Supervised Learning Algorithms — Part Two

Unit 3: Support Vector Machines, Neural Networks & Backpropagation, Decision Trees and Random Forests

Dr. Gopal Chandra Jana

Dept. of CSE, Sharda University
Machine Learning (CSE473) — Unit 3

July 22, 2026

Course Page:
https://www.gcjana.in/courses/shardauniversity/2601/machine-learning/

Outline

Learning Outcomes of Unit 3

After this unit, a student will be able to...

1. **Derive** the maximum-margin formulation of SVM (hard and soft margin).
2. **Convert** the primal to the dual and apply the **kernel trick**.
3. **Relate** SVM to empirical risk minimization via the **hinge loss**.
4. **Describe** feed-forward network functions and weight-space symmetries.
5. **Derive** the backpropagation equations and **compute** them numerically.
6. **Build** a decision tree using **entropy** and **information gain** (ID3).
7. **Explain** overfitting, pruning, MDL, and **bagging** $\rightarrow$ **random forests**.

Mapping to the Syllabus

A. SVM: linear, non-linear, primal/dual/kernel, modern solvers, ERM, properties. **B.** ANN: feed-forward functions, training, backpropagation. **C.** Decision trees (ID3, overfitting) and random forests.

Which Hyperplane? The Maximum-Margin Idea

The question

If the data is linearly separable, **infinitely many** hyperplanes separate it. Which should we choose?

SVM's answer

Choose the one with the **largest margin** — the greatest distance to the nearest point of either class.

- Intuitively the most **robust** to noise.
- Justified by **statistical learning theory**: larger margin $\Rightarrow$ lower capacity $\Rightarrow$ better generalization bound.

Vapnik & Chervonenkis (1963); modern form Boser, Guyon & Vapnik (1992).

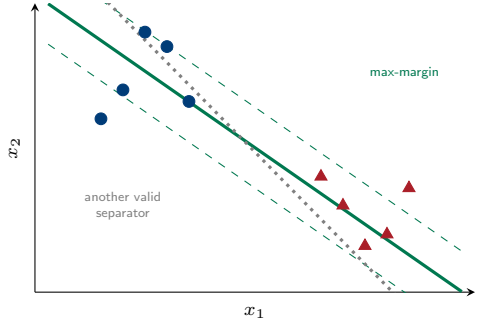


Figure: Both lines separate the data; only one maximises the margin.

Geometry — Functional and Geometric Margin

The hyperplane

$\mathbf{w}^\top \mathbf{x} + b = 0$ with $\mathbf{w}$ **normal** to it; decision rule $\hat{y} = \operatorname{sign}(\mathbf{w}^\top \mathbf{x} + b)$.

Functional margin

$\hat{\gamma}_i = y_i(\mathbf{w}^\top \mathbf{x}_i + b)$ — positive iff correct, but inflatable by scaling $(\mathbf{w}, b) \to (c\mathbf{w}, cb)$.

Geometric margin (the real distance)

$$\gamma_i = \frac{y_i(\mathbf{w}^\top \mathbf{x}_i + b)}{\|\mathbf{w}\|}$$

Scale-invariant: this is the perpendicular distance from $\mathbf{x}_i$ to the hyperplane.

Canonical form

Fix the scale by requiring the closest points to satisfy

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) = 1$$

Then the two supporting hyperplanes are $\mathbf{w}^\top \mathbf{x} + b = \pm 1$ and the **total margin width** is

$$\boxed{\frac{2}{\|\mathbf{w}\|}}$$

Therefore

Maximising the margin $\frac{2}{\|\mathbf{w}\|}$ $\equiv$ **minimising** $\frac{1}{2}\|\mathbf{w}\|^2$ — a convex quadratic objective.

Hard-Margin SVM — The Primal Problem

Optimization problem

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 \qquad \text{subject to} \qquad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \;\geq\; 1, \quad i = 1, \ldots, m$$

Properties

- **Convex quadratic program** with linear constraints $\Rightarrow$ a **unique global** optimum.
- $n + 1$ variables, m constraints.
- Solvable by standard QP solvers.

Limitation

It requires the data to be **perfectly linearly separable**. A *single* outlier makes the feasible set empty — no solution exists. This motivates the **soft margin**.

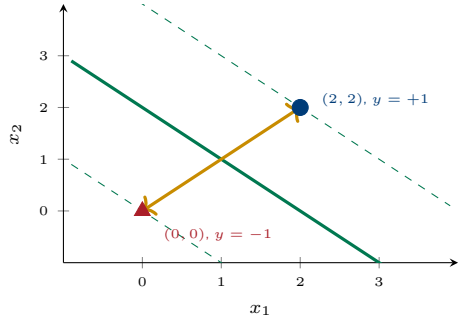


Figure: The tiny example solved on the next slide.

Numerical Example — Hard-Margin SVM

Data

$\mathbf{x}_1 = (2, 2)$ with $y_1 = +1$; $\mathbf{x}_2 = (0, 0)$ with $y_2 = -1$.

Solve the primal

Both points are support vectors, so constraints hold with equality:

$$+1(2w_1 + 2w_2 + b) = 1, \qquad -1(b) = 1 \Rightarrow b = -1$$

Then $2w_1 + 2w_2 = 2$; by symmetry $w_1 = w_2$:

$$\mathbf{w} = (0.5,\ 0.5), \qquad b = -1$$

Margin

$$\|\mathbf{w}\| = \sqrt{0.5^2 + 0.5^2} = 0.7071$$

$$\text{margin} = \frac{2}{\|\mathbf{w}\|} = 2.828 = 2\sqrt{2}$$

— exactly the distance between the two points. ✓

Verify the constraints

$\mathbf{x}_i$	y_i	$\mathbf{w}^\top \mathbf{x}_i + b$	$y_i(\cdot)$
$(2, 2)$	$+1$	$+1$	1 ✓
$(0, 0)$	-1	-1	1 ✓

Decision function

$$\hat{y} = \text{sign}(0.5x_1 + 0.5x_2 - 1)$$

Test $\mathbf{x} = (3, 0)$: $1.5 + 0 - 1 = +0.5 > 0 \Rightarrow$ **+1**.
Test $\mathbf{x} = (0, 1)$: $0 + 0.5 - 1 = -0.5 < 0 \Rightarrow$ **-1**.

Soft-Margin SVM — Slack Variables

Allow violations, but pay for them

Introduce slack $\xi_i \geq 0$ for each example:

$$\min_{\mathbf{w},b,\xi} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i \qquad \text{s.t.} \qquad y_i(\mathbf{w}^\top \mathbf{x}_i + b) \;\geq\; 1 - \xi_i, \quad \xi_i \geq 0$$

Reading the slack

- $\xi_i = 0$: correct, outside the margin.
- $0 < \xi_i \leq 1$: correct, but **inside** the margin.
- $\xi_i > 1$: **misclassified**.

