

UNIT 2

Introduction to Supervised Learning

(11 Hours) — Machine Learning (CACS456) · BCA IV/VIII

⚠ Please Note

*These notes are prepared for quick study and **may contain errors or omissions**. Please cross-check formulas, definitions and worked examples with your textbook, class notes and teacher before relying on them in exams.*

2.1 Classification Problems

In supervised learning the algorithm is given labelled training data (x_i, y_i) and learns a function $f: X \rightarrow Y$. When Y is a **finite set of classes** the task is **classification** (binary: spam / not spam; multi-class: digit 0–9). When Y is a real number it is **regression**.

- **Steps:** collect labelled data → pre-process → split train/test → choose model → train → evaluate → deploy.
- **Decision boundary:** the surface that separates the classes in feature space.
- **Popular classifiers:** Perceptron / neural networks, decision trees, SVM, k-NN, Naïve Bayes, logistic regression.

2.2 Linear Regression — Predicting a Numerical Value

Linear regression models the target as a linear function of the inputs. For one input: $\hat{y} = w_0 + w_1x$; for many inputs: $\hat{y} = w_0 + w_1x_1 + \dots + w_nx_n = \mathbf{w}^T\mathbf{x}$.

Finding the Best-Fit Line

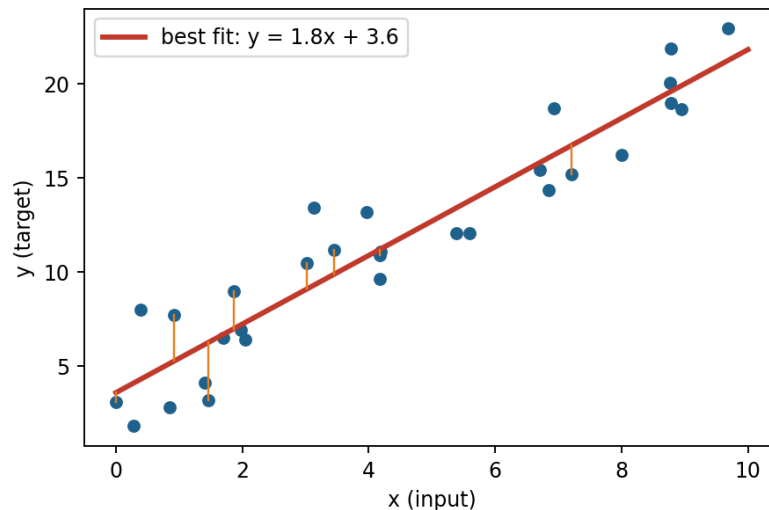


Figure 2.1: Best-fit line minimises the sum of squared vertical errors (orange segments).

The best line minimises the **sum of squared errors (SSE)** / mean squared error — the **method of least squares**:

Least-Squares Formulas (simple regression)

$$\text{Cost: } J(w_0, w_1) = (1/N) \sum (y_i - w_0 - w_1 x_i)^2$$

$$w_1 = \sum (x_i - \bar{x})(y_i - \bar{y}) / \sum (x_i - \bar{x})^2$$

$$w_0 = \bar{y} - w_1 \bar{x}$$

$$\text{Multiple regression (normal equation): } \mathbf{w} = (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{y}$$

Example 2.1

Fit a line to the points (x, y) : (1, 2), (2, 3), (3, 5), (4, 4), (5, 6). Predict y for $x = 6$.

Solution

$$\bar{x} = 15/5 = 3, \bar{y} = 20/5 = 4$$

$$\sum (x - \bar{x})(y - \bar{y}) = (-2)(-2) + (-1)(-1) + (0)(1) + (1)(0) + (2)(2) = 4 + 1 + 0 + 0 + 4 = 9$$

$$\sum (x - \bar{x})^2 = 4 + 1 + 0 + 1 + 4 = 10$$

$$w_1 = 9/10 = \mathbf{0.9}; w_0 = 4 - 0.9 \times 3 = \mathbf{1.3} \rightarrow \hat{y} = 1.3 + 0.9x$$

$$\text{Prediction for } x = 6: \hat{y} = 1.3 + 5.4 = \mathbf{6.7}$$

Gradient Descent (alternative to normal equation)

Start with random weights and repeatedly move opposite to the gradient of the cost: $\mathbf{w}_j \leftarrow \mathbf{w}_j - \alpha \cdot \partial J / \partial \mathbf{w}_j$, where α is the learning rate. Too large α diverges; too small α is slow. For linear regression J is convex, so there is one global minimum.

Evaluation: MSE, RMSE, MAE and $R^2 = 1 - \text{SSE}/\text{SST}$ (fraction of variance explained).

