The mean squared error loss — $L(\theta) = (1/n)\Sigma(y_i - f(x_i; \theta))^2$ — corresponds to the assumption that the residuals are normally

distributed. The cross-entropy loss — $L(\theta) = -\Sigma\, y_i \log f(x_i; \theta)$ — corresponds to the assumption that the labels are generated by a categorical distribution parameterised by the model's output probabilities. Understanding the probabilistic interpretation of loss functions enables the practitioner to make principled choices among them, rather than treating them as interchangeable numerical conveniences.

Gradient descent is the foundational algorithm for minimising differentiable loss functions. The gradient $\nabla_\theta L(\theta)$ is a vector that points in the direction of steepest ascent of the loss function at the current parameter values θ. Gradient descent updates the parameters in the opposite direction — the direction of steepest descent — by a step proportional to the learning rate η:

$$\theta \leftarrow \theta - \eta \cdot \nabla_\theta L(\theta)$$

Repeated application of this update, provided the learning rate is chosen appropriately and the loss function is convex, is guaranteed to converge to the global minimum. For non-convex loss functions — the kind that arise in deep neural network training — gradient descent converges to a local minimum, and the question of whether the local minima of neural network loss landscapes are practically as good as the global minimum has been the subject of intensive theoretical investigation.

Stochastic Gradient Descent (SGD) and its variants are the practical algorithms of choice for training large-scale machine learning models. Rather than computing the gradient over the entire training dataset at each step — which is computationally prohibitive for datasets of the size encountered in practice — SGD approximates the gradient using a small randomly sampled subset of the data, called a mini-batch. The noise introduced by this approximation is, counterintuitively, often beneficial: it acts as an

implicit regulariser, helping the algorithm escape shallow local minima and find solutions that generalise better to unseen data. The Momentum algorithm, which accumulates a velocity vector in the direction of persistent gradient, and the Adam optimiser, which adaptively scales the learning rate for each parameter using estimates of the first and second moments of the gradient, are the most widely used variants in practice.

Convexity is a property of a function whose domain is a convex set and whose graph lies below the line segment connecting any two of its points. A convex loss function has a single global minimum — or a convex set of global minima — and gradient descent is guaranteed to find it. Linear regression and logistic regression both have convex loss functions; neural networks generally do not. Understanding whether a given optimisation problem is convex determines what theoretical guarantees one can make about the convergence and optimality of the solution found by gradient descent, and is therefore an important diagnostic in the analysis of new algorithms.

Table 1.2 Optimisation Algorithms — Properties, Convergence, and Practical Guidance

Algorithm	Update Rule (summary)	Per-step Cost	Convergence (convex case)	Key Hyperparameter(s)	Recommended Use
Gradient Descent (batch)	$\theta \leftarrow \theta - \eta \nabla L(\theta)$ on full data	$O(n \cdot d)$	Linear rate; guaranteed to global min	Learning rate η	Small datasets; convex problems; analysis and comparison baseline

Stochastic GD (SGD)	$\theta \leftarrow \theta - \eta \nabla L(\theta; x_i)$ on one sample	O(d)	Sublinear; oscillates near minimum	η; mini-batch size	Large datasets; online learning; implicit regularisation desired
Mini-batch SGD	$\theta \leftarrow \theta - \eta \nabla L(\theta; B)$ on batch B	O(|B|·d)	Sublinear; variance reduced vs SGD	η; batch size |B|	Standard training; GPU parallelism; most practical settings
SGD with Momentum	$v \leftarrow \beta v + \nabla L$; $\theta \leftarrow \theta - \eta v$	O(|B|·d)	Faster; reduced oscillation	η; momentum $\beta \approx 0.9$	When SGD oscillates; CV tasks; ResNets and standard architectures
RMSProp	Adapts η per parameter using gradient variance	O(|B|·d)	Faster on non-stationary objectives	η; decay rate ρ	RNNs; non-stationary problems; precursor to Adam
Adam	Combines momentum + RMSProp with bias correction	O(|B|·d)	Fast in practice; convergence not always guaranteed in theory	$\eta \approx 0.001$; $\beta_1 \approx 0.9$; $\beta_2 \approx 0.999$	Default for most deep learning; transformers; large models
L-BFGS	Approximate Newton step using limited memory	O(k·|B|·d)	Superlinear (near minimum)	Line search tolerance; history k	Small-medium datasets; fine-tuning; classical ML (scipy.optimize)

n = dataset size. d = parameter dimensionality. |B| = mini-batch size. L-BFGS = Limited-memory Broyden-Fletcher-Goldfarb-Shanno. Sources: Ruder (2016); Kingma and Ba (2014); Nocedal and Wright (2006).

1.4 Information Theory: Measuring What a Model Learns

The connection between probability theory and information theory, established by Shannon in 1948, provides machine learning with a powerful set of tools for measuring the quality and efficiency of learned representations. Three concepts from information theory are of particular importance in the machine learning context: entropy, mutual information, and the Kullback-Leibler divergence.

The entropy of a random variable X, denoted H(X), is the average number of bits required to encode an observation drawn from the distribution of X. A distribution concentrated on a single value has zero entropy — knowing the distribution tells us exactly what to expect, and no information is gained by observation. A uniform distribution over K values has maximum entropy of $\log_2 K$ bits — each observation is a complete surprise. In the context of decision trees, the entropy of the class labels at a node measures the impurity of that node: a node where all examples belong to the same class has zero entropy; a node where examples are equally distributed among all classes has maximum entropy. The information gain criterion for splitting a decision tree node is precisely the reduction in entropy achieved by a particular split.

Mutual information I(X; Y) measures the amount of information that one random variable conveys about another. Formally, it is defined as the KL divergence from the joint distribution p(X, Y) to the product of the marginals p(X)p(Y), which would be the joint distribution if X and Y were independent.

$I(X; Y) = 0$ if and only if X and Y are statistically independent; its maximum value is determined by the marginal entropies of X and Y. Mutual information is the most general measure of statistical dependence between two variables, and it is the theoretical basis for feature selection methods that seek to include, in a model's input, only those features that are genuinely informative about the target variable.

The Kullback-Leibler divergence from distribution q to distribution p, denoted $KL(p\|q)$, is defined as the expected log-ratio of p to q under p. It is not symmetric — $KL(p\|q) \neq KL(q\|p)$ in general — and it is not a metric in the mathematical sense, but it is always non-negative, equalling zero if and only if $p = q$. The practical importance of KL divergence in machine learning stems from its role as the theoretical foundation of maximum likelihood estimation: minimising the cross-entropy loss is equivalent to minimising the KL divergence from the empirical data distribution to the model distribution. This equivalence reveals that every time a machine learning engineer trains a model by minimising cross-entropy loss, she or he is performing a KL projection — fitting the model distribution as close as possible, in information-theoretic terms, to the distribution of the training data.

1.5 From Mathematics to Practice: A Worked Example

The preceding sections have introduced the mathematical concepts that underlie machine learning in a general and somewhat abstract manner. It is pertinent, at this point, to illustrate their interconnection through a concrete worked example, chosen from the Indian agricultural context in which the present author believes the reader will find both relevance and recognition.

Consider the problem of predicting the yield of wheat — measured in quintals per hectare — from seven agronomic and

meteorological features: soil moisture content, average daily temperature during the growing season, total rainfall during the growing season, soil nitrogen content, sowing date (expressed as day of year), crop variety (encoded as a numerical index), and the district in which the field is located (encoded as a district-level average soil quality index). The Indian Council of Agricultural Research (ICAR) maintains datasets of this kind through its network of Krishi Vigyan Kendras, and versions of this prediction problem have been studied by researchers at Punjab Agricultural University, the Directorate of Wheat Research in Karnal, and several IIT agricultural engineering departments.

This prediction problem is a supervised regression problem: we have a set of training examples, each consisting of the seven feature measurements and the observed yield for a particular field and season, and we wish to learn a function from the feature space to the yield space that generalises to new fields and new seasons. Linear regression — the simplest non-trivial supervised learning algorithm — models the yield as a linear combination of the features plus a Gaussian noise term, and estimates the coefficients of this linear combination by minimising the mean squared error over the training data. The solution to this minimisation, which can be derived by setting the gradient of the loss with respect to the coefficients to zero, has the closed form:

$$\theta^{*} = (X^{T}X)^{-1}X^{T}y$$

In this expression, X is the matrix of features (with one row per training example), y is the vector of observed yields, and θ* is the vector of estimated regression coefficients. This expression — called the normal equations — requires inverting the matrix $X^{T}X$, which is a d × d matrix where d is the number of features. Each of the mathematical concepts introduced in the preceding sections

makes its appearance here: X^TX is a matrix product; its inverse is computed using the tools of linear algebra; the coefficients θ^* are the maximum likelihood estimates under the Gaussian noise assumption, connecting to probability theory; and the minimisation of the mean squared error is an optimisation problem whose solution exploits the convexity of the quadratic loss function.

Whether the linear regression model will make good predictions for wheat yield depends on whether the relationship between the features and the yield is, in fact, approximately linear. If there are important non-linear interactions — between rainfall and temperature, for example, or between soil nitrogen and sowing date — then linear regression will systematically underfit the data, producing predictions that are too far from the true yields even on the training data. This is the bias problem; it motivates the more powerful, non-linear algorithms treated in Chapters 3 through 7. The complementary concern — fitting so closely to the training data that the model captures the noise as well as the signal, producing predictions that are accurate on training data but poor on new data — is the variance problem, and it is the subject of Chapter 2.

Learning Theory

Generalisation, Bias-Variance, and the PAC Framework

Chapter 2

Learning Theory — Generalisation, Bias-Variance, and the PAC Framework

There is a story — perhaps apocryphal, but instructive nonetheless — about a group of researchers who trained a neural network to classify images of military vehicles. The network achieved near-perfect accuracy on both training and test images and was deemed ready for deployment. A subsequent investigation revealed that all the training images of tanks had been photographed on overcast days, while the non-tank images had been photographed in sunshine. The network had not learned to recognise tanks; it had learned to recognise overcast skies. It generalised perfectly to images with the same lighting conditions as the training set and would have failed entirely on a sunny day in the field.

This story, variously attributed to different researchers at different times, is almost certainly embellished. But the phenomenon it illustrates is real, has been documented in numerous serious settings, and is perhaps the single most important challenge in supervised machine learning: the gap between performance on the data one has and performance on the data one will encounter. This gap — the generalisation gap — is the central subject of learning theory, the theoretical branch of machine learning that asks not merely whether algorithms work in practice, but why they work, when they fail, and what guarantees of performance we can provide before deploying them.

2.1 The Learning Problem: A Formal Statement

Before examining the conditions under which a machine learning algorithm generalises, it is necessary to state the learning problem with sufficient precision to permit mathematical analysis. The formulation presented here, due in its essential features to Vladimir Vapnik and Alexey Chervonenkis and refined by Leslie Valiant, provides the theoretical foundation for the results that follow.

Let $\mathcal{X}$ denote the input space — the set of all possible instances that a learning algorithm might encounter. For image classification, $\mathcal{X}$ might be the space of all 224×224 pixel images; for credit risk assessment, it might be the space of all possible combinations of the features used to describe a loan applicant. Let $\mathcal{Y}$ denote the output space — the set of possible labels or predictions. For binary classification, $\mathcal{Y} = \{0, 1\}$; for multiclass classification, $\mathcal{Y} = \{1, 2, \ldots, K\}$; for regression, $\mathcal{Y} = \mathbb{R}$.

We assume that there exists an unknown probability distribution D over the product space $\mathcal{X} \times \mathcal{Y}$, from which training examples are drawn independently. This distribution encodes both the distribution of inputs that the algorithm will encounter in deployment and the conditional distribution of labels given inputs. The learning algorithm receives a training set $S = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ consisting of n examples drawn independently from D, and must use this training set to produce a hypothesis $h: \mathcal{X} \to \mathcal{Y}$ that performs well on future examples drawn from the same distribution D.

The error of a hypothesis h with respect to distribution D is defined as the probability that h makes an incorrect prediction on a randomly drawn example:

$$\mathrm{err}_D(h) = \Pr_{(x,y)\sim D}[h(x) \neq y]$$

This is the true error — the quantity we ultimately care about — but it is inaccessible to us, because D is unknown. What we can compute is the empirical error — the fraction of training examples on which h makes incorrect predictions:

$$\mathrm{err}_S(h) = (1/n) \sum_{i=1}^{n} \mathbb{1}[h(x_i) \neq y_i]$$

The fundamental challenge of learning theory is to bound the gap between the true error and the empirical error — to provide guarantees of the form: with high probability over the random choice of training set S, the true error of the hypothesis produced by the learning algorithm is not much larger than its empirical error. Without such guarantees, empirical accuracy on a training set tells us very little about how the algorithm will perform in deployment.

2.2 The Bias-Variance Trade-off

The bias-variance decomposition is one of the most illuminating analytical tools in machine learning, providing a precise account of the two competing sources of error in a supervised learning algorithm and of the trade-off between them that is, in the classical setting, unavoidable.

Consider a regression problem in which the true relationship between the input x and the output y is given by $y = f(x) + \varepsilon$, where ε is noise with mean zero and variance σ^2. A learning algorithm A, given a training set S drawn from distribution D, produces a hypothesis h_S. The expected squared error of h_S at a test point x — where the expectation is taken over the random choice of training sets S of the given size — can be decomposed as:

$$\mathbb{E}_S[(h_S(x) - f(x))^2] = \mathrm{Bias}^2[h_S(x)] + \mathrm{Var}[h_S(x)] + \sigma^2$$

The bias term measures the systematic error introduced by the learning algorithm's inductive bias — the assumptions encoded in the hypothesis class from which it selects hypotheses. An algorithm that always produces linear hypotheses will have high bias when the true function f is non-linear, because no linear function can accurately approximate a non-linear one regardless of how much data is provided. The bias is a property of the algorithm and the problem, not of a specific training set.

The variance term measures the sensitivity of the learning algorithm's output to the specific training set it receives. An algorithm with high variance produces very different hypotheses on different training sets, each of which fits its own training data well but fails to agree with the others in regions where the training data is sparse. High variance is the signature of overfitting: the algorithm has fit the noise in the training data as well as the signal, producing a hypothesis that captures idiosyncrasies of the particular training set rather than the underlying pattern.

The noise term σ^2 is irreducible: it represents the inherent unpredictability of the target variable that no learning algorithm can eliminate, regardless of how much data it receives or how good its hypothesis class is. In the context of the crop yield prediction problem introduced in Chapter 1, the irreducible noise arises from factors affecting yield that are not captured in the seven features: micro-variations in soil quality within a field, pest pressure that was not recorded, and the precise timing of irrigation events. No model can predict these effects, because they are not in the input.

The classical bias-variance trade-off follows from the observation that the bias and variance of a learning algorithm move in opposite directions as the complexity of its hypothesis class increases. A simple hypothesis class — say, the class of linear functions — has low variance (all training sets produce similar

linear hypotheses) but potentially high bias (linear functions may not capture the true relationship). A complex hypothesis class — say, the class of degree-100 polynomials — has low bias (it can approximate almost any function) but high variance (small changes in the training data can produce dramatically different polynomial fits). The art of model selection, which is the subject of Chapter 5, is the art of finding the hypothesis class whose total error — the sum of squared bias and variance — is minimised for the available training data.

Table 2.1 Bias-Variance Trade-off Across Algorithm Families

Algorithm	Bias (typical)	Variance (typical)	Overfitting Risk	Underfitting Risk	Mitigation Strategy
Linear regression (unregularised)	High (if true f is non-linear)	Low	Low	High	Feature engineering; polynomial features; kernel methods
Ridge regression (L2)	Slightly higher than OLS	Lower than OLS	Low	Moderate	Tune regularisation strength λ via cross-validation
Decision tree (full depth)	Low	Very High	Very High	Very Low	Pruning; depth limit; ensemble (Random Forest, Gradient Boosting)
k-Nearest Neighbours (k=1)	Low	High	High	Very Low	Increase k; normalise features; cross-validate k

k-Nearest Neighbours (large k)	High	Low	Low	Moderate	Cross-validate k; use distance weighting
Support Vector Machine (RBF kernel)	Low–Medium	Medium	Moderate	Low	Tune C and γ via cross-validation; standardise features
Random Forest (large ensemble)	Low	Lower than single tree	Low–Moderate	Very Low	Tune n_estimators; max_features; max_depth
Deep Neural Network (large)	Very Low	High (without regularisation)	High	Very Low	Dropout; BatchNorm; weight decay; data augmentation; early stopping

Bias and variance are properties of the algorithm class on a given problem type; they are not fixed quantities for all problems. OLS = Ordinary Least Squares. RBF = Radial Basis Function. The entries reflect typical behaviour on moderately complex supervised learning problems with sufficient (but not unlimited) training data. Sources: Hastie et al. (2009); Geman et al. (1992); Friedman (1997).