$\sum \xi_i$ upper-bounds the number of training errors.

The role of C

C trades margin width against training violations:

- C **large** $\Rightarrow$ few violations tolerated, narrow margin, **low bias / high variance** (may overfit).
- C **small** $\Rightarrow$ wide margin, more violations, **high bias / low variance**.
- $C \to \infty$ recovers the hard margin.

C is a hyperparameter — tune by cross-validation.

SVM as Regularized Empirical Risk Minimization

Eliminating the slack variables

With $\xi_i = \max\left(0, 1 - y_i(\mathbf{w}^\top \mathbf{x}_i + b)\right)$ the soft-margin problem is *exactly*

$$\min_{\mathbf{w},b} \underbrace{\frac{1}{m}\sum_{i=1}^{m} \max\left(0, 1 - y_i f(\mathbf{x}_i)\right)}_{\text{hinge loss}} + \underbrace{\lambda \|\mathbf{w}\|^2}_{\text{regularizer}}, \quad \lambda = \frac{1}{2Cm}$$

The ERM view

Empirical risk (hinge loss) + **regularization** — exactly the structure of Unit 2's Ridge regression, with a different loss.

Regularization and stability

The $\|\mathbf{w}\|^2$ term makes the solution **stable**: a small change in the data gives a small change in the classifier — and stability implies generalization.

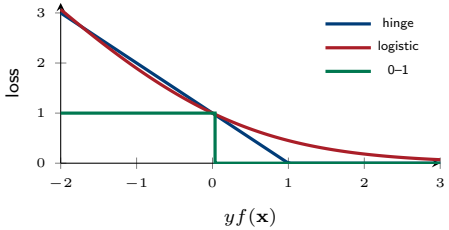


Figure: The hinge loss is a convex upper bound on the 0–1 loss, and is exactly zero once the margin is met.

Numerical Example — Hinge Loss

$$L(y, f) = \max\big(0,\ 1 - y\, f(\mathbf{x})\big)$$

y	$f(\mathbf{x})$	$y\, f(\mathbf{x})$	Loss	Interpretation
$+1$	$+2.5$	$+2.5$	0.0	correct, beyond the margin
$+1$	$+0.8$	$+0.8$	0.2	correct but *inside* the margin
$+1$	-0.4	-0.4	1.4	misclassified
-1	-1.6	$+1.6$	0.0	correct, beyond the margin
-1	$+0.3$	-0.3	1.3	misclassified

Total empirical risk

$$\hat{R} = \frac{0 + 0.2 + 1.4 + 0 + 1.3}{5} = \frac{2.9}{5} = 0.58$$

Key property

The hinge loss is **zero** only when $yf(\mathbf{x}) \geq 1$ — being merely *correct* is not enough; the model must be correct **with a margin**. This is what produces sparse support-vector solutions.

The Lagrangian and the Dual Problem

Lagrangian (hard margin)

With multipliers $\alpha_i \geq 0$:

$$\mathcal{L}(\mathbf{w}, b, \alpha) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i \left[y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 \right]$$

Stationarity conditions

$$\frac{\partial \mathcal{L}}{\partial \mathbf{w}} = 0 \;\Rightarrow\; \boxed{\mathbf{w} = \sum_i \alpha_i y_i \mathbf{x}_i}$$

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \;\Rightarrow\; \sum_i \alpha_i y_i = 0$$

KKT complementary slackness

$$\alpha_i \left[y_i(\mathbf{w}^\top \mathbf{x}_i + b) - 1 \right] = 0$$

- $\alpha_i > 0 \Rightarrow$ the constraint is **active**: $\mathbf{x}_i$ lies *on* the margin — a **support vector**.
- $\alpha_i = 0 \Rightarrow$ the point is irrelevant to the solution.

Substituting back gives the dual

$$\max_{\alpha} \sum_i \alpha_i - \frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j \, \mathbf{x}_i^\top \mathbf{x}_j$$

subject to $\alpha_i \geq 0$ and $\sum_i \alpha_i y_i = 0$.

Two crucial consequences

(i) The solution is **sparse** — determined only by support vectors.
(ii) The data enters **only through inner products** $\mathbf{x}_i^\top \mathbf{x}_j$ — which is precisely what makes the **kernel trick** possible.

Numerical Example — Solving the Dual

Same two points: $\mathbf{x}_1 = (2,2), y_1 = +1;\ \ \mathbf{x}_2 = (0,0), y_2 = -1$

Constraint $\sum \alpha_i y_i = 0$ gives $\alpha_1 - \alpha_2 = 0$, so $\alpha_1 = \alpha_2 = \alpha$.

Build the objective

Inner products: $\mathbf{x}_1^\top \mathbf{x}_1 = 8$, $\mathbf{x}_1^\top \mathbf{x}_2 = 0$, $\mathbf{x}_2^\top \mathbf{x}_2 = 0$.

$$\begin{aligned} W(\alpha) &= \sum_i \alpha_i - \tfrac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \\ &= 2\alpha - \tfrac{1}{2}\left(\alpha^2 (1)(8)\right) \\ &= 2\alpha - 4\alpha^2 \end{aligned}$$

$$\frac{dW}{d\alpha} = 2 - 8\alpha = 0 \ \Rightarrow\ \alpha = 0.25$$

Recover $\mathbf{w}$ and b

$$\begin{aligned} \mathbf{w} &= \textstyle\sum_i \alpha_i y_i \mathbf{x}_i \\ &= 0.25(+1)(2,2) + 0.25(-1)(0,0) \\ &= (0.5,\ 0.5) \quad \checkmark \end{aligned}$$

From any support vector, $b = y_1 - \mathbf{w}^\top \mathbf{x}_1 = 1 - 2 = -1$ ✓

Both routes agree

The dual reproduces the primal solution exactly, and tells us additionally that **both points are support vectors** ($\alpha_i > 0$).

Non-linear Classification — Feature Maps

The idea

Map the data into a higher-dimensional space where it *becomes* linearly separable:

$$\phi : \mathbb{R}^n \to \mathbb{R}^N, \qquad N \gg n$$

then run a linear SVM in that space.

Classic example

Data on concentric circles is not linearly separable in $\mathbb{R}^2$. Map

$$\phi(x_1, x_2) = (x_1^2,\ \sqrt{2}x_1x_2,\ x_2^2)$$

and a **plane** in $\mathbb{R}^3$ now separates them — corresponding to a **circle** back in $\mathbb{R}^2$.

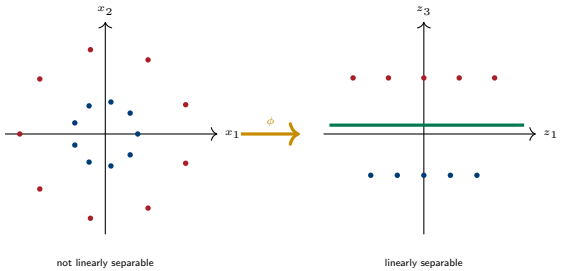


Figure: A non-linear problem becomes linear in a higher-dimensional space.