2.3 Perceptron

The perceptron (Rosenblatt, 1957) is the simplest neural network: a single neuron that computes a weighted sum of its inputs and applies a step function, producing a binary output.

Perceptron Model and Learning Rule

Output: $y = 1$ if $(\sum \mathbf{w}_i \mathbf{x}_i + b) \geq 0$, else 0

Update after each example: $\mathbf{w}_i \leftarrow \mathbf{w}_i + \eta (t - y) \mathbf{x}_i$, $b \leftarrow b + \eta (t - y)$

t = target, y = predicted, η = learning rate ($0 < \eta \leq 1$).

- Converges in finite steps if the data are **linearly separable** (Perceptron Convergence Theorem).
- Cannot solve non-linearly separable problems such as **XOR**; this requires multiple layers.

Example 2.2 — Learning the AND gate

Initial $w_1 = w_2 = 0$, $b = 0$, $\eta = 1$. Training data: $(0,0) \rightarrow 0$, $(0,1) \rightarrow 0$, $(1,0) \rightarrow 0$, $(1,1) \rightarrow 1$. Show the first epoch.

Solution (epoch 1)

$(0,0)$: sum = 0 $\rightarrow y = 1$ (since ≥ 0), $t = 0 \rightarrow$ error $-1 \rightarrow w$ stays (inputs 0), $b = -1$

$(0,1)$: sum = $-1 \rightarrow y = 0$, $t = 0 \rightarrow$ no change

$(1,0)$: sum = $-1 \rightarrow y = 0$, $t = 0 \rightarrow$ no change

$(1,1)$: sum = $-1 \rightarrow y = 0$, $t = 1 \rightarrow$ error $+1 \rightarrow w_1 = 1$, $w_2 = 1$, $b = 0$

Further epochs continue until all four are classified correctly (e.g. $w_1 = w_2 = 1$, $b = -1.5$ works).

2.4 Learning Neural Network Structures

A **multi-layer perceptron (MLP)** stacks neurons in layers: input layer $\rightarrow$ one or more hidden layers $\rightarrow$ output layer. Each neuron applies a non-linear activation to a weighted sum. With hidden layers a network can learn non-linear decision boundaries (including XOR).

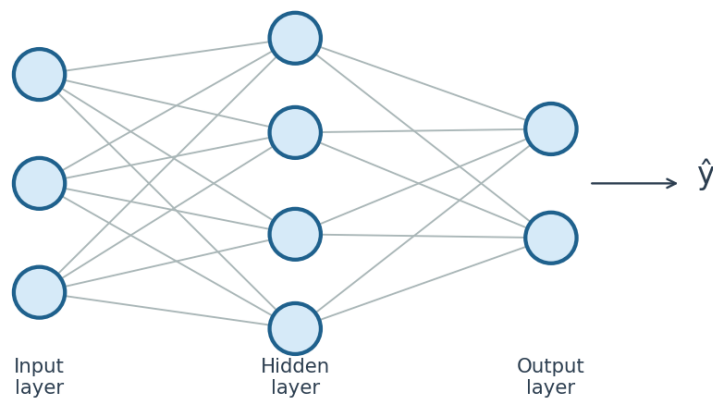


Figure 2.2: A feed-forward neural network with one hidden layer.

Activation	Formula	Notes
Step	1 if $z \geq 0$ else 0	Perceptron; not differentiable
Sigmoid	$\sigma(z) = 1 / (1 + e^{-z})$	Output 0–1; vanishing gradients
Tanh	$(e^z - e^{-z}) / (e^z + e^{-z})$	Output –1 to 1
ReLU	$\max(0, z)$	Default for hidden layers; fast
Softmax	$e^{z_i} / \sum e^{z_j}$	Multi-class output probabilities

Training with Backpropagation

- **Forward pass:** compute activations layer by layer and the output $\hat{y}$.
- **Loss:** compare $\hat{y}$ with target (MSE for regression, cross-entropy for classification).
- **Backward pass:** use the chain rule to compute $\partial \text{Loss} / \partial w$ for every weight, from output to input.
- **Update:** $w \leftarrow w - \eta \partial \text{Loss} / \partial w$ (gradient descent / SGD / Adam). Repeat over many epochs.

Structure design choices: number of hidden layers and neurons, activation functions, learning rate, batch size, regularisation (L2, dropout) and early stopping to avoid overfitting.

2.5 Decision Tree Learning

Decision Tree Representation

A decision tree classifies an instance by sorting it down the tree from the root to a leaf. Each **internal node** tests an attribute, each **branch** corresponds to an attribute value, and each **leaf** assigns a class. A tree is equivalent to a disjunction of conjunctions (if–then rules).

