

Table of Contents

Catalog

Handbook	3
Module B	3
1 Regression: Linear and Polynomial	3
1.1 Linear Regression	4
1.2 Polynomial Regression	4
1.3 Overfitting vs. Underfitting	4
1.3.1 Overfitting	4
1.3.2 Underfitting	4
2 Classification	4
2.1 Key Steps in Classification	4
2.2 Performance Evaluation Metrics	5
2.3 Confusion Matrix	5
2.4 Accuracy Matrix	5
2.5 Precision, Recall, and F1-Score	5
3 Principal Component Analysis (PCA)	5
3.1 Process of PCA	5
3.2 Formula	6
3.3 Covariance Matrix	6
3.4 Important Properties	6
4 PageRank	7
5 Neural Networks	7
5.1 Foundations of the Perceptron	7
5.2 Enhancing the Perceptron with Learning Algorithms	7
5.3 Neuron: Basic Building Block	8
5.4 Components of a Neuron	8
5.5 Mathematical Representation	9
5.6 Structure of a Neural Network	9
6 Forward and Backward Propagation	10
6.1 Overview	10
6.2 Forward Pass	10
6.3 Backward Pass	12
6.4 Error Calculation	13
6.5 Activation Functions	14
7 Convolution neural networks(CNN)	14
7.1 Output Size Calculation	14
8 Support Vector Machine (SVM)	14
8.1 Process	15
8.2 Formula	15
8.3 Important Properties	15
9 Supervised Learning - Evaluation Metrics	15
9.1 Confusion Matrix	15
9.2 Accuracy	16
9.3 Precision	16
9.4 Recall	16
9.5 Mean Absolute Error (MAE)	16
9.6 Mean Squared Error (MSE)	16
10 Evaluation Metrics for Unsupervised Learning	17
10.1 1. Silhouette Score	17
10.2 2. Reconstruction Error (Dimensionality Reduction)	17
10.3 Important Properties	19

11 K-Means Clustering	19
11.1 Process	19
11.2 Formula	19
11.3 Important Properties	20
12 Regression	20
12.1 Intro	20
12.2 Logistic Function	20
12.2.1 Equation of Logistic Regression	20
12.3 Cost Function (Log Loss)	20
12.3.1 Updation	21
12.4 Assumptions of Logistic Regression	21
12.5 Evaluation Metrics for Classification	21

Handbook

Module B

1 Regression: Linear and Polynomial

1.1 Linear Regression

Equation: $y = mx + c$

Properties:

- Assumes a linear relationship between x and y .
- Limited for complex patterns.

1.2 Polynomial Regression

Equation:

$$y = a_0 + a_1x + a_2x^2 + \dots + a_nx^n$$

Properties:

- Fits non-linear data by adding polynomial terms.
- Degree n determines model complexity.
- More flexible but prone to overfitting.

1.3 Overfitting vs. Underfitting

1.3.1 Overfitting

Definition: Model captures noise and patterns specific to training data, resulting in poor generalization. **Causes:** High model complexity (e.g., large degree in polynomial regression). **Fixes:**

- Reduce model complexity.
- Use regularization (e.g., Ridge or Lasso).
- Increase training data.

1.3.2 Underfitting

Definition: Model is too simple to capture underlying patterns in the data. **Causes:** Low model complexity (e.g., linear regression for non-linear data). **Fixes:**

- Increase model complexity.
- Use higher-degree polynomial terms.
- Improve feature engineering.

2 Classification

Classification is a supervised learning task where the goal is to assign a given input into one of several predefined categories. The model learns from labeled training data and predicts the category for unseen data.

2.1 Key Steps in Classification

- Data Preprocessing
 - Handling Missing Data
 - Feature Scaling
 - Encoding Categorical Variables
 - Feature Selection
- Splitting the Dataset (Train/Test Split)
- Training the Model
- Making Predictions
- Evaluating the Model

2.2 Performance Evaluation Metrics

2.3 Confusion Matrix

Actual \ Predicted	Positive (1)	Negative (0)
Positive (1)	True Positive (TP)	False Negative (FN)
Negative (0)	False Positive (FP)	True Negative (TN)

2.4 Accuracy Matrix

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

2.5 Precision, Recall, and F1-Score

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 - Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

3 Principal Component Analysis (PCA)

3.1 Process of PCA

1. Standardize the data to have zero mean and unit variance.
2. Compute the covariance matrix of the data.
3. Calculate the eigenvectors and eigenvalues of the covariance matrix.
4. Select the top k eigenvectors corresponding to the largest k eigenvalues (principal components).
5. Project the data onto the k -dimensional subspace formed by the principal components.