2.3 The PAC Learning Framework

Probably Approximately Correct (PAC) learning, introduced by Leslie Valiant in his seminal 1984 paper A Theory of the Learnable, provides the first rigorous framework for answering the question: how much data does a learning algorithm need to achieve a given level of accuracy? The elegance of the PAC framework lies in its simultaneous modesty — it asks only for approximate correctness with high probability, not for exact correctness with certainty — and its generality: it applies to any hypothesis class

and any target function, subject to conditions that we shall make precise.

A hypothesis class H is said to be PAC-learnable if there exists an algorithm A and a polynomial function poly(·) such that for any distribution D over X, any target function f, and any accuracy parameter $\varepsilon > 0$ and confidence parameter $\delta > 0$, when A is given $m \geq \text{poly}(1/\varepsilon, 1/\delta, n, \text{size}(f))$ training examples drawn independently from D, A outputs with probability at least $1 - \delta$ a hypothesis h such that $\text{err}_D(h) \leq \varepsilon$. The polynomial requirement on the sample complexity ensures that the data requirement grows at a manageable rate as accuracy requirements tighten.

The finiteness of the hypothesis class $H = \{h_1, h_2, ..., h_k\}$ provides the simplest and most instructive version of the PAC analysis. By a union bound argument, the probability that any hypothesis in H has low empirical error but high true error — more than ε above the empirical error — is at most $Ke^{(-2m\varepsilon^2)}$, where m is the number of training examples. Setting this expression less than δ and solving for m, we obtain the sample complexity bound:

$$m \geq (1/2\varepsilon^2) \cdot (\ln|H| + \ln(1/\delta))$$

This bound tells us that, to achieve true error within ε of the empirical error with confidence $1 - \delta$, we need a number of training examples that grows logarithmically with the size of the hypothesis class and logarithmically with the desired confidence. Logarithmic growth in |H| is reassuring: even if the hypothesis class has a million hypotheses, we only need on the order of twenty additional training examples compared to a hypothesis class with one hypothesis.

For infinite hypothesis classes — which include all the parametric model families used in practice — the VC dimension replaces the logarithm of the hypothesis class size in the sample

complexity bound. The VC (Vapnik-Chervonenkis) dimension of a hypothesis class H is the largest number of points that can be shattered by H — that is, the largest number of points for which H can achieve every possible binary labelling. A class of d-dimensional linear classifiers has VC dimension d + 1; a class of degree-2 polynomial classifiers in d dimensions has VC dimension on the order of d^2; a class of all possible binary functions on n points has VC dimension n. The fundamental theorem of statistical learning states that a hypothesis class is PAC-learnable if and only if its VC dimension is finite, providing a complete characterisation of learnability in the binary classification setting.

2.4 The Double Descent Phenomenon: When Classical Theory Breaks Down

The bias-variance trade-off and the PAC learning framework were developed in an era when model complexity was constrained by computational resources and data availability, and when machine learning models were expected to be substantially simpler than the data they modelled. The deep learning revolution of the 2010s produced a striking empirical observation that these classical theories did not predict and initially struggled to explain: large, over-parameterised neural networks — models with far more parameters than training examples — often achieve excellent generalisation performance, even though classical learning theory predicted that such models should overfit catastrophically.

The resolution of this paradox came through the discovery of the double descent phenomenon, documented empirically by Belkin and colleagues in a 2019 paper and studied theoretically in subsequent work. The classical bias-variance trade-off predicts a U-shaped curve of test error as model complexity increases: error decreases as the model gains the capacity to fit the true function,

then increases as the model begins to overfit the training noise. The double descent phenomenon reveals that this U-shaped curve is only the first half of the story. As model complexity continues to increase beyond the interpolation threshold — the point at which the model has exactly enough capacity to fit all the training data — the test error begins to decrease again, eventually reaching levels below the classical minimum.

This second descent occurs because over-parameterised models have many ways to exactly fit the training data, and among these, gradient descent with appropriate initialisation tends to find the smoothest one — the one that interpolates the training data while changing as little as possible in the regions between training points. This implicit bias of gradient descent toward smooth solutions, which is an active area of theoretical investigation, provides an effective form of regularisation that is not present in the classical theory.

The practical implications of double descent for the machine learning practitioner are significant. First, the observation that larger models can generalise better than smaller ones is not a paradox that needs to be explained away — it is a documented empirical regularity that challenges the classical prescription of selecting models at the sweet spot of the bias-variance trade-off. Second, the relationship between model size, dataset size, and generalisation is more complex than the classical theory suggests, and the practitioner who bases model selection decisions entirely on classical intuitions may systematically underestimate the appropriate model size. Third, the computational and environmental cost of training very large models — a concern that the present volume has addressed in its companion volumes on green computing and AI energy — must be weighed against the potential generalisation benefits of scale.

Table 2.2 VC Dimensions of Common Hypothesis Classes

Hypothesis Class	VC Dimension	Intuition	Sample Complexity Implication
Single threshold classifier (1D)	1	Can shatter 1 point; not 2 with same threshold	$\sim 1/\varepsilon^2$ examples for ε-accurate learning
Linear classifier (halfspace) in $\mathbb{R}^d$	$d + 1$	$d+1$ points in general position can be shattered by a hyperplane	$\sim (d+1)/\varepsilon^2$ examples; scales linearly with dimension
Linear regression ($\mathbb{R}^d \to \mathbb{R}$)	$d + 1$	Same as linear classifier for thresholded regression	$\sim (d+1)/\varepsilon^2$ examples; same as classification
Degree-2 polynomial classifier in $\mathbb{R}^d$	$O(d^2)$	Quadratic features; many more shattering configurations	$\sim d^2/\varepsilon^2$ examples; quadratic in dimension
Axis-aligned rectangles in $\mathbb{R}^2$	4	Each side independently determines the rectangle	$\sim 4/\varepsilon^2$ examples; independent of rectangle position
k-NN classifier (any k)	∞ (for k=1)	1-NN can shatter any finite set	No finite PAC sample complexity; consistency requires $m \to \infty$
Neural network (depth L, width W)	$O(LW^2 \log W)$	Exponential in depth; quadratic in width	Must grow with depth and width; practical regularisation essential
All binary functions on $\{0,1\}^n$	2^n	Every labelling achievable; maximum possible VC dim	Exponential sample complexity; not PAC-learnable in practice

The VC dimension of neural networks is an upper bound from Bartlett and Mendelson (2002); tighter bounds remain an active research area, particularly for the over-parameterised regime where double descent applies. k-NN with $k > 1$ has infinite VC dimension for any fixed k but satisfies universal consistency — different from PAC

learnability. Sources: Shalev-Shwartz and Ben-David (2014); Bartlett and Mendelson (2002); Belkin et al. (2019).

2.5 No Free Lunch: The Limits of Inductive Learning

The Wolpert-Macready No Free Lunch (NFL) theorems, published in 1997, establish a result that is both philosophically important and practically instructive. Stated informally, the theorems show that no learning algorithm can outperform any other algorithm when performance is averaged over all possible target functions. Any algorithm that performs well on one class of problems must, by necessity, perform poorly on the complementary class. There is no universally best learning algorithm.

The practical implication of the NFL theorems is not nihilism about the possibility of machine learning but rather a clarification of what machine learning is. Every successful learning algorithm embeds inductive biases — assumptions about the structure of the function it is trying to learn. Linear regression assumes the function is linear; support vector machines with RBF kernels assume smooth decision boundaries; convolutional neural networks assume that useful features can be extracted by spatially local filters applied at multiple scales. These assumptions are what make generalisation possible: they constrain the hypothesis space and prevent the algorithm from fitting the noise in the training data as faithfully as the signal.

The art of machine learning — to the extent that it is an art in addition to a science — lies in the identification of the correct inductive biases for a given problem. When those biases are well-matched to the structure of the problem, learning is efficient and generalisation is reliable. When they are mismatched — as in the tank recognition example that opened the present chapter — the algorithm fits the wrong pattern with perfect confidence. The

theoretical frameworks of the present chapter provide the language for describing this matching process; the experimental practices of Chapter 5 provide the methods for evaluating it; and the domain knowledge that makes the matching possible is the contribution that the human engineer brings to the collaboration between human and machine that machine learning, at its best, represents.

Part II — Supervised Learning

Chapter 3: Linear and Probabilistic Models — Regression, Classification, and Regularisation

Chapter 4: Kernel Methods, SVMs, and Ensemble Learning

Chapter 5: Model Selection, Evaluation, and the Art of Generalisation

Linear and Probabilistic Models

Regression, Classification, and Regularisation

Chapter 3

Linear and Probabilistic Models — Regression, Classification, and Regularisation

In the early months of the COVID-19 pandemic, a team of researchers at All India Institute of Medical Sciences collaborated with data scientists to build a prediction model for patient outcomes. The available data was modest by the standards of modern machine learning: a few hundred patients, each described by twenty or so clinical variables — age, comorbidities, blood oxygen levels, inflammatory markers, days since symptom onset. The team faced a choice that confronts every applied machine learning project: how complex a model is appropriate, given the data available?

A colleague suggested a deep neural network. An older clinician, who had been practising medicine for thirty years and who had seen many enthusiasms come and go, asked a simpler question: which three or four variables, taken together, are most predictive of poor outcome? That question is not the question that a neural network is designed to answer. It is precisely the question that logistic regression, with appropriate regularisation and a careful analysis of its coefficients, can answer with confidence, interpretability, and — given the modest sample size — with generalisability that a neural network could not have provided.

The present chapter is devoted to the algorithms that answer such questions: linear regression, logistic regression, Naive Bayes, and Gaussian discriminant analysis. These are not the most powerful algorithms in the machine learning toolkit. They are, however, the ones that are most often the right answer — and the

ones whose foundations must be thoroughly understood before the more powerful methods of subsequent chapters can be used wisely.

3.1 Linear Regression: From Geometry to Probability

Linear regression is, in the estimation of the present author, the most underappreciated algorithm in all of machine learning. It is introduced in every course and every textbook as a preliminary to more interesting things, and dispatched quickly on the grounds that real data is rarely linear. This dismissal is, in the experience of practitioners, premature. Linear regression with appropriate feature engineering is competitive with much more complex models on a surprisingly wide range of practical problems; its coefficients are interpretable in ways that motivate action; its computational cost is trivial; and its probabilistic foundations, when understood, illuminate the principles that underlie every subsequent algorithm in this volume.

The linear regression model assumes that the target variable y is a linear function of the input features $x \in \mathbb{R}^d$ plus Gaussian noise:

$$y = \theta^T x + \varepsilon, \quad \varepsilon \sim N(0, \sigma^2)$$

In this formulation, $\theta \in \mathbb{R}^d$ is the vector of regression coefficients and σ^2 is the variance of the noise term. The coefficient θ_j associated with feature j represents the expected change in y for a unit increase in x_j, holding all other features constant. This interpretation, which requires no statistical training to understand, is what makes linear regression the preferred tool in contexts where understanding the drivers of the outcome is as important as predicting it — which describes the majority of applications in healthcare, economics, and public policy.

The maximum likelihood estimate of θ, derived by maximising the log-likelihood of the observed targets under the

Gaussian noise assumption, is equivalent to minimising the mean squared error between predictions and observed values. This is the ordinary least squares (OLS) estimator, whose closed form was derived in Chapter 1:

$$\theta\_OLS = (X^T X)^{-1} X^T y$$

The OLS estimator has a number of remarkable theoretical properties. The Gauss-Markov theorem establishes that, among all linear unbiased estimators, OLS has the smallest variance — it is the Best Linear Unbiased Estimator (BLUE). This result holds without requiring the noise to be Gaussian; it requires only that the noise has zero mean, constant variance, and no correlation between observations. The Gaussian assumption is additionally required for the sampling distribution of the estimator to be Gaussian, enabling confidence intervals and hypothesis tests.

The geometry of linear regression is illuminating and worth dwelling upon. The predicted values $\hat{y} = X\hat{\theta}$ lie in the column space of the design matrix X — the subspace of $\mathbb{R}^n$ spanned by the columns of X. The OLS solution is the orthogonal projection of the target vector y onto this column space, which is why the residual vector $y - \hat{y}$ is orthogonal to every column of X. This geometric interpretation provides an immediate intuition for what linear regression does: it finds the element of a given linear subspace that is closest to the target, where 'closest' is measured by Euclidean distance.

3.2 Regularisation: Controlling Complexity Through Priors

The OLS estimator, despite its admirable theoretical properties, performs poorly in two common practical situations: when the number of features d approaches or exceeds the number

of training examples n (rendering the matrix X^TX singular or nearly singular), and when the features are highly correlated with one another (producing a condition number of X^TX that is very large, causing the inversion to amplify small perturbations in the data into large perturbations in the estimates). Regularisation addresses both pathologies by adding a penalty term to the loss function that discourages large coefficient values.

Ridge regression, introduced by Hoerl and Kennard in 1970, adds an L2 penalty on the magnitude of the coefficients:

$$\theta\_ridge = argmin\_\theta \{ ||y - X\theta||^2 + \lambda||\theta||^2 \}$$

The regularisation parameter $\lambda \geq 0$ controls the strength of the penalty: $\lambda = 0$ recovers OLS; large λ shrinks all coefficients toward zero. The ridge regression estimator has a closed form — $(X^TX + \lambda I)^{-1}X^Ty$ — that is always well-defined, even when X^TX is singular, because the addition of λI to X^TX shifts all eigenvalues upward by λ, eliminating any zero eigenvalues. The Bayesian interpretation of ridge regression is immediate and illuminating: ridge regression is equivalent to MAP estimation under a Gaussian prior on θ with mean zero and variance σ^2/λ. The regularisation penalty is not merely a computational convenience; it is the expression of a prior belief that large coefficient values are less plausible than small ones.

Lasso regression, introduced by Tibshirani in 1996, replaces the L2 penalty with an L1 penalty on the absolute value of the coefficients:

$$\theta\_lasso = argmin\_\theta \{ ||y - X\theta||^2 + \lambda||\theta||_1 \}$$

The L1 penalty has a qualitatively different effect from the L2 penalty: rather than uniformly shrinking all coefficients toward zero, it tends to set some coefficients exactly to zero, producing a sparse solution in which only a subset of features receive non-zero

weights. This sparsity-inducing property makes lasso a natural tool for feature selection: the non-zero coefficients in the lasso solution identify the features that the model has selected as most predictive of the target. The Bayesian interpretation of lasso is MAP estimation under a Laplace prior on θ — a prior that assigns higher probability than the Gaussian to values near zero and to values far from zero simultaneously.

The elastic net, proposed by Zou and Hastie in 2005, combines L1 and L2 penalties:

$$\theta\_enet = argmin\_\theta \{ ||y - X\theta||^2 + \lambda_1||\theta||_1 + \lambda_2||\theta||^2 \}$$

It is particularly useful when predictors are correlated: lasso tends to select one predictor from a correlated group and ignore the others (the selection being somewhat arbitrary), while elastic net can select the entire group and share the coefficient weight among them. In the Indian agricultural context — where variables such as soil nitrogen, organic matter, and pH are highly intercorrelated — elastic net regularisation is often the most appropriate choice for regression models of crop yield or soil health outcomes.

3.3 Logistic Regression: The Canonical Binary Classifier

Logistic regression is, despite its name, a classification algorithm rather than a regression algorithm. It models the probability that an input x belongs to class 1 as a sigmoid-transformed linear function of the features:

$$P(y=1 \mid x; \theta) = \sigma(\theta^T x) = 1 / (1 + exp(-\theta^T x))$$

The sigmoid function $\sigma(z) = 1/(1 + e^{-p})$ maps any real number to the interval (0, 1), ensuring that the model output is a valid probability. The decision boundary of logistic regression — the set of inputs for which the model assigns equal probability to both

classes — is the hyperplane $\{x : \theta^T x = 0\}$, which is a linear decision boundary in the feature space. This means that logistic regression, like linear regression, is a linear model; it can only separate classes that are linearly separable in the feature space, or approximately separable with some misclassification allowed.