The Kernel Trick

The key observation

The dual uses the data **only** through inner products. So replace

$$\mathbf{x}_i^\top \mathbf{x}_j \;\longrightarrow\; \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j) \;\equiv\; K(\mathbf{x}_i, \mathbf{x}_j)$$

We never need to compute ϕ explicitly — only the **kernel function** K.

Why this is remarkable

For the polynomial kernel of degree d in n dimensions, ϕ lives in $\binom{n+d}{d}$ dimensions — for the RBF kernel it is **infinite-dimensional**. Yet K costs $O(n)$ to evaluate.

Decision function

$$f(\mathbf{x}) = \text{sign}\Big(\sum_{i \in SV} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b\Big)$$

Only support vectors appear in the sum.

Common kernels

Linear	$\mathbf{x}^\top \mathbf{z}$
Polynomial	$(\mathbf{x}^\top \mathbf{z} + c)^d$
RBF / Gaussian	$\exp(-\gamma \lVert \mathbf{x} - \mathbf{z} \rVert^2)$
Sigmoid	$\tanh(\kappa\, \mathbf{x}^\top \mathbf{z} + c)$

Mercer's condition

K is a valid kernel iff the Gram matrix $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ is **symmetric positive semi-definite** for every finite sample. This guarantees some ϕ exists and keeps the QP convex.

Numerical Example — The Kernel Trick

Take $\mathbf{x} = (1, 2)$, $\mathbf{z} = (3, 4)$ and the polynomial kernel $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z} + 1)^2$

$$\mathbf{x}^\top \mathbf{z} = 1(3) + 2(4) = 11 \qquad \Longrightarrow \qquad K = (11 + 1)^2 = 144$$

Cost: one dot product in 2-D.

Now do it the explicit way

The matching feature map into $\mathbb{R}^6$ is

$$\phi(\mathbf{x}) = \left(1,\ \sqrt{2}x_1,\ \sqrt{2}x_2,\ x_1^2,\ x_2^2,\ \sqrt{2}x_1x_2\right)$$

$$\phi(\mathbf{x}) = (1,\ 1.414,\ 2.828,\ 1,\ 4,\ 2.828)$$

$$\phi(\mathbf{z}) = (1,\ 4.243,\ 5.657,\ 9,\ 16,\ 16.971)$$

$$\phi(\mathbf{x})^\top \phi(\mathbf{z}) = 144 \quad \checkmark$$

The point

Both give 144, but the kernel route needed **2 multiplications**, the explicit route **6-dimensional vectors**. In high dimensions the saving becomes decisive.

RBF on the same points

$\|\mathbf{x} - \mathbf{z}\|^2 = (1 - 3)^2 + (2 - 4)^2 = 8$, so with $\gamma = 0.5$:

$$K = e^{-0.5(8)} = e^{-4} = 0.0183$$

Distant points $\Rightarrow$ near-zero similarity.

Modern Solvers — Sub-gradient and Coordinate Descent

Sub-gradient descent (primal)

The hinge loss is not differentiable at $yf(\mathbf{x}) = 1$, so use a **sub-gradient**:

$$\nabla_{\mathbf{w}} \Big[\max(0, 1 - y_i \mathbf{w}^\top \mathbf{x}_i)\Big] = \begin{cases} -y_i \mathbf{x}_i & y_i \mathbf{w}^\top \mathbf{x}_i < 1 \\ 0 & \text{otherwise} \end{cases}$$

Pegasos update (with regularizer λ):

$$\mathbf{w} \leftarrow (1 - \eta_t \lambda)\mathbf{w} + \eta_t y_i \mathbf{x}_i \quad \text{[if violating]}, \qquad \eta_t = \frac{1}{\lambda t}$$

- Cost per step is $O(n)$ — excellent for **huge** m.
- Converges in $O(1/\epsilon)$ iterations.

Coordinate descent (dual) — SMO

Optimise the dual over **two** multipliers at a time (two, because $\sum \alpha_i y_i = 0$ must be preserved).

- Each 2-variable sub-problem has an **analytic** solution — no QP solver needed.
- **SMO** (Platt, 1998) is the basis of `LIBSVM`.
- Memory-friendly: never forms the full $m \times m$ Gram matrix.

Which to use

Linear SVM on huge data $\Rightarrow$ primal sub-gradient (`LIBLINEAR`, Pegasos).
Kernel SVM $\Rightarrow$ dual coordinate descent (SMO / `LIBSVM`).

Target Functions, Parameter Selection and Issues

Target function of the hinge loss

The population minimiser of the hinge risk is

$$f^*(\mathbf{x}) = \operatorname{sign}\big(2\eta(\mathbf{x}) - 1\big), \quad \eta(\mathbf{x}) = P(y = 1 \mid \mathbf{x})$$

i.e. the **Bayes classifier**. The hinge loss is therefore **classification-calibrated** — minimising it drives us toward the optimal decision rule.

But note

Unlike logistic regression, SVM does **not** estimate $\eta(\mathbf{x})$ itself, so it gives **no calibrated probabilities** (Platt scaling is used as a post-hoc fix).

Parameter selection

- C: margin/violation trade-off.
- γ (RBF): reach of a single example. γ large $\Rightarrow$ narrow, wiggly boundary (overfit); γ small $\Rightarrow$ nearly linear.
- Search a **2-D grid** on a log scale, e.g. $C, \gamma \in \{10^{-3}, \ldots, 10^{3}\}$, by cross-validation.

Practical issues

Scaling is mandatory; training is $O(m^2)$–$O(m^3)$ so large m hurts; multi-class needs one-vs-one/one-vs-rest; the model is **hard to interpret** with non-linear kernels; imbalanced data needs class weights.

Quick Check — Round 1 (SVM)

[Q1]

Why does maximising $\frac{2}{\|\mathbf{w}\|}$ become minimising $\frac{1}{2}\|\mathbf{w}\|^2$?

[Q2]

Compute the hinge loss for $y = +1,\ f(\mathbf{x}) = 0.6$ and for $y = -1,\ f(\mathbf{x}) = +0.5$.

[Q3]

With $\mathbf{x} = (1, 2)$, $\mathbf{z} = (2, 1)$ and $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z} + 1)^2$, compute K.

[Q4]

What does $\alpha_i > 0$ tell you about $\mathbf{x}_i$? What happens to the model if a point with $\alpha_i = 0$ is deleted from the training set?

Answers — Round 1

[A1]

Maximising $2/\|\mathbf{w}\|$ is the same as **minimising** $\|\mathbf{w}\|$; squaring and halving is a monotone transformation that keeps the same minimiser but makes the objective **differentiable and convex quadratic** (nicer for QP solvers).