3.2 Formula

$$Z = XW$$

- Z : Transformed data (principal components)
- X : Original data matrix (standardized)
- W : Matrix of eigenvectors (principal components)

3.3 Covariance Matrix

$$C = \frac{1}{n} X^T X$$

- C : Covariance matrix
- X : Standardized data matrix
- n : Number of samples

3.4 Important Properties

- Reduces dimensionality while retaining maximum variance.
- Works by finding orthogonal directions of data variability.
- Sensitive to data scaling; standardization is required.
- Useful for noise reduction and feature extraction.

4 PageRank

4.1 Process

1. **Graph Representation:** Represent the web as a directed graph where nodes are web pages and edges are hyperlinks.
2. **Initial Rank Assignment:** Assign an initial rank to each page (e.g., equally distributed or random).
3. **Rank Update:** Iteratively update ranks based on incoming links and the rank of linking pages.
4. **Convergence:** Continue until the rank values stabilize (i.e., the change is below a small threshold).

4.2 Components and Their Effect

- **Nodes (Web Pages):** Pages to rank.
- **Edges (Hyperlinks):** Connections between pages; more links pointing to a page increase its importance.
- **Outlinks (Outgoing Links):** Pages share their rank among all pages they link to.
- **Damping Factor:** Represents the probability of a random jump to another page, preventing rank concentration.
- **Iterations:** Repeated calculations to redistribute ranks until they converge.

5 Neural Networks

5.1 Foundations of the Perceptron

Prediction:

$$y = \text{sign}(w \cdot x + b)$$

- y : Output label (+1 or -1)
- w : Weight vector
- x : Input feature vector
- b : Bias

5.2 Enhancing the Perceptron with Learning Algorithms

Weight Update Rule:

$$w = w + \Delta w \quad \text{where} \quad \Delta w = \eta(y_{\text{true}} - y_{\text{pred}})x$$

- η : Learning rate
- y_{true} : True label
- y_{pred} : Predicted label

5.3 Neuron: Basic Building Block

A neuron performs two main operations:

1. **Net Operation:** Computes the weighted sum of inputs.
2. **Out Operation:** Applies an activation function to produce the output.

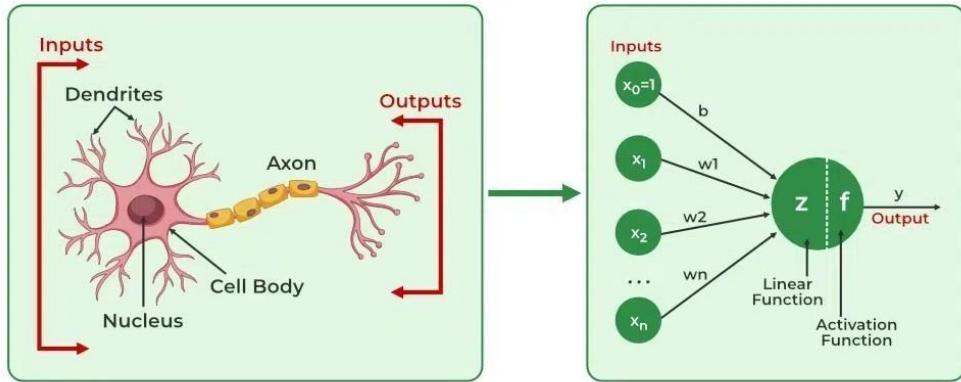


Figure 1: Neuron

5.4 Components of a Neuron

The core elements of a neuron are:

- **Inputs (x_1, x_2, \dots, x_n):** Values that represent features or outputs from the previous layer.
- **Weights (w_1, w_2, \dots, w_n):** Coefficients that signify the importance of each input.
- **Bias (b):** An additional value that shifts the weighted sum to enhance flexibility.
- **Activation Function (f):** A function that introduces non-linearity to the model, enabling it to learn complex patterns.
- **Output (y):** The final result computed by applying the activation function to the weighted sum of inputs and bias.

5.5 Mathematical Representation

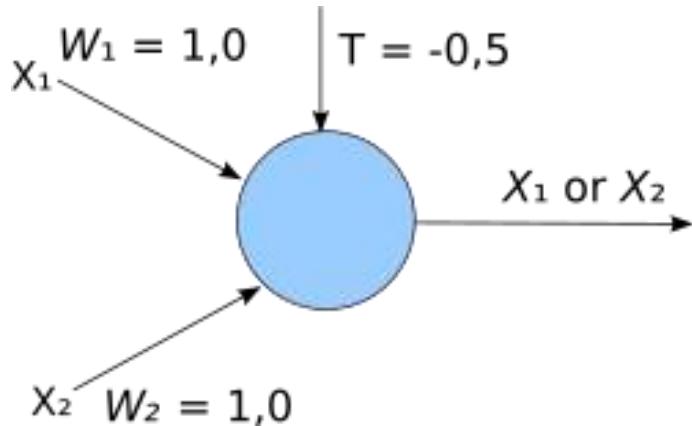


Figure 2: Components of a Neuron

To compute the output of a neuron, first calculate the weighted sum of inputs and bias:

$$\text{Weighted Sum (Net Output)} = \sum_{i=1}^n w_i x_i + b$$

This value is then passed through the activation function:

$$\text{Output (y)} = f \left(\sum_{i=1}^n w_i x_i + b \right)$$

In words: The neuron's output is the result of applying the activation function f to the sum of the weighted inputs ($w_i \times x_i$) and the bias (b).