The parameters θ are estimated by maximising the conditional log-likelihood of the observed labels under the logistic model, which is equivalent to minimising the binary cross-entropy loss:

$$L(\theta) = -\Sigma_i \, [y_i \log \sigma(\theta^T x_i) + (1 - y_i) \log (1 - \sigma(\theta^T x_i))]$$

Unlike linear regression, this optimisation problem has no closed-form solution; it is solved iteratively using gradient descent or Newton's method. The log-odds interpretation of the logistic regression coefficients is of particular value in applied settings: the coefficient θ_j represents the change in the log-odds of class 1 for a unit increase in feature j. In the clinical prediction context described in the opening vignette of the present chapter, a coefficient of +0.8 for serum creatinine level means that each unit increase in serum creatinine multiplies the odds of poor outcome by e^0.8 ≈ 2.2 — a statement that a clinician can evaluate, challenge, and act upon.

The extension of logistic regression to multi-class classification — the softmax regression model, also called multinomial logistic regression — models the probability of each class k as:

$$P(y = k \mid x; \theta) = \exp(\theta_k^T x) \, / \, \Sigma_j \exp(\theta_j^T x)$$

This is the softmax function applied to the vector of class scores, and it reduces to the sigmoid function in the binary case. Softmax regression is the output layer of virtually every neural network classifier, connecting the linear models of the present chapter to the deep learning models of Chapter 7.

3.4 Naive Bayes and Gaussian Discriminant Analysis

Naive Bayes and Gaussian Discriminant Analysis (GDA) are generative classification models: rather than directly modelling the posterior probability $P(y \mid x)$ — as logistic regression does — they model the joint probability $P(x, y) = P(y) \cdot P(x \mid y)$ and use Bayes' theorem to compute the posterior. This distinction between discriminative and generative models has practical consequences for both their data requirements and their behaviour when the model assumptions are violated.

Naive Bayes makes the simplifying assumption that the features are conditionally independent given the class label — that is, $P(x \mid y) = \Pi_j P(x_j \mid y)$. This assumption is almost never exactly true in practice: in a document classification application, for example, the presence of the word 'cricket' makes the presence of the word 'India' more likely, regardless of the document's class. Despite this unrealistic assumption, Naive Bayes classifiers often perform surprisingly well in practice, particularly on high-dimensional problems with limited data. The explanation lies in a variance argument: the Naive Bayes assumption dramatically reduces the number of parameters that must be estimated from data, reducing variance at the cost of some bias, and for many practical problems and sample sizes, the reduction in variance more than compensates for the increase in bias.

In text classification — one of the most commercially important classification tasks in India, given the multilingual nature of Indian digital content — Naive Bayes with a multinomial or Bernoulli feature distribution has been the baseline model of choice for decades. The AI4Bharat project at IIT Madras, which develops natural language processing systems for Indian languages, has used Naive Bayes as a baseline against which more complex models are measured; its simplicity, speed, and

interpretability make it a practical choice for resource-constrained deployment contexts in Indian language computing.

Gaussian Discriminant Analysis assumes that the class-conditional distributions $P(x \mid y = k)$ are multivariate Gaussian, with class-specific means μ_k and either a shared covariance matrix Σ (Linear Discriminant Analysis, LDA) or class-specific covariance matrices Σ_k (Quadratic Discriminant Analysis, QDA). The parameters are estimated by maximum likelihood from the training data, and classification is performed by selecting the class with the highest posterior probability. LDA produces a linear decision boundary, like logistic regression; QDA produces a quadratic one. A well-known theoretical result establishes that when the class-conditional Gaussians are correctly specified and the classes have equal covariance, LDA is the optimal linear classifier — it achieves lower error than any other linear method, including logistic regression, on the population that generated the data.

Table 3.1 Linear and Probabilistic Classification Models — Comparison

Model	Type	Decision Boundary	Assumptions	Key Strength	Key Limitation	Best Applied When
Logistic Regression	Discriminative	Linear hyperplane	Log-linear relationship; i.i.d. samples	Interpretable coefficients; calibrated probabilities	Linear decision boundary only	Interpretability required; moderate dimensions; medical/policy

Ridge-regularised Logistic	Discriminative	Linear hyperplane	Same + spherical prior on θ	Handles correlated/high-dim features well	Still linear boundary	High-dimensional data; feature selection secondary concern
Lasso-regularised Logistic	Discriminative	Linear hyperplane	Same + sparse prior on θ	Automatic feature selection; sparse solution	Unstable with correlated features	Feature selection critical; many irrelevant features expected
Naive Bayes (Gaussian)	Generative	Linear hyperplane (GNB)	Feature conditional independence given class	Fast; few parameters; works well with small data	Independence assumption often violated	Small data; text; document classification; baseline
Naive Bayes (Multinomial)	Generative	Linear in log space	Integer feature counts; conditional independence	Natural for text; highly scalable	Requires non-negative integer features	Document classification; spam filtering; language ID
LDA (Linear Discriminant Analysis)	Generative	Linear hyperplane	Gaussian classes; equal covariance	Optimal linear classifier if assumptions hold; dimensionality reduction	Gaussian assumption often violated	Equal-covariance Gaussian data; also used for dim reduction
QDA (Quadratic Discrimi	Generative	Quadratic surface	Gaussian classes; class-	More flexible boundary than LDA	Many parameters; needs large	Sufficient per-class data; clearly

nant Analysis)			specific covariance		per-class sample size	non-linear boundary needed

GNB = Gaussian Naive Bayes. Sources: Bishop (2006); Murphy (2012); Hastie et al. (2009).

3.5 Case Study: Predicting Crop Disease Risk in India

The present section illustrates the application of the models introduced in this chapter through a case study drawn from Indian agriculture — a domain that simultaneously represents one of the most important application areas for machine learning in the country and one that is often underrepresented in standard textbook treatments, which tend to draw examples from North American or European contexts.

The Indian Council of Agricultural Research, through its network of Krishi Vigyan Kendras and the National Research Centre for Integrated Pest Management, maintains databases of crop disease incidence records matched to meteorological and agronomic variables. A representative problem of practical importance is the prediction of the probability of disease outbreak — specifically blast disease in rice, which is caused by the fungal pathogen Magnaporthe oryzae and is responsible for substantial yield losses across West Bengal, Andhra Pradesh, and Tamil Nadu — given features available at planting time or early in the season: variety susceptibility index, average relative humidity in the ten days preceding the observation, average temperature range in the same period, leaf wetness duration, and nitrogen application rate.

This is a binary classification problem: the label is whether an outbreak occurs within the subsequent two-week window. The training data, derived from district-level records in collaboration with the Directorate of Rice Research in Hyderabad, contains

approximately eight hundred observations over a twelve-year period, with roughly 30 per cent positive (outbreak) cases. Given this modest sample size and the interpretability requirements of extension officers who will use the predictions to advise farmers, logistic regression with L2 regularisation is the most appropriate modelling choice.

The estimated logistic regression coefficients, after standardising all features to zero mean and unit variance, reveal that leaf wetness duration is the strongest predictor of outbreak risk, with a positive coefficient approximately twice the magnitude of the next most important variable, relative humidity. Nitrogen application rate has a positive association with outbreak risk, consistent with the agronomic literature's finding that nitrogen-rich tissue is more susceptible to Magnaporthe infection. Temperature range — larger daily temperature swings are associated with lower outbreak risk — has a negative coefficient. These findings are consistent with established plant pathological knowledge, which provides validation that the model has captured meaningful biology rather than spurious statistical associations.

It is pertinent to observe, in this context, that the interpretability of logistic regression's coefficients is not merely a convenient feature — it is an epistemic necessity in agricultural advisory applications. An extension officer advising a farmer to adjust their nitrogen application rates based on a model's recommendation must be able to explain, in terms the farmer can understand and evaluate, why the model makes that recommendation. A decision that cannot be explained cannot be trusted, and a decision that cannot be trusted will not be implemented. This is the case, in the present author's experience, for the majority of high-stakes machine learning applications in the Indian public sector, and it argues for the careful and deliberate use

of interpretable models wherever they are adequate to the predictive task.

Kernel Methods, SVMs, and Ensemble Learning

Chapter 4

Kernel Methods, SVMs, and Ensemble Learning

In 2007, a team of researchers at the Indian Institute of Technology Bombay, working in collaboration with the Tata Memorial Centre in Mumbai, published a study on the classification of histopathological images of oral cancer biopsies. The dataset contained images of tissue samples from patients at various stages of oral submucous fibrosis — a precancerous condition with high prevalence in India, associated with the widespread use of betel nut products — and the task was to classify each image as normal, mildly dysplastic, moderately dysplastic, or severely dysplastic, based on features extracted from the cellular morphology.

The researchers evaluated several classifiers. Logistic regression achieved modest accuracy. A support vector machine with a radial basis function kernel achieved substantially higher accuracy. The difference, they observed, was not primarily due to the SVM's margin maximisation objective — it was due to the kernel: the RBF kernel implicitly mapped the original low-dimensional feature vectors into a very high-dimensional space in which the class boundaries, which were highly non-linear in the original space, became approximately linearly separable. The kernel trick had turned a problem that a linear classifier could not solve into one that it could.

This observation — that the choice of kernel is often the most important modelling decision in a kernel method, and that the right kernel can transform an apparently intractable classification problem into a tractable one — is the central insight that the present chapter is designed to convey.

4.1 The Kernel Trick: Infinite-Dimensional Feature Spaces

A recurring theme in the preceding chapters has been the limitation of linear models: their decision boundaries and prediction functions are restricted to hyperplanes and linear functions, which may be inadequate for problems with complex, non-linear structure. One approach to overcoming this limitation is to explicitly construct non-linear features from the original inputs — polynomial features, interaction terms, trigonometric features — and then apply a linear algorithm to the expanded feature space. This approach works but becomes computationally prohibitive as the number of original features d grows: the feature vector for degree-2 polynomial features has $O(d^2)$ components; for degree-3 features, $O(d^3)$ components. For the image classification problems that are among the most important applications of machine learning, where d is the number of pixels and easily exceeds ten thousand, explicit polynomial feature expansion is entirely impractical.

The kernel trick circumvents this difficulty by exploiting the following observation: many learning algorithms — including the support vector machine and kernel regression — require access to the training data only through inner products between pairs of examples. If we denote by $\varphi(x)$ the mapping of input x to a feature space, then the inner product $\langle \varphi(x_i), \varphi(x_j) \rangle$ between two mapped examples can often be computed efficiently as a function of the original inputs, without ever explicitly constructing the high-dimensional feature vectors $\varphi(x_i)$ and $\varphi(x_j)$.

A kernel function $k(x, x')$ is a function that computes this inner product: $k(x, x') = \langle \varphi(x), \varphi(x') \rangle$ for some feature mapping φ. Mercer's theorem characterises which functions are valid kernels: a symmetric function $k(x, x')$ is a valid kernel if and only if the

matrix K with entries $K_{ij} = k(x_i, x_j)$ is positive semidefinite for any finite set of points. The practical consequence of this characterisation is that any positive semidefinite function of two inputs can be used as a kernel, even if the corresponding feature space is infinite-dimensional — as in the case of the radial basis function kernel, for which the feature space is an infinite-dimensional Hilbert space.

The radial basis function (RBF) kernel, also called the Gaussian kernel, is defined as:

$$k(x, x') = \exp(-\|x - x'\|^2 / (2\sigma^2))$$

It measures the similarity between two inputs as a decreasing function of their Euclidean distance, with the bandwidth parameter σ controlling the rate of decay. Inputs that are close together have kernel values near 1; inputs that are far apart have kernel values near 0. The RBF kernel corresponds to an infinite-dimensional feature space, and its use converts a linear algorithm into one that can represent decision boundaries of arbitrary complexity — subject, of course, to the constraint of not overfitting, which is managed through regularisation.

The polynomial kernel $k(x, x') = (x^{\mathsf{T}}x' + c)^{\wedge}p$ provides another widely used alternative, implicitly computing all polynomial features up to degree p. It is particularly natural for text and biological sequence data, where the inputs are discrete objects and the polynomial kernel corresponds to matching p-grams in a computationally efficient manner.

4.2 Support Vector Machines: Maximum Margin Classification

The support vector machine, developed by Vladimir Vapnik and colleagues at Bell Laboratories in the 1990s and building on

the earlier theory of structural risk minimisation, is the most theoretically motivated of the standard supervised classification algorithms. Its central insight — that among all decision hyperplanes that correctly classify the training data, the one with the largest margin (the distance between the hyperplane and the nearest training examples of either class) will generalise best to unseen data — connects the geometric concept of margin to the statistical concept of generalisation in a way that is both intuitive and formally justified.

For a binary classification problem with linearly separable classes, the hard-margin SVM finds the maximum-margin separating hyperplane by solving the optimisation problem:

$$\text{minimise } ||w||^2 \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 1 \quad \text{for all } i$$

The training examples that lie exactly on the margin boundaries — the examples for which $y_i(w^T x_i + b) = 1$ — are called support vectors, and they are the only training examples that determine the solution. All other training examples could be removed without changing the optimal hyperplane. This property is not merely mathematically elegant; it has practical implications for computational efficiency and for the stability of the learned boundary: the SVM boundary is determined by the most difficult examples, the ones nearest the boundary, rather than by the entire training dataset.

The soft-margin SVM, necessary for the common case where the classes are not linearly separable, introduces slack variables $\xi_i \geq 0$ that allow some examples to be misclassified, penalising misclassifications in the objective:

$$\text{minimise } ||w||^2 + C \cdot \Sigma_i \, \xi_i \quad \text{subject to} \quad y_i(w^T x_i + b) \geq 1 - \xi_i \quad \text{and} \quad \xi_i \geq 0$$

The regularisation parameter C controls the trade-off between maximising the margin and minimising misclassification: large C produces a smaller margin with fewer training errors; small C produces a larger margin that tolerates more misclassification. The kernel SVM is obtained by replacing the inner products in the dual formulation of the SVM optimisation with kernel function evaluations, enabling non-linear decision boundaries through the kernel trick.

4.3 Decision Trees: Recursive Partitioning

Decision trees partition the feature space into rectangular regions through a sequence of axis-aligned binary splits, assigning a class label or predicted value to each region. Their principal attraction is interpretability: a decision tree with modest depth can be printed and read by a domain expert, who can verify whether the splits the algorithm has discovered are consistent with prior knowledge and whether the tree's decisions can be defended to stakeholders. This interpretability is of particular value in Indian public sector applications — in healthcare, banking, and public administration — where algorithmic decisions may be challenged by citizens and must be explainable to regulators.

The CART (Classification and Regression Trees) algorithm grows a decision tree by recursively selecting the feature and split point that maximally reduce the impurity of the resulting child nodes. For classification, impurity is measured by the Gini index or information gain (entropy reduction); for regression, by the reduction in mean squared error. The tree is grown until it either perfectly classifies all training examples or reaches a specified maximum depth, and then pruned using cost-complexity pruning to remove subtrees that do not substantially improve performance on a held-out validation set.

A fully grown, unpruned decision tree has zero training error — it can shatter any training set whose examples have distinct feature vectors — but very high variance, as was noted in the bias-variance analysis of Chapter 2. The solution to this variance problem is not to simply limit tree depth, which introduces bias, but to combine many high-variance trees in a way that averages out their individual errors. This is the motivation for ensemble methods, to which the present section now turns.

4.4 Ensemble Methods: Bagging, Random Forests, and Gradient Boosting

The ensemble principle is one of the most powerful ideas in machine learning, and its practical success in competition-driven benchmarking — ensemble methods have won the majority of Kaggle competitions and top positions on most tabular data leaderboards — reflects a theoretical truth: the average of many imperfect models is often better than any single model, provided the individual models make different errors.

Bagging (Bootstrap Aggregating), introduced by Breiman in 1994, trains multiple instances of the same base learner on different bootstrap samples of the training data — samples drawn with replacement — and combines their predictions by majority vote (for classification) or averaging (for regression). Bagging reduces variance by averaging over the sources of randomness in the base learner's training process. For base learners with high variance and low bias — such as fully grown decision trees — bagging can dramatically improve generalisation without increasing bias.

Random Forests, introduced by Breiman in 2001, extend bagging by adding an additional source of randomisation: at each split in each tree, rather than considering all features, the algorithm randomly selects a subset of features and chooses the best split

among only those features. This feature subsampling decorrelates the individual trees, reducing the variance of the ensemble beyond what bagging alone achieves. Random Forests are among the most robust and widely applicable algorithms in the machine learning toolkit: they require little hyperparameter tuning, handle high-dimensional inputs naturally, provide built-in estimates of feature importance, and are rarely the wrong choice as a baseline model for structured data.