[A2]

$yf = 0.6 \Rightarrow L = \max(0, 1 - 0.6) = \mathbf{0.4}$ (inside the margin).
$yf = (-1)(0.5) = -0.5 \Rightarrow L = \max(0, 1 + 0.5) = \mathbf{1.5}$ (misclassified).

[A3]

$\mathbf{x}^\top \mathbf{z} = 1(2) + 2(1) = 4$, so $K = (4 + 1)^2 = \mathbf{25}$.

[A4]

$\alpha_i > 0$ means $\mathbf{x}_i$ is a **support vector** lying on (or inside) the margin. Deleting a point with $\alpha_i = 0$ leaves the solution **completely unchanged** — the SVM depends only on its support vectors.

From a Neuron to a Network

The artificial neuron

$$a = \sum_{j=1}^{n} w_j x_j + b, \qquad z = h(a)$$

with h a non-linear **activation function**.

Common activations

Sigmoid	$\sigma(a) = \frac{1}{1+e^{-a}}$
Tanh	$\tanh(a)$, range $(-1, 1)$
ReLU	$\max(0, a)$
Softmax	$\frac{e^{a_k}}{\sum_j e^{a_j}}$ (output)

Why non-linearity is essential

A composition of *linear* maps is still linear — without h, a deep network collapses to a single linear model.

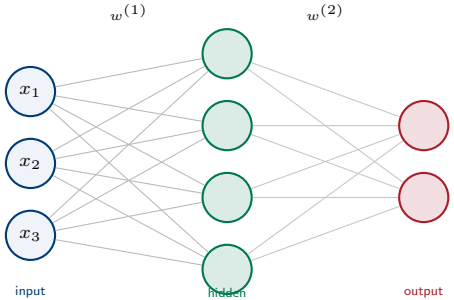


Figure: A two-layer feed-forward network (one hidden layer).

Feed-forward Network Functions

The two-layer network as one formula (Bishop, Ch. 5)

$$y_k(\mathbf{x}, \mathbf{w}) = \sigma\left(\sum_{j=1}^{M} w_{kj}^{(2)}\; h\left(\sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)}\right) + w_{k0}^{(2)}\right)$$

Layer 1 computes **activations** $a_j = \sum_i w_{ji}^{(1)} x_i$, then **hidden units** $z_j = h(a_j)$; layer 2 repeats the pattern.

Compact notation

Absorb the biases with $x_0 = 1$, $z_0 = 1$:

$$\mathbf{z} = h(\mathbf{W}^{(1)}\mathbf{x}), \qquad \mathbf{y} = \sigma(\mathbf{W}^{(2)}\mathbf{z})$$

A network is thus a **composition of parametrised non-linear maps**.

Universal approximation

A two-layer network with **linear outputs** can approximate *any* continuous function on a compact domain to arbitrary accuracy, given **enough hidden units** (Cybenko 1989; Hornik 1991).

But. . .

The theorem is an **existence** result. It says nothing about how many units are needed, or whether training will *find* those weights. Depth is often far more parameter-efficient than width.

Weight-Space Symmetries

Two kinds of symmetry (with $\tanh$ activations)

1. **Sign-flip**: for a hidden unit, negate all its incoming weights. Since $\tanh(-a) = -\tanh(a)$, negating its *outgoing* weights restores the same network function.
   $\Rightarrow 2^M$ equivalent settings.
2. **Unit exchange**: swapping any two hidden units (with their weights) leaves the function unchanged.
   $\Rightarrow M!$ equivalent settings.

Total symmetry factor

$$M! \cdot 2^M$$

equivalent weight vectors give *exactly* the same input–output mapping.

Numeric feel

M	$M!$	2^M	product
3	6	8	48
5	120	32	3 840
10	3 628 800	1 024	$\approx 3.7 \times 10^9$

Consequence

The error surface has **astronomically many equivalent minima**. This is why the weights themselves are not interpretable, and why random initialisation (breaking symmetry) is essential.

Network Training — The Error Function

Sum-of-squares (regression)

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N} \|\mathbf{y}(\mathbf{x}_n, \mathbf{w}) - \mathbf{t}_n\|^2$$

Cross-entropy (classification)

$$E(\mathbf{w}) = -\sum_{n=1}^{N}\sum_{k} t_{nk} \ln y_k(\mathbf{x}_n, \mathbf{w})$$

Both arise as **negative log-likelihoods** — Gaussian noise gives the first, Bernoulli/multinoulli the second.

The hard part

Unlike linear/logistic regression, $E(\mathbf{w})$ is a **highly non-convex** function of $\mathbf{w}$: many local minima, saddle points and flat regions. There is **no closed-form solution**.

Stationary points

We seek $\nabla E(\mathbf{w}) = 0$, but such points may be **local minima**, **maxima** or **saddles**. Classify them by the **Hessian** $\mathbf{H} = \nabla\nabla E$: positive definite $\Rightarrow$ local minimum.

Local Quadratic Approximation

Taylor expansion about a point $\hat{\mathbf{w}}$

$$E(\mathbf{w}) \simeq E(\hat{\mathbf{w}}) + (\mathbf{w} - \hat{\mathbf{w}})^\top \mathbf{b} + \tfrac{1}{2}(\mathbf{w} - \hat{\mathbf{w}})^\top \mathbf{H}(\mathbf{w} - \hat{\mathbf{w}})$$

with $\mathbf{b} = \nabla E|_{\hat{\mathbf{w}}}$ and $\mathbf{H} = \nabla\nabla E|_{\hat{\mathbf{w}}}$.

At a minimum $\mathbf{w}^*$

Here $\nabla E = 0$, so

$$E(\mathbf{w}) \simeq E(\mathbf{w}^*) + \tfrac{1}{2}(\mathbf{w} - \mathbf{w}^*)^\top \mathbf{H}(\mathbf{w} - \mathbf{w}^*)$$

Diagonalising $\mathbf{H}\mathbf{u}_i = \lambda_i \mathbf{u}_i$ and writing $\mathbf{w} - \mathbf{w}^* = \sum_i \alpha_i \mathbf{u}_i$:

$$E = E(\mathbf{w}^*) + \tfrac{1}{2} \sum_i \lambda_i \alpha_i^2$$

Condition for a minimum

$\mathbf{w}^*$ is a local minimum $\iff$ **all** $\lambda_i > 0$ ($\mathbf{H}$ positive definite).

Why it matters for optimisation

The contours are ellipses with axes $\propto \lambda_i^{-1/2}$. The **condition number** $\lambda_{\max}/\lambda_{\min}$ controls convergence: gradient descent must use

$$\eta < \frac{2}{\lambda_{\max}}$$

yet progress along the flattest direction goes as $\lambda_{\min}$ — so a **badly conditioned** Hessian means very slow learning (hence the zig-zag we saw in Unit 2, and the value of momentum/Adam).

Use of Gradient Information — Why It Pays

Counting parameters

Let W be the total number of weights.