5.6 Structure of a Neural Network

A neural network consists of three main layers:

- **Input Layer:** Accepts raw data and passes it to the next layer.
- **Hidden Layer(s):** Processes inputs using weights and biases, and learns patterns.
- **Output Layer:** Produces the final predictions.

Each neuron performs two steps:

1. Compute the weighted sum of inputs (net operation).
2. Apply the activation function to get the output (out operation).

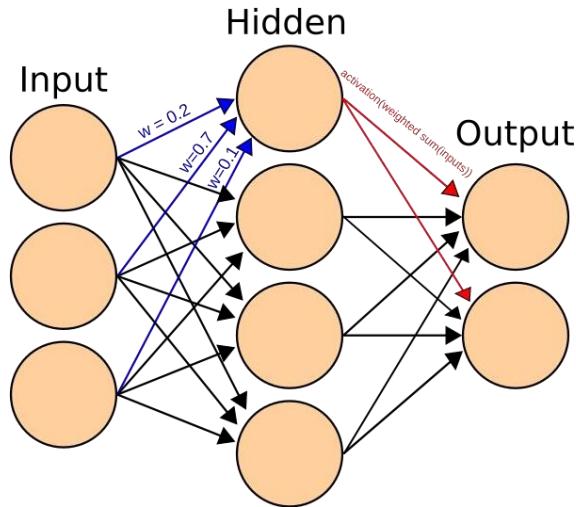


Figure 3: Structure of an Artificial Neural Network

6 Forward and Backward Propagation

6.1 Overview

1. **Forward Pass:** Compute the outputs based on the current weights.
2. **Backward Pass:** Calculate gradients (the rate of change of the error) and propagate errors backward using the chain rule of differentiation.

6.2 Forward Pass

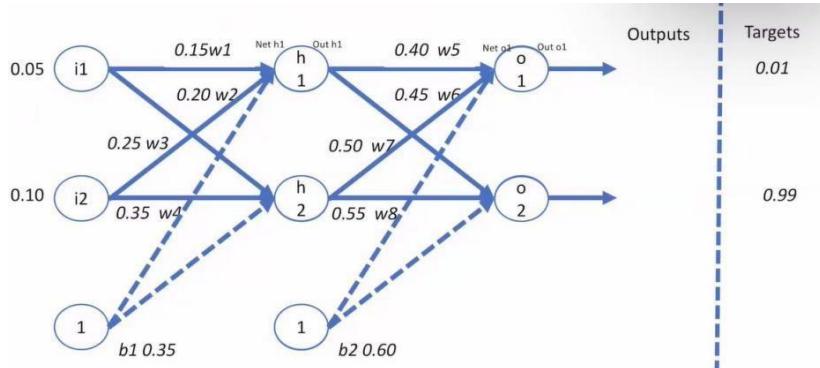


Figure 4: Example of Forward Propagation in a Neural Network

1. **Weighted Sum of Inputs:** For a neuron, the net input is calculated as the weighted sum of the inputs to the neuron, plus a bias term:

$$\text{net}_j = \sum_i w_{ij}x_i + b_j$$

where:

- w_{ij} are the weights associated with the input x_i ,
- b_j is the bias term for neuron j ,
- x_i are the input features.

2. **Activation Function:** The output of the neuron is then obtained by applying an activation function to the net input.