Gradient boosting, introduced by Friedman in 2001, takes a fundamentally different approach to ensemble construction: rather than combining weak learners trained independently on random subsets of data, it trains them sequentially, with each new learner fitting the residual errors of the ensemble so far. The gradient boosting framework casts this sequential construction as gradient descent in function space: each new learner is trained to predict the negative gradient of the loss with respect to the current ensemble's predictions. The result is an ensemble that systematically reduces both bias and variance as the number of learners grows, subject to the constraint of not overfitting — which is managed through the learning rate (shrinkage) and tree depth.

XGBoost, introduced by Chen and Guestrin in 2016, and its successors LightGBM and CatBoost, are engineering-optimised implementations of gradient boosting that have dominated structured data machine learning for nearly a decade. Their performance on problems of the kind encountered in Indian applications — credit scoring for banks and NBFCs, fraud detection for payment systems, demand forecasting for FMCG companies, risk assessment in insurance — is consistently strong, and they are the algorithm of first choice for practitioners working with tabular data in the absence of compelling reasons to try alternatives.

Table 4.1 Kernel Functions — Properties, Feature Spaces, and Use Cases

Kernel	Formula $k(x, x')$	Implicit Feature Space	Key Hyperparameter(s)	Strengths	Best Applied When
Linear	$x^T x'$	Original input space $\mathbb{R}^d$	None	Equivalent to linear SVM; fast; interpretable	Linearly separable data; text classification with TF-IDF features
Polynomial	$(x^T x' + c)^p$	All monomials up to degree p	Degree p; offset c	Captures feature interactions; natural for count data	NLP n-gram matching; biological sequence analysis
Radial Basis Function (RBF/Gaussian)	$\exp(-\|\|x-x'\|\|^2/2\sigma^2)$	Infinite-dim. Gaussian RKHS	Bandwidth σ (or $\gamma = 1/2\sigma^2$)	Universal approximator; smooth decision boundaries	Unknown structure; moderate dimensions; first kernel to try
Sigmoid (tanh)	$\tanh(\alpha x^T x' + c)$	Not a valid kernel for all α, c	Scale α; shift c	Inspired by neural network activations	Rarely preferred; use RBF or neural network instead
Laplacian (L1 RBF)	$\exp(-\|\|x-x'\|\|_1/\sigma)$	Infinite-dim. Laplace RKHS	Bandwidth σ	Less sensitive to outliers	Robust classification; medical

				than Gaussian kernel	imaging; non-Gaussian noise
String kernel	Counts shared substrings	Space of all substrings	Length of substrings considered	Directly operates on raw sequences; no feature engineering	Protein homology; DNA analysis; text at character level
Graph kernel	Counts shared subgraph patterns	Space of graph substructures	Subgraph type; depth	Directly compares graph-structured data	Molecular property prediction; social network classification

RKHS = Reproducing Kernel Hilbert Space. Not all sigmoid kernel parameters produce a valid (positive semidefinite) kernel; use requires verification for specific parameter settings. Sources: Schölkopf and Smola (2002); Shawe-Taylor and Cristianini (2004).

Table 4.2 Ensemble Methods — Comparison of Construction Philosophy and Performance Profile

Method	Construction	Base Learner	Variance Reduction	Bias Reduction	Overfitting Risk	Key Hyperparameter(s)
Bagging	Parallel; bootstrap samples	Any high-variance model	High (averaging)	None (same bias as base)	Low	Number of estimators; base learner params
Random Forest	Parallel; bootstrap + feature subsampling	Decision trees (full	Very High	None	Low	n_estimators; max_features; max_depth

		depth)				
Extra-Trees	Parallel; random splits (no bootstrap)	Decision trees (full depth)	Very High (more random)	Slightly higher than RF	Very Low	n_estimators; max_features; min_samples_split
AdaBoost	Sequential; reweights misclassified samples	Shallow decision stumps	Moderate	High	Moderate (sensitive to outliers)	n_estimators; learning rate
Gradient Boosting (sklearn)	Sequential; fits residuals	Shallow decision trees	Moderate	High	Moderate	n_estimators; learning rate; max_depth
XGBoost	Sequential; regularised gradient boosting	Shallow trees with L1/L2 reg	Moderate	Very High	Low (built-in regularisation)	n_estimators; learning rate; max_depth; lambda/alpha
LightGBM	Sequential; leaf-wise growth; histogram features	Shallow trees (leaf-wise)	Moderate	Very High	Low	num_leaves; learning rate; min_data_in_leaf; feature_fraction

XGBoost and LightGBM are the preferred implementations of gradient boosting for most practical applications. Extra-Trees (Extremely Randomised Trees) differ from Random Forests in that they do not use bootstrap sampling and choose split points randomly rather than optimally, producing higher variance reduction at the cost of slightly higher bias. Sources: Breiman (1996, 2001); Friedman (2001); Chen and Guestrin (2016); Ke et al. (2017).

Model Selection, Evaluation, and the Art of Generalisation

Chapter 5

Model Selection, Evaluation, and the Art of Generalisation

In 2019, a startup in Bengaluru received a series A investment to deploy a machine learning model for predicting customer churn in a telecommunications company. The model had been developed by a team of skilled data scientists over six months, and it reported an accuracy of 96.2 per cent on the test set. The founders presented this figure prominently in their investor pitch deck. Within three months of deployment, the business team had lost confidence in the model: it was flagging large numbers of customers as high-churn-risk who subsequently remained loyal, and missing many customers who were genuinely at risk of leaving.

A post-deployment analysis revealed several things. The dataset had been severely imbalanced — approximately 4 per cent of customers had churned, and 96 per cent had not. A classifier that predicted ‘no churn’ for every customer would achieve exactly 96 per cent accuracy on this dataset. The model had learned something, but not much more than this trivial baseline, and the metric used to evaluate it — accuracy — had completely failed to reveal this. Moreover, the test set had been constructed by randomly sampling from the full dataset, which included customers from periods before a major competitor had entered the market; the model had been evaluated on data that was, in an important sense, no longer representative of the deployment environment.

This episode, which is representative of a pattern the present author has observed in numerous industry projects, illustrates the central importance of the topic addressed in the present chapter:

the rigorous evaluation of machine learning models. Getting predictions right is not enough. One must also know when predictions are wrong, how wrong they are, whether the evaluation methodology is trustworthy, and whether a good evaluation score in the laboratory will translate to good performance in the field. These are not peripheral concerns to be addressed after the interesting modelling is done; they are the foundation on which all confidence in a model's real-world utility must rest.

5.1 The Evaluation Infrastructure: Train, Validation, and Test Sets

The distinction between training data, validation data, and test data is one of the most fundamental in applied machine learning, and it is one that is, in practice, violated with surprising frequency — sometimes out of convenience, sometimes out of ignorance, and occasionally out of a desire to report numbers that are better than the model genuinely deserves. The present section establishes the purpose of each data partition and the conditions under which each is the appropriate source of evaluation evidence.

The training set is the data on which the model's parameters are optimised. By definition, the model has been tuned to perform well on this data — it has, in the statistical sense, seen it — and training set performance is therefore an optimistic estimate of generalisation performance. Reporting only training set accuracy is essentially meaningless for evaluating a model's utility; it measures the model's ability to memorise, not to generalise. Any model with sufficient capacity can achieve near-zero training error; the question is what happens on data the model has not seen.

The validation set serves as an estimate of generalisation performance during the development process — it is used to make model selection decisions, to choose between alternative

algorithms, to tune hyperparameters, and to decide when to stop training. Because validation set performance is used to make decisions, the model is implicitly being adapted to perform well on the validation set as well as the training set, and repeated evaluation and comparison on the same validation set will eventually produce a model that is overfitted to the validation set. This phenomenon — sometimes called ‘validation set leakage’ or ‘adaptive overfitting’ — is a real risk in long projects with many modelling decisions, and it is the reason why a held-out test set must remain entirely unused until the final evaluation.

The test set is used exactly once, at the conclusion of model development, to provide the final unbiased estimate of the model’s generalisation performance. Any use of the test set during model development — even merely inspecting the test set distribution to guide data preprocessing decisions — constitutes a form of information leakage that compromises the integrity of the evaluation. The test set error is the only honest estimate of how the model will perform in deployment, provided the test set is genuinely representative of the deployment distribution.

Cross-validation provides an alternative to a single fixed split when the dataset is too small to partition into training, validation, and test sets with adequate statistical power in each. In k-fold cross-validation, the data is partitioned into k equal-sized folds; the model is trained k times, each time using $k-1$ folds for training and the remaining fold for validation; the k validation scores are averaged to produce the cross-validation estimate. The choice of k involves a variance-bias trade-off: larger k (up to the leave-one-out extreme of $k = n$) produces estimates with lower bias but higher variance; smaller k produces less variable but potentially more biased estimates. $k = 5$ or $k = 10$ are conventional choices that balance these concerns well for moderate-sized datasets.

Stratified k-fold cross-validation preserves the class distribution in each fold — each fold contains approximately the same fraction of each class as the full dataset. This is essential for imbalanced datasets, where random sampling may place all examples of the minority class in a single fold, producing folds that are unrepresentative of the true data distribution. Time-series cross-validation — in which only past data is used to train models that are validated on subsequent time periods — is the appropriate approach whenever the training and deployment data are temporally ordered, as is the case for stock price prediction, demand forecasting, and customer behaviour modelling.

5.2 Classification Metrics: Beyond Accuracy

The bankruptcy of accuracy as a primary evaluation metric for imbalanced classification problems was illustrated in the opening vignette of the present chapter. The present section provides a systematic treatment of the metrics that are actually informative for binary classification problems, and guidance on their appropriate use.

The confusion matrix is the starting point for all classification evaluation. For a binary classifier, it is a 2×2 table that counts the four possible combinations of predicted and actual class: True Positives (TP, correctly classified as positive), True Negatives (TN, correctly classified as negative), False Positives (FP, incorrectly classified as positive), and False Negatives (FN, incorrectly classified as negative). Every scalar metric for binary classification is a function of these four counts, and the confusion matrix itself is the most informative summary of a classifier's performance — inspecting it directly, rather than relying on a single scalar summary, is the mark of a careful practitioner.

Precision is the fraction of positive predictions that are correct: TP / (TP + FP). It measures the cost of false alarms — the fraction of the time that when the model says 'positive,' it is right. Recall (also called sensitivity or the true positive rate) is the fraction of actual positives that are correctly identified: TP / (TP + FN). It measures the cost of missed detections — the fraction of actual positives that the model catches. In the crop disease prediction context introduced in Chapter 3, high recall means that most actual disease outbreaks are flagged by the model; high precision means that most flags correspond to actual outbreaks. The relative importance of precision and recall depends on the costs of false alarms versus missed detections — a question that requires domain knowledge to answer.

The F1 score, defined as the harmonic mean of precision and recall — 2 · Precision · Recall / (Precision + Recall) — provides a single-number summary that balances both concerns. The harmonic mean penalises extreme imbalances between precision and recall more severely than the arithmetic mean: a classifier with precision 0.9 and recall 0.1 achieves F1 = 0.18, not the arithmetic average of 0.5, reflecting that a recall of 0.1 is unacceptably low regardless of how precise the few positive predictions happen to be.

The Area Under the ROC Curve (AUC-ROC) is a metric that evaluates the classifier's discrimination ability across all possible classification thresholds. The ROC curve plots the true positive rate against the false positive rate as the classification threshold varies from 0 to 1. A perfect classifier achieves an AUC of 1.0; a random classifier achieves an AUC of 0.5. The AUC has a useful probabilistic interpretation: it is equal to the probability that the classifier assigns a higher score to a randomly chosen positive example than to a randomly chosen negative example. For

imbalanced datasets, however, the AUC can be misleadingly optimistic — the precision-recall curve and its area (AUPRC) are more informative when the positive class is rare.

The Matthews Correlation Coefficient (MCC), defined as:

$$MCC = (TP \cdot TN - FP \cdot FN) / \sqrt{((TP+FP)(TP+FN)(TN+FP)(TN+FN))}$$

is a single-number metric that takes into account all four cells of the confusion matrix and is robust to class imbalance. A value of +1 indicates perfect classification; 0 indicates performance no better than random; −1 indicates perfect inverse classification. Several comparative studies have argued that MCC is the most informative single-number summary for imbalanced binary classification, and the present author concurs with this assessment for general-purpose evaluation.

Table 5.1 Classification Evaluation Metrics — Definitions, Interpretations, and Recommended Use

Metric	Formula	Range	Sensitivity to Imbalance	Recommended Use
Accuracy	(TP+TN)/(TP+TN+FP+FN)	[0, 1]	Severely misleading for imbalanced data	Only when classes are balanced; always report alongside confusion matrix
Precision	TP/(TP+FP)	[0, 1]	Depends on class distribution	When false alarms are costly (spam filter, fraud alert)

Recall (Sensitivity)	TP/(TP+FN)	[0, 1]	Focuses on positives; unaffected by TN	When missed detections are costly (disease screening, safety systems)
F1 Score	$2 \cdot P \cdot R/(P+R)$	[0, 1]	Moderate — ignores TN	Balanced importance of precision and recall; general purpose
Fβ Score	$(1+\beta^2) \cdot P \cdot R / (\beta^2 P+R)$	[0, 1]	Moderate — β weights recall	When recall β× more important than precision (e.g., β=2 for screening)
AUC-ROC	Area under TPR vs FPR curve	[0.5, 1.0]	Can be optimistic with severe imbalance	Comparing classifiers; threshold-free evaluation; well-calibrated models
AUPRC	Area under Precision-Recall curve	[0, 1]	Naturally accounts for imbalance	Imbalanced datasets; when positive class is rare and important
Matthews Correlation Coefficient	$(TP \cdot TN - FP \cdot FN)/\sqrt{(...)}$	[−1, +1]	Robust to imbalance	Most reliable single metric for imbalanced binary classification
Cohen's Kappa (κ)	$(p_o - p_e)/(1 - p_e)$	[−1, +1]	Accounts for chance agreement	Multi-class; inter-rater agreement; when chance baseline matters

TPR = True Positive Rate (Recall). FPR = False Positive Rate. p_o = observed agreement; p_e = expected agreement by chance. Sources: Powers (2011); Chicco and Jurman (2020); Davis and Goadrich (2006).

5.3 Hyperparameter Tuning: The Search for Good Configurations

Almost every machine learning algorithm has one or more hyperparameters — settings that control the algorithm's behaviour and must be specified before training begins, as distinct from the model parameters that are learned from data. The regularisation strength λ in ridge regression, the number of estimators in a random forest, the kernel bandwidth σ in an SVM, the learning rate η and batch size in stochastic gradient descent — these are hyperparameters, and their values substantially affect the generalisation performance of the trained model.

Grid search evaluates the model on a regular grid of hyperparameter values, exhaustively covering the search space in the specified range. It is conceptually simple and guarantees that the best configuration on the grid will be found, but its computational cost grows exponentially with the number of hyperparameters: a grid of 10 values per hyperparameter requires 10^k evaluations for k hyperparameters. For more than three or four hyperparameters, grid search is computationally impractical.

Random search, introduced by Bergstra and Bengio in 2012, samples hyperparameter configurations uniformly at random from the search space and evaluates a fixed number of configurations. Despite its apparent naivety, random search is often more efficient than grid search in practice: the key insight is that in many problems, the model's performance depends strongly on a small number of hyperparameters and only weakly on the rest. In a grid search, computational budget is wasted on exhaustive coverage of the unimportant dimensions; random search allocates budget more evenly across all dimensions, resulting in better coverage of the important ones.

Bayesian hyperparameter optimisation uses a probabilistic surrogate model — typically a Gaussian process or a Tree-structured Parzen Estimator (TPE) — to model the relationship between hyperparameter configurations and validation performance, and uses this model to intelligently guide the search toward promising regions of the hyperparameter space. At each iteration, the surrogate model is updated with the result of the latest evaluation, and the next configuration to evaluate is chosen by maximising an acquisition function that trades off between exploitation (evaluating configurations predicted to perform well) and exploration (evaluating configurations where the uncertainty about performance is high). Bayesian optimisation has been shown to find good configurations in substantially fewer evaluations than grid or random search, and it is the preferred approach when each evaluation is expensive — as it is for the training of deep neural networks.