- The quadratic approximation has $\frac{W(W+3)}{2}$ independent terms $\Rightarrow$ locating the minimum needs $O(W^2)$ pieces of information.
- **Without** gradients: each function evaluation gives **1** number, and each costs $O(W)$ $\Rightarrow$ total $O(W^3)$.
- **With** gradients: each evaluation of ∇E gives W numbers and (by backpropagation) costs only $O(W)$ $\Rightarrow$ total $O(W^2)$.

The payoff

A whole factor of W — and W can be millions. This single argument is why **gradient-based training** (and hence backpropagation) made neural networks practical.

Gradient-descent variants

- **Batch**: $\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \eta \nabla E$.
- **Stochastic (SGD)**: one example per step — cheap, escapes shallow minima.
- **Mini-batch**: 32–512 examples; the practical standard.
- **Momentum**, RMSProp, **Adam**: adapt the step to curvature/history.

Error Backpropagation — The Derivation

Setup

With $a_j = \sum_i w_{ji} z_i$, $z_j = h(a_j)$, define the **error signal** $\delta_j \equiv \partial E / \partial a_j$.

The key chain-rule step

a_j affects E only through z_j, which feeds every unit k above:

$$\delta_j = \sum_k \frac{\partial E}{\partial a_k} \frac{\partial a_k}{\partial a_j} \;\Rightarrow\; \boxed{\delta_j = h'(a_j) \sum_k w_{kj} \delta_k}$$

Errors flow *backwards* through the same weights.

Output layer

With linear outputs and squared error (or softmax with cross-entropy), the δ takes the beautifully simple form

$$\delta_k = y_k - t_k$$

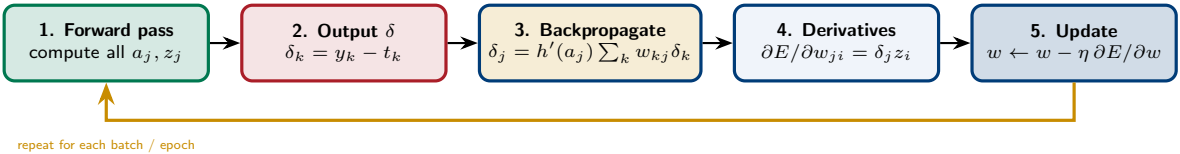
The derivative we want

Since $\partial a_j / \partial w_{ji} = z_i$,

$$\boxed{\frac{\partial E}{\partial w_{ji}} = \delta_j \, z_i}$$

(error at output end) $\times$ (activation at input end).

The Backpropagation Algorithm



Useful derivatives

$$\sigma'(a) = \sigma(a)\big(1 - \sigma(a)\big), \qquad \tanh'(a) = 1 - \tanh^2(a)$$

$$\mathrm{ReLU}'(a) = \begin{cases} 1 & a > 0 \\ 0 & a < 0 \end{cases}$$

Note

Backpropagation computes **gradients**; it is *not* itself the learning rule. Any gradient-based optimiser (SGD, Adam, L-BFGS) then consumes those gradients.

Numerical Example — Forward Pass

A 2–2–2 network with sigmoid activations

Inputs $i_1 = 0.05,\ i_2 = 0.10$; targets $t_1 = 0.01,\ t_2 = 0.99$.
Weights: $w_1..w_4 = 0.15, 0.20, 0.25, 0.30$ with $b_1 = 0.35$; $\quad w_5..w_8 = 0.40, 0.45, 0.50, 0.55$ with $b_2 = 0.60$.

Hidden layer

$$a_{h1} = 0.15(0.05) + 0.20(0.10) + 0.35 = 0.3775$$
$$z_{h1} = \sigma(0.3775) = 0.593270$$
$$a_{h2} = 0.25(0.05) + 0.30(0.10) + 0.35 = 0.3925$$
$$z_{h2} = \sigma(0.3925) = 0.596884$$

Output layer

$$a_{o1} = 0.40(0.593270) + 0.45(0.596884) + 0.60$$
$$= 1.105906 \ \Rightarrow\ y_1 = 0.751365$$
$$a_{o2} = 0.50(0.593270) + 0.55(0.596884) + 0.60$$
$$= 1.224921 \ \Rightarrow\ y_2 = 0.772928$$

Total error

$$E = \tfrac{1}{2}(0.01 - 0.751365)^2 + \tfrac{1}{2}(0.99 - 0.772928)^2$$
$$E = 0.274811 + 0.023560 = 0.298371$$

Numerical Example — Backward Pass

Goal: update w_5 (from z_{h1} to output o_1)

Apply the chain rule:

$$\frac{\partial E}{\partial w_5} = \frac{\partial E}{\partial y_1} \cdot \frac{\partial y_1}{\partial a_{o1}} \cdot \frac{\partial a_{o1}}{\partial w_5}$$

The three factors

$$\frac{\partial E}{\partial y_1} = -(t_1 - y_1) = -(0.01 - 0.751365)$$
$$= 0.741365$$

$$\frac{\partial y_1}{\partial a_{o1}} = y_1(1 - y_1) = 0.751365(0.248635)$$
$$= 0.186816$$

$$\frac{\partial a_{o1}}{\partial w_5} = z_{h1} = 0.593270$$

Combine

$$\delta_{o1} = 0.741365 \times 0.186816 = 0.138499$$

$$\frac{\partial E}{\partial w_5} = \delta_{o1}\, z_{h1} = 0.082167$$

Weight update ($\eta = 0.5$)

$$w_5^+ = 0.40 - 0.5(0.082167) = 0.358916$$

Then

Repeat for w_6, w_7, w_8; propagate δ back to the hidden layer with $\delta_{h1} = z_{h1}(1 - z_{h1}) \sum_k w_{k1}\delta_{ok}$ and update $w_1..w_4$.

Efficiency of Backpropagation

Cost of backpropagation

One forward pass plus one backward pass costs $O(W)$ operations, where W is the number of weights (each weight is touched a constant number of times).

This gives **all W partial derivatives** in a single sweep.

Compare: numerical differentiation

Finite differences,

$$\frac{\partial E}{\partial w_{ji}} \simeq \frac{E(w_{ji} + \epsilon) - E(w_{ji})}{\epsilon}$$

needs one forward pass *per weight*, each costing $O(W)$ $\Rightarrow$ total $O(W^2)$ — and it is numerically inaccurate.

Central differences

$$\frac{E(w + \epsilon) - E(w - \epsilon)}{2\epsilon} + O(\epsilon^2)$$

More accurate but **twice** as expensive — still $O(W^2)$.

Practical use

Numerical differentiation is far too slow for training, but it is invaluable for **gradient checking**: verify a hand-written backprop implementation against finite differences on a small network, then switch it off.

Scale

For $W = 10^6$, backprop is roughly 10^6 times cheaper per gradient than the naive approach.

Quick Check — Round 2 (Neural Networks)

[Q1]

Why must a hidden layer use a non-linear activation?

[Q2]

A network has $M = 5$ hidden units with $\tanh$. How many weight vectors give exactly the same function?