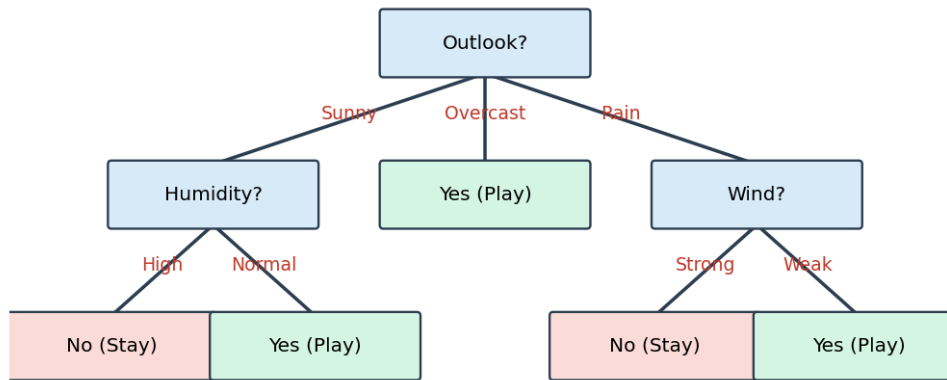


Figure 2.3: Decision tree for “Play outdoors?”. Rule: $(Sunny \wedge Normal) \vee Overcast \vee (Rain \wedge Weak) \rightarrow Yes$.

Appropriate Problems for Decision Tree Learning

- Instances are described by attribute–value pairs (e.g. Wind = Strong).
- The target function has discrete output values (can be extended to regression trees).
- Disjunctive descriptions may be required.
- Training data may contain errors and missing attribute values.
- Interpretability is important (medical diagnosis, loan approval, fault diagnosis).

Basic Decision Tree Algorithm (ID3)

ID3 builds the tree top-down with a greedy strategy: at each node choose the attribute that best splits the data, using **entropy** and **information gain**.

Entropy and Information Gain

Entropy(S) = $-\sum p_i \log_2 p_i$ (0 = pure, 1 = maximally mixed for two classes)

Gain(S, A) = **Entropy**(S) – $\sum (|S_v| / |S|) \cdot \text{Entropy}(S_v)$

Alternative impurity: Gini = $1 - \sum p_i^2$ (CART); C4.5 uses gain ratio.

- 1. If all examples have the same class → make a leaf.
- 2. If no attributes remain → leaf with the majority class.
- 3. Otherwise pick attribute A with highest Gain, create a branch for each value and recurse on each subset.

- **Overfitting** is controlled by pruning (pre-pruning by depth/min samples; post-pruning by reduced-error pruning).

Example 2.3 — Information Gain

A dataset S has 14 examples: 9 Yes, 5 No. Attribute Wind splits it into Weak (8: 6 Yes, 2 No) and Strong (6: 3 Yes, 3 No). Compute $\text{Gain}(S, \text{Wind})$.

Solution

$$\text{Entropy}(S) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14) = 0.410 + 0.530 = \mathbf{0.940}$$

$$\text{Entropy}(\text{Weak}) = -(6/8)\log_2(6/8) - (2/8)\log_2(2/8) = \mathbf{0.811}$$

$$\text{Entropy}(\text{Strong}) = \mathbf{1.000} \text{ (3 vs 3)}$$

$$\text{Gain} = 0.940 - (8/14)(0.811) - (6/14)(1.000) = 0.940 - 0.463 - 0.429 = \mathbf{0.048}$$

2.6 Support Vector Machines (SVM)

Separating Data with a Maximum Margin

Many hyperplanes can separate two linearly separable classes. SVM chooses the one with the **maximum margin** — the largest distance to the nearest training points of either class. These nearest points are the **support vectors**; only they determine the boundary. A larger margin generally gives better generalisation.

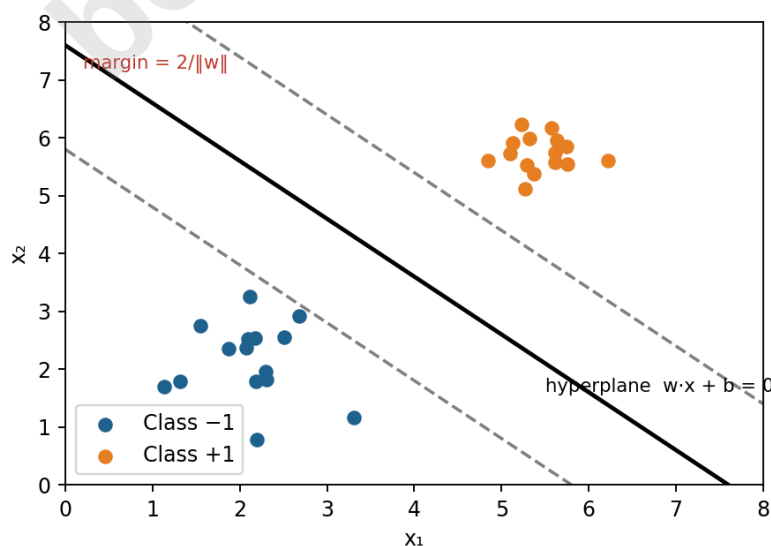


Figure 2.4: Maximum-margin hyperplane with dashed margin boundaries; points on them are support vectors.