Table 5.2 Hyperparameter Tuning Methods — Efficiency and Practical Guidance

Method	Search Strategy	Computational Cost	Number of Hyperparams	When to Use	Key Implementation
Manual tuning	Human expertise + intuition	Low (per config)	Any	Initial exploration; when expert knowledge is strong	—
Grid search	Exhaustive grid over all combinations	O(10^k) × training cost	≤ 3	Small hyperparameter spaces; expensive-	sklearn GridSearchCV

				to-tune baselines	
Random search	Uniform random sampling	O(n_iter) × training cost	Any	3+ hyperparameters; sufficient budget for ~50–200 evals	sklearn RandomizedSearchCV
Bayesian optimisation (GP-based)	Gaussian Process surrogate + EI/UCB acquisition	$O(n\_iter^3)$ for GP update	≤ 20	Expensive evaluations (deep learning); 20–100 eval budget	scikit-optimize, BayesOpt
Bayesian optimisation (TPE-based)	Tree-structured Parzen Estimator	O(n_iter log n_iter)	Any	Large search spaces; many hyperparameters; recommended default	Optuna, Hyperopt
Population-based training (PBT)	Evolutionary; copies + mutations	Large (requires parallel workers)	Any + schedule	Deep learning with compute budget for parallel training	Ray Tune PBT
Neural Architecture Search (NAS)	Search over architecture space	Very Large	Architecture choices	When architecture is the primary design variable	AutoKeras, DARTS

EI = Expected Improvement; UCB = Upper Confidence Bound (acquisition functions). PBT = Population Based Training. NAS = Neural Architecture Search. GP = Gaussian Process. TPE = Tree-structured Parzen Estimator. For most practical machine learning projects, Optuna with TPE-based Bayesian optimisation

represents the best default choice. Sources: Bergstra and Bengio (2012); Bergstra et al. (2011); Snoek et al. (2012); Jaderberg et al. (2017).

5.4 Statistical Model Comparison: Are the Differences Real?

Having evaluated multiple models on the same dataset, the practitioner faces the question of whether observed differences in performance are genuine — reflecting differences in the models' fundamental capabilities — or merely the result of random variation in the evaluation procedure. This question requires a statistical answer, and the statistical testing of machine learning results is a topic that receives insufficient attention in most treatments of the subject.

The Wilcoxon signed-rank test, recommended by Demšar (2006) as the preferred method for comparing two classifiers on multiple datasets, tests the null hypothesis that the performance differences between two models, measured across multiple datasets or cross-validation folds, come from a distribution with median zero. It is a non-parametric test that makes no assumptions about the distribution of performance differences, which is appropriate given that cross-validation scores are not normally distributed. It is more powerful than the sign test (which discards the magnitude of differences) and more robust than the paired t-test (which requires the differences to be normally distributed).

The Friedman test, extended by the Nemenyi post-hoc test, generalises to the comparison of multiple classifiers across multiple datasets. The Friedman test is a non-parametric equivalent of the repeated-measures ANOVA that ranks algorithms by performance on each dataset and tests whether the average ranks differ significantly. When the Friedman test rejects the null hypothesis, the Nemenyi test identifies which pairs of algorithms

differ significantly from one another. Critical difference diagrams — which plot the average rank of each algorithm on a number line, connecting algorithms whose ranks do not differ significantly — provide a compact and informative visualisation of multi-algorithm comparisons.

It is pertinent to note, in the context of Indian research publications in machine learning and pattern recognition, that the statistical comparison of algorithms remains considerably less prevalent in published work than it should be. The observation that algorithm A achieves 94.2 per cent accuracy while algorithm B achieves 93.7 per cent, without any statistical test of whether this difference is significant, is a common — and regrettable — feature of the literature. The practical computation of the Wilcoxon signed-rank test using standard statistical software requires no specialised expertise and adds negligible burden to the evaluation process; its adoption as standard practice would substantially improve the quality of published results.

Part III — Unsupervised and Deep Learning

Unsupervised Learning

Clustering, Dimensionality Reduction, and Density Estimation

Chapter 6

Unsupervised Learning — Clustering, Dimensionality Reduction, and Density Estimation

In 2019, researchers at the CSIR Institute of Genomics and Integrative Biology in New Delhi undertook a population genomics study of remarkable scope. Their dataset contained single nucleotide polymorphism (SNP) profiles for several thousand individuals drawn from diverse linguistic, geographical, and occupational communities across the subcontinent. The research question was genuinely exploratory: does the genetic variation in this data organise itself into meaningful structure, and if so, what does that structure reveal about the history of Indian populations?

No label existed to guide the analysis. There was no target variable to predict. The task was to let the data speak for itself — to discover, without any prior hypothesis about the number or nature of groups, whether clusters existed and how they related to the cultural and geographical map of India. The methods that enabled this discovery — principal component analysis to compress hundreds of thousands of SNP dimensions into informative axes, clustering algorithms to identify groups of individuals with similar genetic profiles, and visualisation tools to project high-dimensional structure onto a two-dimensional map the human eye could read — are the methods of unsupervised learning.

Unsupervised learning is the branch of machine learning that operates without labels, seeking to understand the structure inherent in data. It is, in some respects, the more fundamental half of the discipline: before one can predict, one must understand; and

understanding begins with discovering what patterns are present. The present chapter examines the principal methods of unsupervised learning through both their mathematical foundations and their application to problems of genuine scientific and practical significance.

6.1 Clustering: Discovering Natural Groups

K-Means and Its Geometry

K-means is the most widely used clustering algorithm by virtue of its simplicity, computational efficiency, and surprisingly strong performance on well-separated spherical clusters. Given a dataset of n observations and a target number of clusters k, k-means seeks the partition $C_1, \ldots, C_k$ that minimises the total within-cluster sum of squared distances from each observation to its cluster centroid:

$$\text{minimise\_}C \ \ \Sigma_k \, \Sigma_{\{x_i \in C_k\}} \, ||x_i - \mu_k||^2, \quad \text{where } \mu_k = (1/|C_k|) \, \Sigma_{\{x_i \in C_k\}} \, x_i$$

The Lloyd alternating minimisation algorithm solves this problem iteratively: given fixed centroids, assign each point to its nearest centroid; given fixed assignments, update each centroid to the mean of its assigned points. The algorithm is guaranteed to converge — each iteration reduces the objective unless a fixed point has been reached — but to a local minimum rather than the global one, which is NP-hard to find. The k-means++ initialisation, due to Arthur and Vassilvitskii (2007), places initial centroids with probability proportional to their squared distance from the nearest previously chosen centroid, dramatically reducing the probability of poor local minima and providing an O(log k)-competitive approximation guarantee.

The fundamental geometric limitation of k-means is its implicit assumption that clusters are convex, spherical, and of equal volume — a consequence of partitioning space by nearest-centroid Voronoi diagrams with Euclidean distance. When clusters are elongated, curved, or of different sizes and densities — as is the case for the genetic ancestry components in the IGIB study — k-means produces partitions that reflect its geometric assumptions rather than the genuine structure of the data. This limitation motivates the probabilistic generalisations examined in the next section.

Gaussian Mixture Models and the EM Algorithm

A Gaussian Mixture Model (GMM) represents the data distribution as a weighted sum of K Gaussian components, each with its own mean μ_k, covariance matrix Σ_k, and mixing proportion π_k:

$$p(x) = \Sigma_{k=1}^{H} \pi_k \cdot N(x\ ;\ \mu_k,\ \Sigma_k), \quad \Sigma_k\ \pi_k = 1, \quad \pi_k \geq 0$$

Because each component has an unrestricted covariance matrix, a GMM can represent elliptical clusters of arbitrary orientation, size, and shape — a substantial generalisation over k-means. The parameters are estimated by maximising the marginal log-likelihood of the observed data, which does not admit a closed-form solution because the cluster assignments (which component generated each observation) are unobserved. The Expectation-Maximisation (EM) algorithm, introduced by Dempster, Laird, and Rubin in 1977, alternates between two steps: the E-step computes the posterior probability $r_{ki} = P(z_i = k \mid x_i, \theta)$ that observation i was generated by component k, given the current parameter estimates; the M-step updates the parameters to maximise the expected complete-data log-likelihood, weighted by these posterior

probabilities. Closed-form M-step updates exist for Gaussian components and are proportional to the responsibility-weighted means and covariances of the data. The EM algorithm converges to a local maximum of the marginal likelihood, and like k-means, it requires k to be specified and may converge to poor solutions from bad initialisations.

The connection between k-means and GMMs is illuminating: k-means corresponds to a GMM with isotropic, equal covariance matrices in the limit of zero variance, where the soft E-step posterior probabilities collapse to hard 0/1 assignments. This derivation reveals that k-means is not an ad hoc heuristic but a maximum likelihood algorithm for a specific, restrictive probabilistic model — a perspective that clarifies both its strengths and its limitations.

Hierarchical Clustering and DBSCAN

Agglomerative hierarchical clustering builds a dendrogram — a tree of nested clusters — by starting with each observation as its own singleton cluster and merging the two most similar clusters at each step. The linkage criterion determines similarity between clusters: complete linkage uses the maximum pairwise distance (producing compact clusters); average linkage uses the mean pairwise distance (a compromise); Ward's linkage minimises the increase in total within-cluster variance at each merge (producing equal-sized, compact clusters). The dendrogram produced by hierarchical clustering represents the merging structure at all scales simultaneously, enabling the analyst to choose the number of clusters by 'cutting' the tree at the desired level — a significant advantage over k-means, which requires k to be fixed in advance.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise), introduced by Ester and colleagues in 1996, defines

clusters as dense regions of the feature space separated by low-density regions. A point is a ‘core point’ if at least minPts observations lie within radius ε of it; a cluster is the connected component of core points and the non-core points in their neighbourhoods; points not within ε of any core point are labelled as noise. DBSCAN discovers clusters of arbitrary shape and explicitly identifies outliers — both features that k-means and GMMs lack. It requires no specification of k, though the two parameters ε and minPts must be chosen. DBSCAN is the natural choice for applications where clusters are defined by geographic concentration — the spatial distribution of agricultural disease outbreaks, the clustering of road accident locations for urban safety planning, the identification of informal settlement boundaries in satellite imagery of Indian cities.

Table 6.1 Clustering Algorithm Comparison

Algorithm	Cluster Shape	Handles Noise	Complexity	Requires k	Key Strength	Best Application
K-Means++	Spherical, equal size	No	O(nkd) per iter	Yes	Fast; globally effective with good init	Large data; spherical clusters; baseline
Gaussian Mixture Model	Elliptical (full covariance)	Soft (low probability)	O(nk d^2) per iter	Yes	Soft assignments; flexible shapes; principled	Overlapping elliptical clusters; density modelling
Hierarchical (Ward)	Compact, roughly spherical	No	O(n^2 log n)	Dendrogram cut	No k required; multi-scale	Small datasets; exploratory; dendrogra

					structure visible	m informative
DBSCAN	Arbitrary shape	Yes — explicit noise label	O(n log n) with index	No	Arbitrary shapes; explicit outlier handling	Geospatial; irregular clusters; outlier detection
HDBSCAN	Arbitrary, hierarchical	Yes — robust	O(n log n)	No	Handles varying densities; robust to ε	Complex varying-density datasets; genomics
Spectral Clustering	Arbitrary (via graph)	No	$O(n^2–n^3)$	Yes	Non-convex clusters; graph-structured data	Image segmentation; small datasets; graph data

Sources: Arthur & Vassilvitskii (2007); Dempster et al. (1977); Ester et al. (1996); McInnes et al. (2017, HDBSCAN).

6.2 Dimensionality Reduction: Finding the Signal

Principal Component Analysis

PCA finds the orthogonal linear subspace of dimension r that maximises the variance of the projected data — equivalently, the subspace that minimises the average squared reconstruction error. The r principal components are the eigenvectors of the sample covariance matrix $S = (1/n)\ X^TX$ (for centred data X) corresponding to the r largest eigenvalues, or equivalently the top-r right singular vectors of X. The projection of observation x_i onto the r principal components gives a score vector $z_i \in \mathbb{R}^r$; the original observation is approximately reconstructed as $\hat{x}_i = Uz_i + \mu$, where

U is the matrix of principal component directions and μ is the sample mean.

The proportion of variance explained by the first r components — the ratio of the sum of the top-r eigenvalues to the total trace of the covariance matrix — is the standard diagnostic for choosing r. A scree plot, which plots eigenvalues in decreasing order, typically shows a 'kink' at the point where eigenvalues begin to decline more slowly, suggesting that additional components beyond this point represent noise rather than signal. For the genomics application, plotting the first two principal components produces a map of genetic similarity that strikingly mirrors India's geographic and linguistic divisions — a result that has been reproduced in multiple population genomics studies and that illustrates how much interpretable information can be extracted from two dimensions of a 600,000-dimensional dataset.

PCA is a linear method, and its inability to capture non-linear manifold structure motivates the non-linear dimensionality reduction methods that follow. However, it is worth emphasising that PCA's linearity is also a strength in many contexts: PCA components are interpretable as linear combinations of the original features; the algorithm is deterministic and computationally efficient; and the reconstruction criterion is exact and analytically tractable. For many practical applications in Indian research — preprocessing of high-dimensional tabular data, noise reduction in spectroscopic data from ISRO's remote sensing instruments, and decorrelation of features before logistic regression — PCA remains the dimensionality reduction method of first choice.

t-SNE and UMAP

t-distributed Stochastic Neighbour Embedding (t-SNE), due to Van der Maaten and Hinton (2008), is a non-linear method

designed for visualisation. It constructs a probability distribution over pairs of high-dimensional points — placing high probability on nearby points and low probability on distant ones — and a corresponding distribution over pairs in a two-dimensional embedding, using a Student-t distribution to avoid the crowding problem that affects Gaussian-based embeddings. The embedding is optimised to minimise the KL divergence between these two distributions. t-SNE produces vivid, well-separated cluster visualisations that have made it the dominant tool for exploratory analysis of high-dimensional biological data. Its limitations are equally well-documented: global distances in a t-SNE plot are not meaningful; the perplexity hyperparameter substantially affects the visual appearance; and the algorithm is non-deterministic.

Uniform Manifold Approximation and Projection (UMAP), introduced by McInnes, Healy, and Melville (2018) and grounded in the mathematical theory of fuzzy simplicial sets, addresses several of t-SNE's limitations while achieving comparable or superior visualisation quality. UMAP is substantially faster than t-SNE, preserves more global topological structure, and produces more reproducible results. It has displaced t-SNE as the preferred visualisation method for single-cell RNA sequencing data in the global genomics community, and its adoption for the analysis of Indian language embedding spaces — visualising the geometric structure learned by models such as IndicBERT — illustrates its generality beyond biological applications.

Table 6.2 Dimensionality Reduction Methods — Comparative Reference

Method	Type	Preserved Structure	Output Interpretability	Scalability	Primary Application
PCA	Linear	Global variance (max var projection)	High — linear combinations of features	O(nd·min(n,d)) — very fast	Preprocessing; noise reduction; genomics PCA plots
Kernel PCA	Non-linear via kernel	Non-linear variance	Low — implicit feature space	$O(n^2)$ kernel matrix — limited to small n	Non-linear structure; small/medium datasets
ICA	Linear	Statistical independence of components	Moderate — independent source signals	$O(nd^2)$ — moderate	Signal separation; EEG; financial returns
t-SNE	Non-linear (probabilistic)	Local neighbourhood structure only	None — for 2-3D visualisation only	O(n log n) with Barnes-Hut	Visualisation; single-cell biology; exploratory
UMAP	Non-linear (topological)	Local + partial global	None — visualisation + general use	O(n log n) — faster than t-SNE	Visualisation; single-cell; Indian language embeddings
Autoencoder	Non-linear (neural)	Reconstruction loss-defined	Very low — implicit latent code	Scalable via mini-batch SGD	Anomaly detection; feature extraction; pre-training

VAE	Non-linear (generative)	ELBO (reconstruction + KL)	Low — probabilistic latent space	Scalable via mini-batch SGD	Generation; imputation; representation learning

ICA = Independent Component Analysis. EEG = Electroencephalogram. ELBO = Evidence Lower Bound. Sources: Jolliffe (2002); Van der Maaten & Hinton (2008); McInnes et al. (2018); Kingma & Welling (2013).