[Q3]

For a sigmoid output with $y = 0.8$ and target $t = 1.0$, compute $\delta = \frac{\partial E}{\partial a}$ for squared error.

[Q4]

State the computational cost of backpropagation and of finite differences for obtaining the full gradient, in terms of W.

Answers — Round 2

[A1]

Because a composition of linear maps is **still linear**: without a non-linear h, any number of layers collapses into a single linear model and the network gains no expressive power.

[A2]

$M! \cdot 2^M = 5! \cdot 2^5 = 120 \times 32 =$ **3840** equivalent weight vectors (unit exchanges $\times$ sign flips).

[A3]

$$\delta = \frac{\partial E}{\partial y}\frac{\partial y}{\partial a} = -(t-y)\,y(1-y) = -(1.0-0.8)(0.8)(0.2) = \textbf{-0.032}.$$

[A4]

Backpropagation: $O(W)$ for the whole gradient. Finite differences: $O(W^2)$ (one $O(W)$ forward pass per weight) — and less accurate.

Decision Tree Representation

Structure

- **Internal node** = test on an attribute.
- **Branch** = outcome of the test.
- **Leaf** = class label (or value).

A path from root to leaf is a **conjunction** of tests; the whole tree is a **disjunction of conjunctions** — i.e. a set of IF–THEN rules.

Suited to problems with. . .

instances described by attribute–value pairs; discrete output; possibly **missing values** or noise; a need for **interpretable** rules.

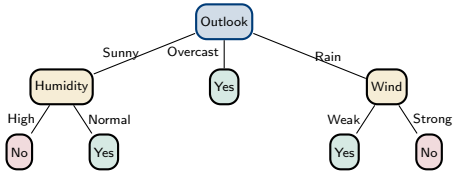


Figure: The *PlayTennis* tree we will build from data.

Entropy — Measuring Impurity

Definition

For a set S with class proportions $p_1, \ldots, p_c$:

$$\text{Entropy}(S) = -\sum_{i=1}^{c} p_i \log_2 p_i$$

For two classes: $\text{Entropy}(S) = -p_+ \log_2 p_+ - p_- \log_2 p_-$.

Interpretation

- 0 bits: the set is **pure** (all one class).
- 1 bit: a **50/50** two-class split — maximum impurity.
- In general $\max = \log_2 c$.
- It is the average number of **bits** needed to encode the class of a random member of S.

Worked

S has 9 positives and 5 negatives ($m = 14$):

$$E(S) = -\tfrac{9}{14} \log_2 \tfrac{9}{14} - \tfrac{5}{14} \log_2 \tfrac{5}{14} = 0.940$$

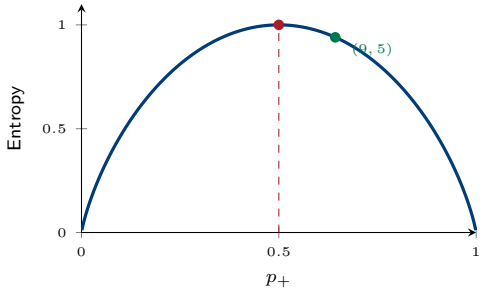


Figure: Entropy peaks at $p_+ = 0.5$ and vanishes at the pure extremes.

Information Gain and the ID3 Algorithm

Information gain

The expected reduction in entropy from splitting S on attribute A:

$$\text{Gain}(S, A) = \text{Entropy}(S) - \sum_{v \in \text{Values}(A)} \frac{|S_v|}{|S|} \text{Entropy}(S_v)$$

ID3 (Quinlan, 1986)

1. If all examples share a class $\Rightarrow$ return a leaf.
2. Otherwise choose the attribute A with the **highest information gain**.
3. Split S on A, creating a branch per value.
4. **Recurse** on each subset with the remaining attributes.

A **greedy**, top-down search — no backtracking.

Bias of information gain

It **favours attributes with many values**. An ID column with a unique value per row gives perfect purity and maximal gain — yet is useless.

Fix: gain ratio (C4.5)

$$\text{SplitInfo}(S, A) = -\sum_{v} \frac{|S_v|}{|S|} \log_2 \frac{|S_v|}{|S|}$$

$$\text{GainRatio} = \frac{\text{Gain}(S, A)}{\text{SplitInfo}(S, A)}$$

Numerical Example — Information Gain (PlayTennis)

S: 14 days, 9 Yes / 5 No, Entropy$(S) = 0.940$

Attribute	Partition (entropy)	Remainder	Gain
Outlook	Sunny(2,3)=.971 Overcast(4,0)=0 Rain(3,2)=.971	0.694	0.247
Humidity	High(3,4)=.985 Normal(6,1)=.592	0.789	0.152
Wind	Weak(6,2)=.811 Strong(3,3)=1.000	0.892	0.048
Temperature	Hot(2,2)=1.0 Mild(4,2)=.918 Cool(3,1)=.811	0.911	0.029

Sample calculation — Outlook

$$\text{Rem} = \tfrac{5}{14}(0.971) + \tfrac{4}{14}(0) + \tfrac{5}{14}(0.971) = 0.694$$

$$\text{Gain} = 0.940 - 0.694 = 0.247$$

Decision

Outlook has the largest gain ⇒ it becomes the **root**. The *Overcast* branch is already pure (entropy 0) and becomes a leaf "Yes" immediately.

Recursing on the Sunny and Rain subsets yields Humidity and Wind respectively — the tree shown earlier.

Gini Index and Other Split Criteria

Gini impurity (CART)

$$\text{Gini}(S) = 1 - \sum_{i=1}^{c} p_i^2$$

The probability of misclassifying a random element if labelled by the class distribution of S.

Numeric on the same data

$$\text{Gini}(9,5) = 1 - \left(\tfrac{9}{14}\right)^2 - \left(\tfrac{5}{14}\right)^2 = 0.459$$

After the Outlook split:

$$\tfrac{5}{14}(0.48) + \tfrac{4}{14}(0) + \tfrac{5}{14}(0.48) = 0.343$$

Reduction $= 0.459 - 0.343 = 0.116$.

Comparison

Entropy	used by ID3/C4.5; $\log$ is costlier
Gini	used by CART; faster, similar trees
Misclass. error	$1 - \max_i p_i$; too insensitive for growing

In practice

Entropy and Gini agree on the chosen split in the large majority of cases — the difference rarely matters. **Pruning strategy matters far more than the impurity measure.**

Overfitting in Decision Trees

Definition (Mitchell)

A hypothesis h **overfits** the training data if there exists h' with

$$\text{err}_{\text{train}}(h) < \text{err}_{\text{train}}(h')$$

but $\quad \text{err}_{\text{test}}(h) > \text{err}_{\text{test}}(h')$

Why trees overfit so easily

A fully grown tree can **isolate every training example** in its own leaf, achieving 0% training error — while learning the **noise**. Deep trees have very **high variance**.