Finding the Maximum Margin

The hyperplane is $\mathbf{w} \cdot \mathbf{x} + \mathbf{b} = 0$. With labels $y_i \in \{-1, +1\}$, correct classification with margin requires $y_i(\mathbf{w} \cdot \mathbf{x}_i + \mathbf{b}) \geq 1$. The distance between the two margin lines is $2/\|\mathbf{w}\|$, so maximising the margin = minimising $\|\mathbf{w}\|$.

Optimisation Problem (Hard-margin SVM)

minimise $\frac{1}{2}\|\mathbf{w}\|^2$ **subject to** $y_i(\mathbf{w} \cdot \mathbf{x}_i + \mathbf{b}) \geq 1$ **for all** i

Solved with Lagrange multipliers $\alpha_i \geq 0$ (dual form): maximise $\sum \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j)$, subject to $\sum \alpha_i y_i = 0$.

Only support vectors have $\alpha_i > 0$; $\mathbf{w} = \sum \alpha_i y_i \mathbf{x}_i$; prediction: $\text{sign}(\mathbf{w} \cdot \mathbf{x} + \mathbf{b})$.

- **Soft margin:** allow some violations using slack variables ξ_i ; minimise $\frac{1}{2}\|\mathbf{w}\|^2 + C \sum \xi_i$. Large $C \rightarrow$ narrow margin, fewer errors; small $C \rightarrow$ wider margin, more tolerance.
- **Kernel trick:** replace $\mathbf{x}_i \cdot \mathbf{x}_j$ by $K(\mathbf{x}_i, \mathbf{x}_j)$ to learn non-linear boundaries. Common kernels: linear, polynomial $(\mathbf{x} \cdot \mathbf{y} + 1)^d$, RBF $\exp(-\gamma \|\mathbf{x} - \mathbf{y}\|^2)$.
- **Strengths:** effective in high dimensions, memory-efficient. **Weaknesses:** slow for very large datasets, sensitive to kernel and C choice.

Example 2.4

Support vectors: (1, 1) with $y = -1$ and (3, 3) with $y = +1$. Find the hyperplane and margin.

Solution

By symmetry the optimal $\mathbf{w}$ is along (1,1) and the boundary passes through the midpoint (2,2): $x_1 + x_2 - 4 = 0$.

Scale so that $y_i(\mathbf{w} \cdot \mathbf{x}_i + \mathbf{b}) = 1$: at (3,3): $w_1 \cdot 3 + w_2 \cdot 3 + \mathbf{b} = 1$ with $w_1 = w_2 = a$, $\mathbf{b} = -4a \rightarrow 6a - 4a = 2a = 1 \rightarrow a = 0.5$.

So $\mathbf{w} = (0.5, 0.5)$, $\mathbf{b} = -2$. Margin = $2/\|\mathbf{w}\| = 2/0.707 = 2.83$ (= distance between the two points).

Algorithm	Type	Strength	Weakness
Linear regression	Regression	Simple, interpretable	Only linear relations
Perceptron	Classification	Online, fast	Linearly separable only
Neural network	Both	Learns complex non-linear patterns	Needs lots of data, black box
Decision tree	Both	Interpretable, handles categorical	Overfits, unstable
SVM	Both	Max margin, kernels	Slow on huge data

Key Takeaways — Unit 2

- ✓ Supervised learning uses labelled data: classification (discrete) vs regression (continuous).
- ✓ Linear regression minimises squared error: $w_1 = \Sigma(x-\bar{x})(y-\bar{y})/\Sigma(x-\bar{x})^2$, $w_0 = \bar{y} - w_1\bar{x}$.
- ✓ Perceptron update: $w \leftarrow w + \eta(t - y)x$; works only for linearly separable data.
- ✓ MLPs with non-linear activations are trained by backpropagation + gradient descent.
- ✓ Decision trees use entropy / information gain to choose splits; prune to avoid overfitting.
- ✓ SVM maximises margin $2/\|w\|$; support vectors define the boundary; kernels give non-linearity.

— *End of Unit 2* —

These notes may contain errors — please verify with your textbook and teacher.

bcalink.com