6.3 Density Estimation and Variational Autoencoders

Kernel Density Estimation (KDE) estimates the probability density function of the data-generating distribution non-parametrically by placing a kernel K at each training point and summing contributions, with a bandwidth h controlling the smoothness of the estimate. For low-dimensional problems — up to about ten dimensions — KDE is consistent and statistically well-understood. Beyond ten dimensions, the curse of dimensionality renders KDE impractical: the volume of space grows exponentially with dimension, data becomes sparse, and the bias-variance trade-off for bandwidth selection becomes unmanageable.

Variational Autoencoders (VAEs), introduced by Kingma and Welling (2013), address density estimation and generation in high-dimensional spaces through the framework of amortised variational inference. A VAE encoder network $q_\varphi(z|x)$ maps an observed input x to a distribution over a low-dimensional latent variable z; a decoder network $p_\theta(x|z)$ maps a latent sample back to the observation space. Training maximises the Evidence Lower Bound (ELBO) with respect to both encoder and decoder parameters:

$$ELBO(\varphi,\theta;x) = \mathbb{E}_{q\varphi(z|x)}[\log p\theta(x|z)] - KL(q\varphi(z|x) \,||\, p(z))$$

The first term is the reconstruction loss; the second is the KL regulariser that encourages the encoder's posterior to remain close to the standard Gaussian prior $p(z) = N(0,I)$. The reparameterisation trick — sampling $z = \mu + \sigma \odot \varepsilon$ where $\varepsilon \sim N(0,I)$ and (μ, σ) are the encoder's output — allows gradients to flow through the sampling operation, enabling end-to-end training by backpropagation. VAEs are used in Indian medical research for generating synthetic training data to augment small clinical imaging datasets, for imputing missing values in genomic records, and for anomaly detection in industrial sensor data — applications where the ability to model the full data distribution, rather than merely a discriminative boundary, is the critical capability.

Neural Networks and Deep Learning

From Perceptrons to Transformers

Chapter 7

Neural Networks and Deep Learning — From Perceptrons to Transformers

In October 2012, a convolutional neural network trained by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton submitted predictions to the ImageNet Large Scale Visual Recognition Challenge. AlexNet achieved a top-5 error rate of 15.3 per cent. The second-place entry, using conventional computer vision methods, achieved 26.2 per cent. The gap was not merely quantitative; it was categorical — it signalled that the right question was not how to engineer image features but how to learn them, at multiple levels of abstraction, directly from data.

A decade later, a student at IIT Madras could in an afternoon fine-tune a pretrained neural network on a dataset of Indian road signs in Tamil, Hindi, and English, achieving state-of-the-art recognition without designing a single feature, using twelve lines of code built on six decades of accumulated research. The story of deep learning is a story of compounding ideas, each building on the last, across a timeline longer and more discontinuous than the popular AlexNet narrative suggests.

The present chapter traces this story from mathematical foundations — the perceptron and the universal approximation theorem — through convolutional and recurrent architectures, to the transformer that defines the field's present state. At each stage, both the theoretical understanding and the engineering intuition that a serious practitioner requires are presented.

7.1 The Perceptron and Universal Approximation

The perceptron, introduced by Rosenblatt in 1958, computes a weighted sum of its inputs and passes the result through a threshold activation. Given input $x \in \mathbb{R}^d$ and weight vector $w \in \mathbb{R}^d$:

$$\hat{y} = sign(w^T x + b)$$

The perceptron convergence theorem guarantees that the learning rule — which adjusts w in the direction of the misclassified example when an error occurs — finds a perfect separating hyperplane in finite steps, provided the data is linearly separable. The limitation identified by Minsky and Papert (1969) — that no single perceptron can compute XOR, and more generally that linear classifiers cannot solve problems whose classes are not linearly separable — motivated the development of multi-layer architectures. The missing ingredient was not the architecture but the training algorithm: an efficient method for computing gradients through stacked layers.

The Universal Approximation Theorem, established by Hornik, Stinchcombe, and White (1989), provides the theoretical justification for multi-layer networks: a single hidden layer of sufficient width can approximate any continuous function on a compact domain to arbitrary precision, given a non-constant bounded activation. This result establishes representational sufficiency — any function we might wish to approximate is in principle representable — but says nothing about learnability (whether gradient descent will find the right parameters) or efficiency (how many neurons are required). The practical case for depth — multiple layers — rests on the empirical and theoretical observation that deep networks can represent certain functions exponentially more efficiently than shallow ones, requiring far fewer parameters for equivalent approximation accuracy.

7.2 Backpropagation: The Chain Rule at Scale

Backpropagation is the algorithm for computing the gradient of the loss with respect to all network parameters efficiently, using the chain rule applied layer by layer from output to input. For a network with L layers, where layer l computes $a^l = \sigma(W^l a^{l-1} + b^l)$, the weight gradient $\partial L/\partial W^l$ is computed as:

$$\partial L/\partial W^l = \delta^l (a^{l-1})^T, \quad \text{where } \delta^l = (W^{l+1})^T \delta^{l+1} \odot \sigma'(W^l a^{l-1} + b^l)$$

The error signal δ^l propagates backward through the network, with each layer's contribution determined by the chain rule product of the upstream error and the local derivative $\sigma'(\cdot)$. The total cost of the backward pass is proportional to the forward pass — approximately twice the cost of a single forward evaluation — making gradient-based training of networks with millions of parameters computationally practical.

The Rectified Linear Unit (ReLU), defined as $\text{ReLU}(z) = \max(0, z)$, largely resolves the vanishing gradient problem that plagued sigmoid and tanh activations in deep networks: its derivative is 1 for $z > 0$, allowing gradients to propagate unchanged through any number of positive-activation layers. ReLU's simplicity, combined with its empirical superiority to sigmoid and tanh in deep networks, has made it the default activation function in virtually all modern feedforward and convolutional architectures. Variants including Leaky ReLU, PReLU, ELU, and GELU (Gaussian Error Linear Unit — the standard choice in transformers) address the 'dying ReLU' problem, in which units with permanently negative pre-activations produce zero gradients and never recover.

7.3 Convolutional Neural Networks

CNNs incorporate two structural priors about image data that dramatically reduce parameter count and improve data efficiency: spatial locality (useful features are defined by local spatial patterns) and translational equivariance (a feature detector useful at one location is useful everywhere). The convolution operation applies a small learnable filter to every position of the input, computing a dot product between the filter and each local receptive field. Multiple filters in parallel produce multiple feature maps, together constituting one layer's output. Successive layers combine these local features into increasingly complex patterns: edges → textures → parts → objects.

The residual connection, introduced by He and colleagues in their ResNet paper (2016), is the most consequential architectural innovation since the convolutional layer. A residual block learns:

$$output = F(x, \{W_i\}) + x$$

where F(x, $\{W_i\}$) is the residual function — the difference between desired output and input — and the skip connection adds the input identity directly. Residual connections solve the degradation problem empirically observed in networks deeper than about 20 layers by providing direct gradient pathways to earlier layers and making identity mappings trivially learnable. ResNet-50 and ResNet-152 are still widely deployed as backbone networks in Indian computer vision applications, including smart city object detection systems in Bengaluru and Hyderabad and ISRO's satellite image analysis pipelines.

7.4 Recurrent Networks and Long-Range Memory

Recurrent neural networks process sequential data by maintaining a hidden state h_t that summarises the sequence seen thus far:

$$h_t = \sigma(W^h h_{t-1} + W_x x_t + b)$$

Training by backpropagation through time (BPTT) unrolls the recurrence across timesteps and applies the chain rule. The vanishing gradient problem — gradients shrink exponentially with the number of timesteps because the same weight matrix W^h is multiplied repeatedly — prevents standard RNNs from learning dependencies across more than about 10–20 timesteps.

The Long Short-Term Memory (LSTM) cell, due to Hochreiter and Schmidhuber (1997), addresses this through a gating mechanism that controls information flow into and out of a cell state c_t:

$$f_t = \sigma(W_f [h_{t-1}, x_t] + b_f) \quad \text{(forget gate — what to discard from } c_{t-1})$$

$$i_t = \sigma(W_i [h_{t-1}, x_t] + b_i) \quad \text{(input gate — what new information to add)}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tanh(W_c [h_{t-1}, x_t] + b_c)$$

$$h_t = \sigma(W_o [h_{t-1}, x_t] + b_o) \odot \tanh(c_t)$$

The cell state provides a gradient highway: when $f_t \approx 1$, the cell state — and therefore the gradient — passes through the timestep essentially unchanged, allowing LSTMs to learn dependencies across hundreds of timesteps. LSTMs and GRUs dominated sequence modelling from 1997 until approximately 2018, when the transformer superseded them for most NLP tasks, though LSTMs remain competitive for time series forecasting and

for applications where computational resources preclude large transformer models.

7.5 The Attention Mechanism and the Transformer

The transformer, introduced by Vaswani and colleagues in Attention Is All You Need (2017), replaces recurrent processing entirely with self-attention — a mechanism that allows every position in a sequence to attend directly to every other position, computing a weighted sum of value representations where weights are determined by query-key similarity:

$$Attention(Q,K,V) = softmax(QK^T / \sqrt{d_k})\ V$$

where $Q = XW^d$, $K = XW^W$, $V = XW^\beta$ are learned linear projections of the input X, and the scaling factor $1/\sqrt{d_k}$ prevents the dot products from becoming large in magnitude and saturating the softmax. Multi-head attention runs h self-attention operations in parallel with separate projections, concatenating their outputs — allowing the model to simultaneously attend to different aspects of the input from different representation subspaces. The transformer encoder stacks L identical blocks, each comprising multi-head self-attention and a position-wise feedforward sublayer, with layer normalisation and residual connections throughout. Positional encodings — fixed sinusoidal or learned — supply the model with token position information, since self-attention is permutation-equivariant.

The pretraining-finetuning paradigm — training a large transformer on unlabelled text to predict masked tokens (BERT) or the next token (GPT), then fine-tuning on a specific task — democratised high-quality NLP by making large pretrained representations available for downstream tasks without the resources to train from scratch. For Indian language NLP, this

paradigm is embodied in IndicBERT (AI4Bharat, IIT Madras), trained on twelve Indian languages; MuRIL (Google AI), covering seventeen languages; IndicBART for generation; and the Dhruva models from Sarvam AI, optimised for Indian automatic speech recognition. These models have substantially improved the state of the art on Hindi, Tamil, Telugu, Bengali, Marathi, and other Indian language benchmarks, and their continued development is one of the most active areas of applied ML research in India.

Table 7.1 Neural Architecture Comparison — Domain, Innovation, and Indian Applications

Architecture	Primary Domain	Key Innovation	Pretrained Models	Indian Application Examples
MLP (Feedforward)	Tabular, general	Multi-layer non-linear composition	Limited	Credit scoring (NBFCs); insurance risk; structured clinical data
CNN — ResNet family	Images	Convolution + residual connections	Very rich (ImageNet)	Smart city surveillance; agricultural disease detection; ISRO image analysis
Vision Transformer (ViT)	Images	Self-attention on image patches	Available (ViT-B/16, DeiT)	Medical image analysis; satellite crop mapping; document digitisation

LSTM / GRU	Sequences, time series	Gated cell state for long-range memory	Limited	Demand forecasting; power grid prediction; river flow modelling
BERT-style Encoder	NLP classification, tagging	Bidirectional pretraining	Very rich (IndicBERT, MuRIL)	Sentiment analysis in 22 languages; government document classification; NER
GPT-style Decoder	NLP generation	Autoregressive causal pretraining	Very rich (GPT, Llama, Sarvam)	Chatbots; content generation; code completion for Indian developers
Encoder-Decoder Transformer	Seq-to-seq translation	Cross-attention between encoder/decoder	Available (mT5, IndicBART)	Hindi-English translation; Indic language MT; speech-to-text

IndicBERT, IndicBART, and the IndicNLP Suite are developed by AI4Bharat at IIT Madras (https://ai4bharat.iitm.ac.in/). MuRIL is from Google AI India Research. Sarvam AI's Dhruva models are available at https://www.sarvam.ai/. NER = Named Entity Recognition. MT = Machine Translation.

Table 7.2 Deep Learning Regularisation Techniques — Mechanisms and Application

Technique	Mechanism	Primary Effect	When to Apply	Key Risk if Misused
L2 Weight Decay	Adds $\lambda\|\|w\|\|^2$ to loss	Discourages large weights; implicit Gaussian prior on w	Always as baseline; tune λ on validation set	Too large λ: underfitting; loss of representational capacity

Dropout	Randomly zeroes activations at rate p during training	Prevents co-adaptation; implicit ensemble over 2^d sub-networks	Dense layers; moderate-large models; not standard conv layers	High p: slow convergence; instability; p=0 at inference
Batch Normalisation	Normalises activations per mini-batch; scales and shifts	Smooths loss landscape; faster training; mild regularisation	Convolutional networks; before or after activation (try both)	Small batches: noisy statistics; variable length sequences: use Layer Norm
Layer Normalisation	Normalises across feature dimension per sample	Stable training for variable batch sizes and sequence models	Transformers; RNNs; any architecture with variable batch size	Slower convergence than BatchNorm on some CNN tasks; higher memory
Early Stopping	Halt training when validation loss stops improving (patience p)	Prevents overfitting; implicit regularisation	Always when validation set available; set patience ≥ 10 epochs	Too low patience: underfitting; too high: overfitting resumes after plateau
Data Augmentation	Random transformations of training samples	Increases effective training set size; improves invariance	Images (crop, flip, colour); text (back-translation); audio (time stretch)	Domain-inappropriate augmentations introduce distribution shift
Mixup	Trains on convex combinations of training pairs ($\lambda \sim$ Beta(α,α))	Smoother decision boundaries; improved calibration; reduced overconfidence	Classification with sufficient data; image, text	Hurts performance on fine-grained or imbalanced datasets

Sources: Ioffe & Szegedy (2015, BatchNorm); Srivastava et al. (2014, Dropout); Ba et al. (2016, LayerNorm); Prechelt (1998, Early Stopping); Zhang et al. (2018, Mixup). BatchNorm should not be used with variable batch sizes or in transformer architectures; Layer Norm is the transformer standard.

Part IV — Applications, Ethics, and Frontiers

Part IV · Chapter 8

Machine Learning in Practice

Pipelines, Deployment, and Indian Applications

Chapter 8

Machine Learning in Practice — Pipelines, Deployment, and Indian Applications

In the financial year 2022–2023, the National Payments Corporation of India's Unified Payments Interface processed over 83 billion transactions with a total value exceeding Rs 139 lakh crore. At any given moment, several hundred transactions per second were being authorised or declined by the NPCI's risk engines — systems that must, within a few hundred milliseconds, evaluate each transaction for potential fraud against the background of billions of historical transactions, a continuously evolving landscape of fraud patterns, and the legitimate behaviour of hundreds of millions of users who transact in ways that are each individually idiosyncratic.

The machine learning models that power such systems were not produced by a data scientist who cleaned a dataset, trained a classifier, and deployed the resulting pickle file to a server. They are the outputs of complex socio-technical pipelines: data collection and labelling workflows, feature engineering systems that compute hundreds of features from raw transaction streams in real time, model training infrastructures that retrain models daily as new fraud patterns emerge, deployment systems that serve predictions at low latency under strict availability requirements, and monitoring systems that detect when model performance has degraded — perhaps because a new fraud technique has emerged that the model has never seen. The gap between the machine learning of academic papers and textbooks, and the machine learning of production systems that process the financial transactions of a billion people, is the subject of the present chapter.

8.1 The Full Stack of a Production ML System

The academic presentation of machine learning — which necessarily focuses on algorithms, mathematical derivations, and benchmark results — can create the impression that building a machine learning system is primarily an algorithmic task: choose a model family, tune the hyperparameters, measure the accuracy on a held-out test set, and the work is done. This impression is, in the experience of practitioners who have built systems that operate at real-world scale, significantly mistaken. Sculley and colleagues at Google estimated, in their influential 2015 paper Hidden Technical Debt in Machine Learning Systems, that in a typical large-scale ML system, only a small fraction of the code consists of the ML model itself; the overwhelming majority is devoted to data pipeline infrastructure, feature engineering, serving infrastructure, and monitoring. The model is the iceberg tip; the infrastructure is the submerged mass.