Causes

noisy labels, too few examples at deep nodes, coincidental regularities.

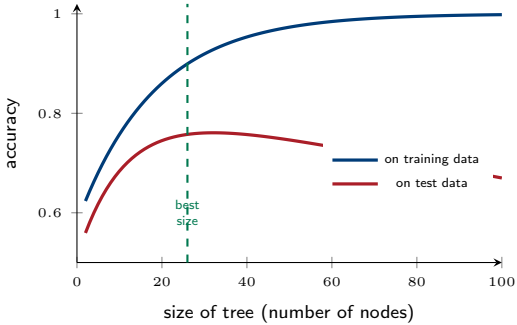


Figure: The classic overfitting curve for decision trees.

Avoiding Overfitting — Pruning and Validation

Two families

- **Pre-pruning** (early stopping): stop growing when a node has too few examples, depth exceeds a limit, or the gain is not statistically significant (χ^2 test). *Risk:* the **horizon effect** — a weak split may enable a strong one below it.
- **Post-pruning**: grow the full tree, then remove subtrees. **Generally preferred.**

Reduced-error pruning

Using a separate **validation set**: repeatedly replace the subtree whose removal most improves validation accuracy with a leaf; stop when no removal helps.

Rule post-pruning (C4.5)

1. Grow the tree fully.
2. Convert each root-to-leaf path into a **rule**.
3. Drop any precondition that improves the estimated accuracy.
4. Sort the rules by accuracy and apply in order.

More flexible than pruning nodes — context differs per path.

Validation methods

Hold-out (e.g. 1/3 for validation); **k-fold cross-validation** (standard $k = 5$ or 10); **cost-complexity pruning** (CART): minimise $R_\alpha(T) = R(T) + \alpha|T|$ with α chosen by CV.

Minimum Description Length and Noise

The MDL principle

Choose the hypothesis minimising the total code length

$$\min_h \quad \underbrace{L(h)}_{\text{describe the model}} \quad + \quad \underbrace{L(D \mid h)}_{\text{describe the data given the model}}$$

Applied to trees

$L(h)$ grows with tree size; $L(D \mid h)$ counts the bits needed to correct the tree's mistakes.

- A **tiny** tree: cheap to describe, many errors to encode.
- A **huge** tree: no errors, but expensive to describe.
- MDL selects the balance — a principled, **validation-free** Occam's razor.

Noise in the data

Two kinds:

- **Attribute noise**: wrong feature values.
- **Class noise**: wrong labels — more damaging.

Handling it

- **Prune** — the single most effective defence.
- Require a **minimum number of samples** per leaf.
- Use **majority vote** at leaves rather than pure splits.
- **Ensembles** (bagging) average noise away.

Missing values

C4.5 sends the instance down *all* branches with fractional weights proportional to the observed value frequencies.

Bagging — Bootstrap Aggregating

Procedure (Breiman, 1996)

Draw B bootstrap samples (size m, *with* replacement); fit a tree on each; aggregate by **majority vote** (classification) or **average** (regression):

$$\hat{f}_{\text{bag}}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B} \hat{f}^{*b}(\mathbf{x})$$

Why it works — variance reduction

For B identically distributed variables with variance σ^2 and pairwise correlation ρ:

$$\text{Var}\left(\tfrac{1}{B}\sum \hat{f}_b\right) = \rho\sigma^2 + \tfrac{1-\rho}{B}\sigma^2$$

The second term $\to 0$ as B grows — but the **first term does not**. Correlation between trees limits the benefit.

Numeric ($\sigma^2 = 1$)

ρ	$B = 1$	$B = 10$	$B = 100$
0.0	1.000	0.100	0.010
0.2	1.000	0.280	0.208
0.5	1.000	0.550	0.505

The insight that leads to random forests

With $\rho = 0.5$, even infinitely many trees cannot push the variance below 0.5. **To gain more, we must decorrelate the trees.**

Out-of-Bag Estimation

How much data does a bootstrap sample miss?

The probability that a particular example is *never* drawn in m draws is

$$\left(1 - \frac{1}{m}\right)^m \xrightarrow[m \to \infty]{} \frac{1}{e} \approx 0.368$$

Numeric convergence

m	$(1 - 1/m)^m$	OOB share
10	0.3487	34.9%
100	0.3660	36.6%
1000	0.3677	36.8%
∞	$1/e = 0.3679$	36.8%

So each tree trains on about **63.2%** of the data.

The OOB error estimate

For each example, average the predictions of only those trees that did *not* see it (about $B/3$ of them). The resulting error is an **almost unbiased** estimate of test error.

Why this is valuable

It comes **free** with training — no separate validation set and no cross-validation loop is needed. A distinctive practical advantage of bagged ensembles.

From Bagging to Random Forests

The extra ingredient (Breiman, 2001)

At **each split**, consider only a **random subset of m_{try} features** out of p, rather than all of them.

$$m_{\text{try}} = \lfloor \sqrt{p} \rfloor \text{ (classification)}$$

$$m_{\text{try}} = \lfloor p/3 \rfloor \text{ (regression)}$$

m_{try} values

p	$\sqrt{p}$ (class.)	$p/3$ (regr.)
9	3	3
16	4	5
100	10	33

Why it helps

If one feature is very strong, *every* bagged tree splits on it first, so the trees are highly correlated (ρ large). Restricting the candidate set forces **diversity**, lowering ρ and hence the variance term $\rho\sigma^2$.

Note the trade-off

Smaller $m_{\text{try}} \Rightarrow$ **less correlation** (good) but **weaker individual trees** (bad). The default values above balance the two; m_{try} is the forest's main tuning knob.

$m_{\text{try}} = p$ recovers plain bagging.

Extra Trees and Properties of Forests

Extremely Randomised Trees

(Geurts, Ernst & Wehenkel, 2006) — add *more* randomness:

- Split thresholds are drawn **at random** rather than optimised.
- Usually grown on the **whole sample** (no bootstrap).

Effect: **lower variance**, slightly higher bias, and noticeably **faster** training (no threshold search).

Randomness ladder

single tree $\rightarrow$ bagging (sample) $\rightarrow$ random forest (sample + features) $\rightarrow$ extra trees (sample + features + thresholds).

Properties of random forests

- **Accurate** and robust with little tuning.
- Handles mixed types; **no scaling** required.
- **Free OOB** error estimate.
- Resistant to overfitting as B grows (more trees never hurts).
- Parallelises trivially.

Limitations

Loses interpretability (hundreds of trees); large memory footprint; slower prediction than one tree; **cannot extrapolate** beyond the training range in regression.

Variable Importance

1. Mean decrease in impurity (Gini importance)

Sum the impurity reduction contributed by each split on feature j, weighted by the number of samples reaching that node, and average over all trees.

- Cheap — computed during training.
- **Biased** toward high-cardinality and continuous features.