A production machine learning pipeline consists, at minimum, of the following components. Data ingestion handles the collection, validation, and storage of raw data from its sources — databases, APIs, event streams, sensor feeds. Feature engineering transforms raw data into the feature representations that the model consumes — computing derived variables, encoding categorical features, normalising numerical features, handling missing values, and constructing features that require temporal or relational reasoning across multiple records. Training infrastructure manages the periodic or continuous retraining of models as new data becomes available, including experiment tracking, model versioning, and comparison of candidate models against one another. Serving infrastructure deploys trained models for inference, handling prediction requests at the latency and throughput required by the application, and managing the lifecycle of multiple model versions simultaneously. Monitoring systems track model performance in production — measuring prediction accuracy against actual outcomes when those become available,

detecting data drift when the distribution of incoming features changes, and triggering alerts when performance degradation exceeds defined thresholds.

The design of each component raises engineering challenges that are qualitatively different from the algorithmic challenges of model selection and training. The feature engineering system must compute features consistently across training and serving — a training-serving skew, in which features are computed differently in the two contexts, is among the most common and most damaging production bugs in ML systems. The training infrastructure must support reproducibility — the ability to recreate any historical model exactly, from data and code, for debugging and audit purposes. The serving infrastructure must handle peak load gracefully and degrade gracefully under failure. The monitoring system must distinguish genuine model degradation from statistical noise in evaluation metrics, and must provide actionable diagnostics rather than merely raising alarms.

8.2 Data Quality: The Foundation on Which Models Rest

It is a truism in the machine learning practitioner community that data quality matters more than algorithm quality — that a carefully cleaned dataset with a simple model will outperform a carelessly cleaned dataset with a sophisticated one. This truism is, in the author's experience, broadly accurate and consistently underweighted in the allocation of effort in ML projects. The present section examines the principal data quality issues that arise in practice and the strategies for addressing them.

Missing data is endemic in real-world datasets. In the Indian context, missingness takes several characteristic forms. In clinical datasets from government hospitals, measurements are often missing because they were not clinically indicated for a given patient rather than because they were lost or not collected — a pattern called missing not at random (MNAR), in which the probability of missingness depends on the unobserved value itself.

Imputing missing laboratory values as the population mean, without accounting for the informative nature of the missingness, can introduce systematic bias into the model's predictions. In agricultural datasets from ICAR's Krishi Vigyan Kendras, records are frequently incomplete because the data collection is volunteer-based and farmers or extension workers may not record every variable for every observation. In administrative data from government agencies, fields are often missing because the underlying form was not completed — a pattern that reflects administrative behaviour rather than the phenomenon of interest.

The appropriate response to missing data depends on the missing-data mechanism. When data is missing completely at random (MCAR) — when the probability of missingness is independent of both observed and unobserved values — simple imputation with the column mean or median introduces minimal bias. When data is missing at random (MAR) — when the probability of missingness depends only on observed variables, not on the unobserved value itself — multiple imputation or model-based imputation can produce unbiased estimates. When data is MNAR, these standard approaches are biased, and either domain-knowledge-based adjustments or joint modelling of the outcome and the missingness mechanism are required.

Class imbalance is a pervasive challenge in the classification problems of greatest practical importance — fraud detection, disease diagnosis, equipment failure prediction, credit default — precisely because these problems are important in proportion to how rare the positive class is. The approaches to class imbalance fall into three categories. Sampling methods adjust the training set by oversampling the minority class (either by replication — SMOTE and its variants — or by generating synthetic examples), or by undersampling the majority class. Cost-sensitive learning assigns higher misclassification costs to minority class errors during training, directly optimising for the desired class-specific performance. Threshold adjustment shifts the decision threshold of

a well-calibrated probabilistic classifier from the default of 0.5 toward a value that achieves the desired precision-recall trade-off. The choice among these approaches should be guided by the evaluation metric that matters for the application — as Chapter 5 established, accuracy is not that metric for imbalanced problems.

8.3 Feature Engineering: The Practitioner's Craft

Feature engineering — the transformation of raw data into the feature representations that a machine learning model consumes — is the component of the ML pipeline that most distinguishes skilled practitioners from novices, and the component that most determines model performance on structured data problems. A model trained on well-engineered features consistently outperforms a more sophisticated model trained on raw features, and the effort invested in feature engineering returns more reliably than an equivalent effort invested in hyperparameter tuning or architecture search.

The handling of categorical variables deserves careful treatment, as it is an area where naive approaches introduce subtle errors. One-hot encoding — representing a categorical variable with k possible values as k binary indicator variables — is the standard approach for nominal categories with a small number of values. For high-cardinality categoricals — postal codes, product SKUs, hospital procedure codes — one-hot encoding produces extremely high-dimensional sparse feature vectors that are both computationally expensive and statistically inefficient. Target encoding replaces each category value with the mean of the target variable for training examples in that category, producing a compact single-feature representation that captures the predictive signal in the category. Target encoding must be applied with care to avoid overfitting: cross-validation-based target encoding, which computes the encoding for each training fold using the other folds' data, is the appropriate approach.

Temporal feature engineering is of particular importance for the class of prediction problems that dominate Indian commercial applications: transaction fraud detection, customer churn prediction, demand forecasting, credit scoring. In these applications, the most informative features are often not the raw values of individual observations but aggregates computed over historical windows: the number of transactions by this customer in the last hour; the average transaction value over the last thirty days; the ratio of the current transaction value to the customer's historical mean; the number of days since the customer's last transaction. These 'rolling aggregates' capture the temporal pattern of the customer's behaviour, which is the most powerful signal available for distinguishing legitimate from fraudulent or churning users. Computing rolling aggregates efficiently — without introducing data leakage by including future observations in a window that should contain only past observations — is one of the most technically demanding aspects of feature engineering for temporal applications.

8.4 MLOps: The Engineering Discipline of ML in Production

MLOps — a portmanteau of Machine Learning and Operations — is the set of practices, tools, and cultural norms that govern the deployment, monitoring, and maintenance of machine learning systems in production. It is the ML analogue of DevOps in conventional software engineering, and its emergence as a recognised discipline reflects the maturation of machine learning from an academic practice to an engineering one that must meet the same standards of reliability, reproducibility, and maintainability expected of any production software system.

Experiment tracking is the practice of recording, for every model training run, the complete specification of the experiment: the dataset version used, the hyperparameters, the training code version, the computed evaluation metrics, and the resulting model artefact. Without systematic experiment tracking, the ML practitioner quickly finds herself in the position of being unable to

reproduce a result she obtained last week, because she cannot determine exactly which combination of data, code, and hyperparameters produced it. Tools including MLflow, Weights and Biases, and Neptune.ai provide experiment tracking infrastructure that stores this information automatically for each training run and makes it searchable and comparable.

Model versioning extends the version control discipline of software engineering — where every change to source code is tracked, attributed, and reversible — to model artefacts and the datasets that produced them. A model version consists not merely of the serialised model weights but of the training code, the dataset snapshot, the preprocessing pipeline, and the evaluation results — everything needed to reproduce the model from scratch. This comprehensive versioning is essential for audit and regulatory compliance in the financial and healthcare applications that are among ML's most important Indian deployments: when a loan application is declined or a medical diagnosis is made by an ML model, the organisation deploying that model must be able to demonstrate exactly which model version made that decision and to reproduce its reasoning.

Concept drift — the phenomenon by which the statistical relationship between features and the target variable changes over time — is the most insidious threat to the long-term performance of deployed ML systems. In the fraud detection context, drift occurs as fraudsters adapt their techniques in response to the model's decisions; the model that was trained on historical fraud patterns progressively loses its ability to detect new patterns it has never seen. In the credit scoring context, drift occurs as the economic environment changes — a model trained before a recession may be poorly calibrated to the risk patterns of recession-era borrowers. Monitoring for concept drift requires continuous evaluation of the model's predictions against ground truth as it becomes available, and automated retraining pipelines that detect

performance degradation and trigger model refresh before the degradation becomes operationally significant.

Table 8.1 MLOps Tool Ecosystem — Open-Source and Cloud-Managed Options

MLOps Function	Open-Source Tools	AWS	GCP	Azure	Indian Cloud (NIC/CtrlS)
Experiment tracking	MLflow, Weights & Biases, Neptune	SageMaker Experiments	Vertex AI Experiments	Azure ML Experiments	MLflow on NIC cloud (self-hosted)
Model versioning and registry	MLflow Model Registry, DVC	SageMaker Model Registry	Vertex AI Model Registry	Azure ML Model Registry	MLflow registry; manual versioning on MeghRaj
Data versioning	DVC, LakeFS, Delta Lake	S3 + Glue Data Catalog	BigQuery + Dataplex	Azure Data Lake + Purview	MinIO on-prem; ICAR data portal

Feature store	Feast (open-source), Hopsworks	SageMaker Feature Store	Vertex AI Feature Store	Azure ML Feature Store	No managed offering; Feast self-hosted
Training infrastructure	Kubeflow, Ray Train, PyTorch Lightning	SageMaker Training Jobs	Vertex AI Training	Azure ML Compute Clusters	NSM HPC (PARAM series); NIC compute
Model serving	Seldon Core, BentoML, TorchServe	SageMaker Endpoints	Vertex AI Endpoints	Azure ML Online Endpoints	Custom Flask/FastAPI on NIC; CDAC serving
Monitoring and drift detection	Evidently AI, Alibi Detect, Prometheus	SageMaker Model Monitor	Vertex AI Model Monitoring	Azure ML Data Drift Monitor	Custom monitoring; limited managed options
Automated retraining	AutoML with trigger on drift	SageMaker Autopilot	Vertex AI AutoML	Azure AutoML	Not available; manual retraining

(AutoML)					workflows

NSM = National Supercomputing Mission (PARAM series at IIT/IISc nodes). NIC = National Informatics Centre. CDAC = Centre for Development of Advanced Computing. MeghRaj = Government of India cloud (NIC). Feast, MLflow, DVC, and BentoML are open-source and deployable on any infrastructure, including government cloud. For organisations subject to data residency requirements, self-hosted open-source tools on NIC infrastructure are the recommended approach. Sources: Sculley et al. (2015); Breck et al. (2017); various vendor documentation.

8.5 Case Studies: ML in the Indian Context

Fraud Detection at UPI Scale

The fraud detection challenge at the scale of UPI — processing hundreds of transactions per second, with a false positive rate that must be kept very low to avoid disrupting legitimate payments — illustrates several of the production ML challenges described in the present chapter simultaneously. The feature engineering problem is demanding: the most informative features require real-time aggregation of the customer's transaction history over multiple time windows, and the system must compute these aggregates with a latency measured in milliseconds. The model must be updated frequently as fraud patterns evolve, requiring a retraining pipeline that can produce a new model and validate it against a held-out evaluation set within hours of new labelled data becoming available. The deployment must serve predictions at sub-100-millisecond latency under peak load, requiring a serving infrastructure that is both horizontally scalable and highly available. The evaluation metric must be chosen carefully — precision at high recall thresholds, or the area under the precision-recall curve — to reflect the asymmetric costs of false positives (blocking legitimate transactions, harming user

experience and merchant acceptance) and false negatives (allowing fraudulent transactions to proceed, with direct financial loss).

The NPCI has published limited technical details about its fraud detection infrastructure, but available information suggests a hybrid architecture combining rule-based systems — which handle known fraud patterns with high precision — with gradient boosted tree models and, more recently, graph neural network models that exploit the structure of the transaction network to identify coordinated fraud rings. This multi-layer architecture, in which simpler and more interpretable rule-based components handle high-confidence cases and machine learning models handle the uncertain middle ground, is characteristic of responsible production deployment in high-stakes financial applications.

AI for Agriculture: Crop Monitoring and Advisory Systems

India's Kisan AI initiative and several state-level digital agriculture projects represent a category of ML deployment with different characteristics from fraud detection: the data is sparse and heterogeneous, the users are often low-literacy farmers communicating through voice interfaces in regional languages, and the consequences of incorrect predictions — a farmer who follows incorrect advisory advice may lose an entire season's income — demand conservative uncertainty handling. The machine learning models in these applications are typically ensemble models (random forests or gradient boosted trees) for tabular agronomic data, combined with convolutional networks for satellite imagery analysis and speech recognition models for voice interfaces. The challenge of deployment is not primarily computational — the prediction volumes are modest — but epidemiological: ensuring that advisory recommendations are calibrated to local agroclimatic conditions and that uncertainty is communicated to farmers in a form they can understand and act upon. A crop disease prediction model that reports '72% probability of blast outbreak in the next two weeks' is less useful to a farmer than one that says 'conditions are favourable for blast — apply fungicide before the next rain.'

The Directorate of Wheat Research in Karnal, the National Rice Research Institute in Cuttack, and several state agricultural universities have developed location-specific prediction models for their respective crops and regions. The aggregation of these geographically specific models into a national advisory platform — as attempted by the ICAR through its Crop Weather Watch Group — illustrates the federated nature of Indian agricultural ML deployment, in which domain expertise is distributed across a large number of specialised institutions rather than concentrated in a single national centre.

Table 8.2 Indian ML Deployment Case Studies — Technical and Institutional Characteristics

Application	Institution(s)	Data Characteristics	Model Family	Key Deployment Challenge	Evaluation Metric
UPI Transaction Fraud Detection	NPCI, partner banks	High-volume streams; severe class imbalance (~0.1% positive); temporal	GBT + rule engine + (experimental) GNN	Sub-100ms latency; daily model refresh; false positive minimisation	Precision at 95%+ recall; AUPRC
Crop Disease	ICAR, NCIPM,	Sparse; heterogen	Random Forest;	Low-bandwidt	Calibrated

Prediction Advisory	state agri universities	eous; seasonal; district-level aggregates	logistic regression; LSTM for time series	h rural delivery; voice interface; calibrated uncertainty	probability; advisory precision-recall
Diabetic Retinopathy Screening	Aravind Eye Care System, Sankara Nethralaya	Medical images; class-imbalanced; quality variation	CNN (ResNet, EfficientNet)	Low-resource deployment on mobile fundus cameras; regulatory approval	Sensitivity ≥ 90% at 95% specificity; AUC-ROC
Credit Scoring (MSME)	SIDBI, CIBIL, fintech NBFCs	Tabular; thin-file (limited history); linguistic diversity	GBT (XGBoost, LightGBM); logistic regression for explainability	Thin-file users; regulatory explainability; geographic bias	Gini coefficient; KS statistic; default rate by decile
Satellite Crop	ISRO, SAC	Multi-spectral	CNN + LSTM;	Ground truth	Overall accurac

Mapping	Ahmedabad	time series; high resolution; seasonal	Random Forest on spectral indices	labelling cost; mixed cropping patterns	y; per-class F1; Kappa coefficient
Judicial Document Classification	eCourts Mission, IIT research groups	Long legal documents; Indian language variation; label scarcity	BERT-style (Legal-BERT, InLegalBERT); few-shot	Domain-specific vocabulary; 22 official languages; judicial sensitivity	Macro-F1; human expert agreement; deployment equity

GBT = Gradient Boosted Trees. GNN = Graph Neural Network. NCIPM = National Centre for Integrated Pest Management (ICAR). SIDBI = Small Industries Development Bank of India. CIBIL = Credit Information Bureau (India) Limited. SAC = Space Applications Centre (ISRO). InLegalBERT is a legal-domain pretrained model for Indian court documents developed at IIT Kharagpur. Sources: NPCI Annual Report (2023); ICAR Digital Agriculture reports; Aravind Eye Care publications; ISRO SAC technical reports.

Part IV · Chapter 9

Responsible Machine Learning

Fairness, Interpretability, Privacy, and Frontiers

Chapter 9

Responsible Machine Learning — Fairness, Interpretability, Privacy, and Frontiers

In 2023, a non-governmental organisation working on digital rights in India documented a pattern in the automated loan approval system of a large digital lending platform. Applicants from certain districts in Uttar Pradesh and Bihar — districts with lower average incomes and historically weaker credit infrastructure — were being declined at substantially higher rates than applicants from comparable districts in Maharashtra and Karnataka, even after controlling for the standard credit risk variables that the platform disclosed as the basis for its decisions. The platform maintained that its model was accurate, and on the overall population, it was. The model's developers, when pressed, could not identify what was driving the differential — their model was a gradient-boosted tree with several hundred features, and they had not examined its behaviour by district.

The case did not involve intentional discrimination. Nobody had instructed the model to discriminate by geography. The model had learned, from historical lending data, that certain postal codes were associated with higher default rates, and it had used that signal legitimately. But postal code, in the Indian context, is a strong proxy for caste and religious community — the product of decades of discriminatory settlement and development patterns that the lending data had absorbed and that the model had faithfully reproduced. The model was accurate. The model was also discriminatory. And the fact that neither the developers nor the regulators could identify the mechanism — because nobody had looked — is precisely the problem that the discipline of responsible machine learning exists to address.