2. Permutation importance

Randomly **permute** feature j in the OOB data and measure the drop in OOB accuracy:

$$\text{Imp}(j) = \text{acc}_{\text{OOB}} - \text{acc}_{\text{OOB}}^{\text{perm}(j)}$$

Slower but far more **reliable**.

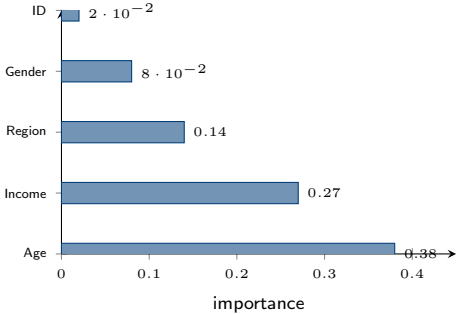


Figure: A typical importance ranking.

Caution

With **correlated** features, importance is *split* among them, so an individually important variable can appear weak. Interpret groups of correlated features together.

Quick Check — Round 3 (Trees & Forests)

[Q1]

Compute the entropy of a set with 8 positives and 8 negatives, and of one with 12 positives and 0 negatives.

[Q2]

A set of 20 examples (12+, 8−) is split into A: (8+, 2−) and B: (4+, 6−). Compute the information gain. [$E(20) = 0.971$, $E(A) = 0.722$, $E(B) = 0.971$]

[Q3]

What fraction of the data is out-of-bag for a large bootstrap sample, and what is it used for?

[Q4]

A dataset has $p = 16$ features. What is the default m_{try} for a classification forest, and why not use all 16?

Answers — Round 3

[A1]

$(8, 8)$: $-0.5 \log_2 0.5 - 0.5 \log_2 0.5 = \mathbf{1.0}$ bit (maximum impurity).
$(12, 0)$: the set is pure, so entropy $= \mathbf{0}$.

[A2]

Remainder $= \frac{10}{20}(0.722) + \frac{10}{20}(0.971) = 0.361 + 0.4855 = 0.8465$.
Gain $= 0.971 - 0.8465 = \mathbf{0.1245}$.

[A3]

About $1/e \approx \mathbf{36.8\%}$ of examples are out-of-bag. They provide a **free, nearly unbiased estimate of test error** (and permutation-based variable importance) without a separate validation set.

[A4]

$m_{\text{try}} = \lfloor \sqrt{16} \rfloor = \mathbf{4}$. Using all 16 would make every tree split on the same dominant features, leaving the trees **highly correlated**; since $\text{Var} = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$, a large ρ blocks further variance reduction.

Summary of Unit 3

What we covered

1. **SVM** — geometric margin, hard-margin QP, soft margin with slack and C.
2. **Duality** — Lagrangian, KKT, sparsity of support vectors; data enters only via inner products.
3. **Kernels** — feature maps, the kernel trick, Mercer's condition; polynomial and RBF.
4. **ERM view** — hinge loss + $\|\mathbf{w}\|^2$ regularizer; stability; modern solvers (Pegasos, SMO).
5. **Neural networks** — feed-forward functions, universal approximation, $M!2^M$ weight symmetries.
6. **Training** — non-convex error, local quadratic approximation, Hessian eigenvalues, why gradients give $O(W^2)$ instead of $O(W^3)$.
7. **Backpropagation** — $\delta_j = h'(a_j)\sum_k w_{kj}\delta_k$, $\partial E/\partial w_{ji} = \delta_j z_i$, cost $O(W)$.
8. **Trees** — entropy, information gain, ID3, Gini, overfitting, pruning, MDL.
9. **Forests** — bagging, OOB (36.8%), decorrelation via m_{try}, extra trees, variable importance.

Practice Questions (Exam Oriented)

Short answer (2–5 marks)

1. Define functional and geometric margin; why is the latter used?
2. State the soft-margin primal and explain the role of C and ξ_i.
3. State Mercer's condition and name three valid kernels.
4. Why does a network with M hidden units have $M!2^M$ equivalent weight vectors?
5. Write the backpropagation formula for δ_j and for $\partial E/\partial w_{ji}$.
6. Define entropy, information gain and gain ratio.
7. What is out-of-bag error and why is it useful?

Long answer (8–10 marks) — include numerics

8. Derive the SVM dual from the primal and state the KKT conditions.
9. For two given points, find $\mathbf{w}$, b and the margin by both primal and dual.
10. Show $\phi(\mathbf{x})^\top \phi(\mathbf{z}) = (\mathbf{x}^\top \mathbf{z} + 1)^2$ numerically for given vectors.
11. Derive the backpropagation equations; perform one forward and one backward pass on a 2–2–2 network.
12. Compute entropy and information gain for all attributes of a given table and choose the root.
13. Explain bagging, OOB and random forests; derive the variance of an average of B correlated trees.

Think & Explore (Take-Home)

Numeric drill

- Points $(1,1), y{=}+1$ and $(-1,-1), y{=}-1$: find $\mathbf{w}$, b, the margin, and the dual α_i.
- Compute the hinge loss for five (y, f) pairs of your choice.
- Build a decision tree by hand for a 10-row table you invent; show every entropy calculation.

Coding (self-learning)

With `scikit-learn`: compare `SVC` (linear vs. RBF) on a 2-D dataset and plot the boundaries; sweep C and γ on a log grid. Then compare `DecisionTreeClassifier` at several depths against `RandomForestClassifier`, and plot the OOB error against B.

Reading

- Bishop, *PRML* — Ch. 5 (networks), Ch. 7 (SVM).
- Hastie, Tibshirani & Friedman, *ESL* — Ch. 9, 12, 15.
- Mitchell, *Machine Learning* — Ch. 3 (decision trees), Ch. 4 (ANNs).
- Breiman, *Random Forests* (2001).

Reminder

Verify your backprop by **gradient checking** against central differences — it catches almost every implementation bug.

References

Textbooks

1. C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006. **[Ch. 5, 7]**
2. T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*, 2nd ed., Springer.
3. T. M. Mitchell, *Machine Learning*, McGraw Hill, 1997. **[Ch. 3, 4]**
4. N. Cristianini, J. Shawe-Taylor, *An Introduction to Support Vector Machines*, Cambridge.
5. G. James et al., *An Introduction to Statistical Learning*, Springer.

Classic papers

6. C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, 20:273–297, 1995.
7. D. Rumelhart, G. Hinton, R. Williams, "Learning representations by back-propagating errors," *Nature*, 323:533–536, 1986.
8. J. R. Quinlan, "Induction of decision trees," *Machine Learning*, 1:81–106, 1986.
9. L. Breiman, "Random forests," *Machine Learning*, 45:5–32, 2001.

Thank You

Any Questions?

"Nothing is more practical than a good theory."
— Vladimir Vapnik

Next: **Unit 4** — unsupervised learning and model evaluation.

Post your doubts on the course page →
https://www.gcjana.in/courses/shardauniversity/2601/machine-learning/#Post-doubts