Responsible machine learning is not a single technique but a posture: a commitment to understanding what our models do, to whom, at what cost, and under what conditions. The present

chapter, which closes this volume, examines the principal dimensions of this posture — algorithmic fairness, model interpretability, data privacy, and the open research frontiers that define what responsible ML will need to address next — in the conviction that an engineer who does not engage seriously with these dimensions is not fully practising the discipline this book has sought to teach.

9.1 Algorithmic Fairness: Definitions, Conflicts, and Practice

The question of what it means for a machine learning model to be ‘fair’ does not have a single correct answer. Mathematical formalisations of fairness have proliferated in the research literature since approximately 2016, and a disturbing — but important — set of impossibility results has established that the most intuitive fairness definitions are mutually incompatible: a model that satisfies one cannot, in general, satisfy another simultaneously. Understanding these definitions and their mutual incompatibility is a prerequisite for any serious engagement with the fairness of ML systems.

Demographic parity (also called statistical parity) requires that the proportion of positive predictions be equal across groups defined by a sensitive attribute — gender, religion, caste, geographic origin. A loan approval model satisfies demographic parity if it approves the same fraction of male and female applicants, or the same fraction of applicants from scheduled caste and general category communities. Demographic parity does not require that the model be equally accurate across groups; a model that approves 30% of applicants from each group satisfies demographic parity even if applicants from one group have systematically higher credit risk and the model is therefore making more errors for that group than for the other.

Equalised odds requires that the model’s true positive rate and false positive rate be equal across groups. Requiring equal true positive rates ensures that applicants who are genuinely

creditworthy have equal chances of approval regardless of group membership; requiring equal false positive rates ensures that applicants who would default have equal chances of being approved regardless of group membership. This definition is more demanding than demographic parity and more directly aligned with the intuition that equally qualified applicants should receive equal treatment.

Predictive parity (calibration) requires that the model's predicted probability of the positive outcome be equally accurate across groups — that a predicted probability of 70% means a 70% chance of the event, regardless of which group the applicant belongs to. A well-calibrated model ensures that the meaning of its scores is consistent across groups, which is essential for threshold-based decision-making.

The impossibility results of Chouldechova (2017) and Kleinberg, Mullainathan, and Raghavan (2017) establish that, except in the degenerate case where base rates of the outcome are equal across groups, no model can simultaneously satisfy equalised odds and predictive parity. These results are not merely mathematical curiosities; they reflect a genuine tension in the goals of fairness that cannot be resolved by algorithmic means alone. The choice among fairness definitions is ultimately a normative question — a question about values — that must be answered by the stakeholders in a specific application domain, not by the algorithm designer. The algorithm designer's responsibility is to make the choice explicit and its consequences transparent.

In the Indian context, the sensitive attributes most relevant to fairness analyses are not the same as those that dominate the Western fairness literature (which focuses primarily on gender and racial classification). Caste is the most historically and contemporarily significant axis of social inequality in India, and its relationship to economic outcomes — credit access, employment, health — is extensively documented. Religious community, linguistic group, and geographic origin (rural versus urban;

regional identities associated with specific states or districts) are additional dimensions along which Indian ML systems have the potential to perpetuate or amplify existing inequalities. The Digital Personal Data Protection Act, 2023, prohibits the use of caste and religion as explicit features in commercial ML systems, but as the lending case in the opening vignette illustrates, the prohibition of explicit use does not prevent their implicit reproduction through correlated proxy variables.

Table 9.1 Algorithmic Fairness Metrics — Definitions, Conflicts, and Indian Context

Metric	Definition	What It Ensures	Incompatible With	Indian Application Context	Regulatory Basis
Demographic Parity	$P(\hat{Y}=1\|A=0) = P(\hat{Y}=1\|A=1)$	Equal positive prediction rates across groups	Predictive parity when base rates differ	Credit access across caste/religion (Constitution Art. 15); hiring platforms	Implied by Constitutional equality provisions; no explicit ML statute
Equalised Odds	TPR and FPR equal across groups	Equal error rates; equal opportunity for	Calibration (predictive parity) when	Medical screening across districts; credit	Emerging; referenced in SEBI AI governa

		true positives	base rates differ	default prediction	nce consultations
Equal Opportunity	TPR equal across groups (weaker than equalised odds)	True positives equally likely regardless of group	Demographic parity when base rates differ	Job recommendation; university admission; loan approval	ACM FAccT principles; no Indian statute yet
Predictive Parity (Calibration)	$P(Y=1\|\hat{Y}=p,A=0) = P(Y=1\|\hat{Y}=p,A=1)$	Scores mean same thing across groups	Equalised odds when base rates differ	Clinical risk scores used across hospital types	Medical device regulation; CDSCO guidance on AI diagnostics (draft)
Individual Fairness	Similar individuals treated similarly	No arbitrary inconsistency	Requires defining 'similar' — context-dependent	Credit assessment; judicial risk assessment	No formal definition; general anti-discrimination

					principle
Counterfactual Fairness	Decision unchanged if protected attribute counterfactually altered	Causal — decision not caused by protected attribute	Requires causal model; may conflict with group fairness	Sentencing support tools; admission systems	Aspirational; no legislative basis currently

TPR = True Positive Rate. FPR = False Positive Rate. The impossibility results of Chouldechova (2017) and Kleinberg et al. (2017) establish that demographic parity, equalised odds, and predictive parity cannot simultaneously be satisfied when group base rates differ — which they invariably do in practice. Sources: Dwork et al. (2012); Hardt et al. (2016); Chouldechova (2017); Kleinberg et al. (2017); Digital Personal Data Protection Act, 2023.

9.2 Interpretability: Understanding What Models Learn

The demand for interpretability in machine learning models arises from multiple sources that are worth distinguishing, because they require different kinds of interpretability and are satisfied by different methods. Regulatory compliance requires that decisions affecting individuals be explainable to those individuals and to oversight bodies — a requirement that the Reserve Bank of India's model risk management guidelines impose on credit-scoring models, and that the draft AI governance framework proposed by MeitY extends to government AI systems. Scientific validity requires that models used in research settings be consistent with

domain knowledge and not merely statistically fitting noise. Debugging requires that developers be able to understand why a model makes the errors it does, in order to correct them. Trust requires that users — whether they are doctors, judges, or farmers — be given enough information about the model's reasoning to make informed decisions about how much weight to place on its recommendations.

The most widely used model-agnostic interpretability methods in current practice are SHAP (SHapley Additive exPlanations), introduced by Lundberg and Lee (2017), and LIME (Local Interpretable Model-agnostic Explanations), introduced by Ribeiro, Singh, and Guestrin (2016). Both methods explain individual predictions by quantifying the contribution of each feature to the model's output for a specific instance.

SHAP values are grounded in cooperative game theory: the Shapley value of a feature is its average marginal contribution to the model's prediction across all possible orderings in which features could be introduced. For a prediction on a specific instance, the SHAP value of feature j is the average change in the prediction when feature j is added to all possible subsets of the other features. SHAP values have several attractive theoretical properties: they are the unique feature attribution method that simultaneously satisfies efficiency (attributions sum to the difference between the prediction and the mean prediction), symmetry (features with identical contributions receive identical attributions), dummy (features with no contribution receive zero attribution), and linearity (attributions from multiple models can be summed). The TreeSHAP algorithm computes exact SHAP values for tree-based models (random forests, gradient boosted trees) in polynomial time, making it practical for the large ensembles typically used in production.

LIME approximates the model locally around a specific prediction by fitting a simple, interpretable model — typically a linear regression — to the model's predictions on a set of perturbed

versions of the input. The coefficients of the linear approximation are then presented as the explanation. LIME is computationally cheaper than SHAP for some model types and produces explanations that are, in some user studies, perceived as more intuitive. Its limitations are equally well-documented: the definition of 'locality' and the perturbation strategy substantially affect the explanations, and the local linear approximation may be a poor model of the actual decision boundary even in the vicinity of the explained instance.

Global interpretability — understanding the model's overall behaviour rather than its prediction for a single instance — is provided by partial dependence plots (PDPs), which show the marginal effect of a feature on the model's prediction averaging over the other features; by accumulated local effects (ALE) plots, which correct for feature correlation issues that affect PDPs; and by SHAP summary plots, which display the distribution of SHAP values for each feature across the training dataset, revealing both the direction and the heterogeneity of each feature's effect. These tools, taken together, enable a practitioner to answer the key questions about a model's behaviour: which features matter most, how does each feature affect predictions, and are there unexpected or problematic patterns in the model's behaviour?

9.3 Data Privacy: Differential Privacy and Federated Learning

The training of machine learning models on sensitive personal data — medical records, financial transactions, communications, location histories — raises privacy concerns that are not merely ethical but increasingly legal and regulatory. The Digital Personal Data Protection Act, 2023, India's comprehensive data protection legislation, establishes the principle of data minimisation and requires explicit consent for processing sensitive personal data. The implications for ML training are direct: a model trained on patient records without appropriate consent and safeguards may

violate the Act's provisions, regardless of whether the model itself ever reveals individual records.

Differential privacy, introduced by Dwork, McSherry, Nissim, and Smith in 2006, provides a formal mathematical guarantee about the privacy of individuals whose data is used to train a model. A randomised algorithm M is ε-differentially private if, for any two datasets D and D' that differ in at most one individual's record, and for any output set S, the probability that M(D) produces any given output is at most exp(ε) times the probability that M(D') produces the same output. Informally, the output of a differentially private algorithm is approximately as likely regardless of whether any particular individual's record is included, providing a quantified bound on the information that can be inferred about any individual from the algorithm's output. The privacy parameter ε controls the strength of the guarantee: smaller ε provides stronger privacy at the cost of greater noise added to the algorithm's outputs. Differentially private stochastic gradient descent (DP-SGD), which clips per-example gradients to a maximum norm and adds Gaussian noise before aggregating them into each mini-batch update, is the standard approach for training neural networks with differential privacy. The noise added by DP-SGD degrades model accuracy — the privacy-utility trade-off is fundamental and unavoidable — and finding the optimal operating point on this trade-off for a specific application is an active research problem.

Federated learning, introduced by McMahan and colleagues at Google in 2017, addresses the privacy problem from a different angle: rather than adding noise to the training process, it keeps training data entirely on the devices or institutions that own it, sharing only model updates (gradients) rather than raw data. In the federated training protocol, a central server distributes the current model to participating clients; each client computes a model update using its local data; the updates are aggregated at the central server

— typically by weighted averaging (FedAvg) — and the global model is updated. No raw data ever leaves any client's possession.

Federated learning is of particular relevance in the Indian context for applications where data is inherently distributed and subject to privacy constraints: health data held by individual hospitals in the Ayushman Bharat Digital Mission ecosystem, financial data held by individual banks in the Account Aggregator framework, and agricultural data held by individual Krishi Vigyan Kendras. Federated learning enables the training of models on the combined data of all these institutions without requiring any institution to share its data with a central entity or with other institutions. The technical challenges of federated learning — handling non-IID data distributions across clients, managing communication efficiency, defending against adversarial clients that send malicious updates — are significant and are the subject of active research in India's NLP and healthcare AI communities.

9.4 Open Frontiers: Causal ML, Continual Learning, and Foundation Models

The preceding sections have addressed established and well-studied aspects of responsible machine learning. The present section examines three emerging research directions that will substantially shape the field's development over the next decade and that the graduate student or researcher reading this volume should be aware of as areas of active investigation.

Causal machine learning addresses the fundamental limitation of standard supervised learning, which learns correlational patterns in data rather than causal mechanisms. A model that learns that carrying an umbrella is correlated with rain will not generalise correctly to the policy question of whether distributing umbrellas causes it to rain. This distinction — between correlation and causation, between prediction and intervention — matters enormously for the applications where ML has the greatest potential social impact: medicine (does this treatment cause

recovery?), education (does this intervention cause learning?), economics (does this policy cause poverty reduction?). The potential outcomes framework of Rubin and the do-calculus of Pearl provide the mathematical foundations for causal reasoning from observational data; causal ML seeks to integrate these frameworks with modern machine learning tools to enable models that can answer questions about interventions, not merely about associations.

Continual learning — the ability of a model to learn from a stream of new data without catastrophically forgetting what it has previously learned — addresses a fundamental limitation of standard deep learning: once a model has been trained on a dataset and then trained on new data, it tends to lose its ability to perform well on the original data. This catastrophic forgetting is particularly acute for neural networks, which adjust all their parameters to minimise loss on the current data regardless of its effect on previously learned representations. Continual learning methods — elastic weight consolidation, progressive neural networks, replay-based methods — seek to preserve previously learned knowledge while accommodating new information, enabling truly lifelong learning systems. For the fraud detection and agricultural advisory applications described in the present chapter, where the data distribution changes continuously, continual learning would replace the periodic full retraining that current MLOps systems require with a more efficient and adaptive updating procedure.

Foundation models — large pretrained models that serve as general-purpose starting points for a wide range of downstream tasks — have transformed NLP (through BERT, GPT, and their successors), are beginning to transform computer vision (through ViT and CLIP), and are extending to protein structure (AlphaFold), climate modelling, drug discovery, and scientific reasoning. The responsible deployment of foundation models raises a distinct set of concerns from those addressed in the present chapter: the scale

of their pretraining data makes auditing for bias, intellectual property violations, and privacy risks substantially more difficult than for models trained on curated datasets; their capabilities are emergent and difficult to predict; and their concentration of capability in a small number of large organisations raises concerns about access, governance, and accountability. For the Indian research community, the development of domain-specific and language-specific foundation models — building on the work of AI4Bharat, Sarvam AI, and the emerging ecosystem of Indic AI research — represents both an opportunity and a responsibility: the opportunity to build foundation models that genuinely serve Indian languages and contexts, and the responsibility to do so in a way that embeds the fairness, interpretability, and privacy principles described in this chapter from the start.

Table 9.2 Open Research Problems in Responsible ML — Priority and Horizon

Research Problem	Responsible ML Dimension	Suggested Lead Institution (s)	Why Urgent for India	Indicative Horizon
Fairness auditing methodology for caste and geographic discrimination in Indian ML systems	Fairness	IIT Bombay, IIIT Hyderabad, Namati India, non-profits	Caste discrimination is India's most significant social inequality axis; no standard	2–4 years

			audit tool exists	
Differential privacy for federated learning on Indian health data (Ayushman Bharat ecosystem)	Privacy	IIT Madras, AIIMS, C-DAC, National Health Authority	ABDM creates federated health data infrastructure; privacy-preserving learning needed for clinical ML	3–6 years
Causal ML for agricultural intervention evaluation (does AI advisory increase yield?)	Causal inference	IIT Kharagpur, ICAR, IFPRI India	India has large-scale agricultural ML deployments but no rigorous causal evaluation of their impact	3–5 years
Interpretable ML for Indian judicial applications (risk assessment, case	Interpretability + fairness	IIT Kharagpur (legal AI), National Law University, eCourts	Judicial AI raises highest-stakes interpretability and fairness questions;	4–7 years

outcome prediction)			no established Indian framework	
Continual learning for adversarially drifting fraud detection (UPI ecosystem)	Technical robustness	IIT Delhi, IIT Bombay, NPCI Technology Labs	Fraud patterns evolve adversarially; retraining latency is a security vulnerability	2–4 years
Foundation model governance for Indic languages: bias, provenance, and accountability	Fairness + interpretability + privacy	AI4Bharat (IIT Madras), Sarvam AI, TDIL (MeitY)	Indic foundation models are being built now; governance frameworks must be developed in parallel	2–5 years
Privacy-preserving ML under India's Digital Personal Data	Privacy + legal	IDRBT, RBI Innovation Hub, IIT Delhi, MeitY	DPDPA creates new legal obligations for ML training; practical compliance	1–3 years

Protection Act, 2023			methodologies needed	

ABDM = Ayushman Bharat Digital Mission. TDIL = Technology Development for Indian Languages (MeitY programme). IDRBT = Institute for Development and Research in Banking Technology (RBI). IFPRI = International Food Policy Research Institute. eCourts = e-Courts Mission Mode Project (Supreme Court of India). Sources: Digital Personal Data Protection Act, 2023; National Health Policy; NPCI Annual Report; AI4Bharat publications.