6.3 Backward Pass

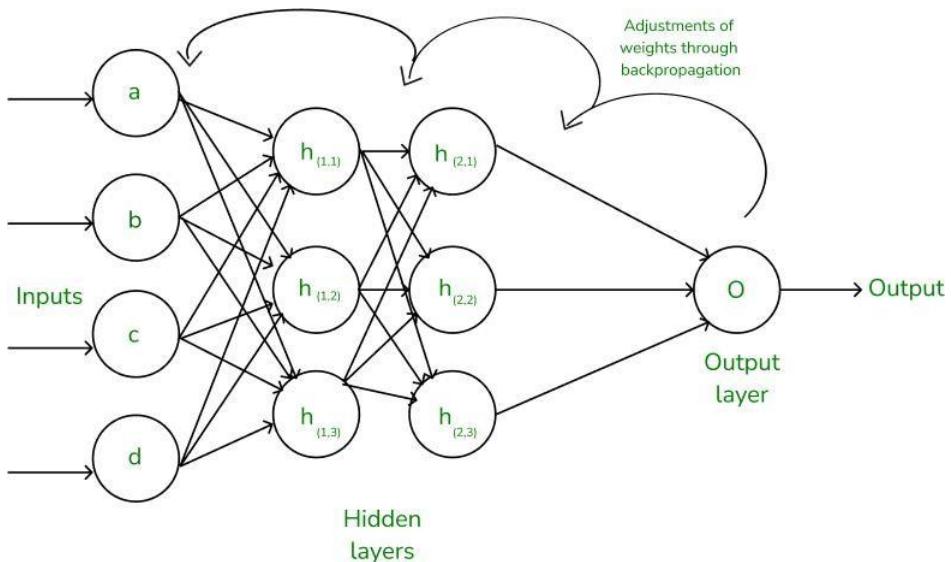


Figure 5: Backward Propagation in an Artificial Neural Network

In the backward pass, we calculate how much each weight in the network contributed to the overall error, and adjust the weights accordingly.

- Error Calculation:** The error for each output neuron is computed using the Mean Squared Error (MSE) formula:

$$E_j = \frac{1}{2}(t_j - y_j)^2$$

where:

- t_j is the target output,
- y_j is the predicted output from the forward pass.

This error measures how far the predicted output y_j is from the desired target output t_j .

- Total Error:** The total error of the network across all output neurons is the sum of the individual errors:

$$E_{\text{total}} = \sum_j E_j$$

- (c) **Gradient Calculation:** To adjust the weights, we need to compute the gradient of the total error with respect to each weight. Using the chain rule of differentiation, the gradient of the total error with respect to a weight w_{ij} is:

$$\frac{\partial E_{\text{total}}}{\partial w_{ij}} = \frac{\partial E_j}{\partial y_j} \cdot \frac{\partial y_j}{\partial \text{net}_j} \cdot \frac{\partial \text{net}_j}{\partial w_{ij}}$$

Each term is computed as follows:

- $\frac{\partial E_j}{\partial y_j} = -(t_j - y_j)$,
- $\frac{\partial y_j}{\partial \text{net}_j} = f'(\text{net}_j)$, where $f'(\text{net}_j)$ is the derivative of the activation function (e.g., for the sigmoid, $f'(\text{net}_j) = y_j(1 - y_j)$),
- $\frac{\partial \text{net}_j}{\partial w_{ij}} = x_i$, the input to the neuron.

- (d) **Weight Update:** The weights are then updated using the gradient descent algorithm:

$$w_{ij}^{\text{new}} = w_{ij} - \eta \cdot \frac{\partial E_{\text{total}}}{\partial w_{ij}}$$

where η is the learning rate, a hyperparameter that controls the step size of the weight update.

6.4 Error Calculation

The total error is calculated using the Mean Squared Error (MSE) function:

$$E_{\text{total}} = \frac{1}{2} \sum_j (t_j - y_j)^2$$

where t_j is the target output and y_j is the predicted output. This error guides the weight update in the backward pass.

6.5 Activation Functions

Activation functions introduce non-linearity into the model. Common types:

- **Sigmoid:** Squashes input into (0, 1). Useful for probabilities.

$$f(x) = \frac{1}{1 + e^{-x}}$$

- **Tanh:** Squashes input into (-1, 1). Centered at zero.

$$f(x) = \tanh(x)$$

- **ReLU:** Replaces negative values with zero.

$$f(x) = \max(0, x)$$

- **Leaky ReLU:** Allows a small gradient for negative values.

$$f(x) = \begin{cases} x & x \geq 0 \\ \alpha x & x < 0 \end{cases}$$

- **ELU:** Exponential Linear Unit, smoothes ReLU's zero gradient issue.

7 Convolution neural networks(CNN)

7.1 Output Size Calculation

The spatial size of the output feature map in a CNN can be calculated using the formula:

$$\bullet \quad O = \left\lfloor \frac{W - K + 2P}{S} \right\rfloor + 1 \quad (1)$$

Where:

- O : Output feature map size
- W : Input size
- K : Kernel (filter) size
- P : Padding size
- S : Stride

8 Support Vector Machine (SVM)

8.1 Process

- (a) **Define Decision Boundary:** Find the hyperplane that best separates classes.
- (b) **Maximize Margin:** Ensure the largest distance (margin) between the hyperplane and nearest data points (support vectors).
- (c) **Kernel Trick (if required):** Map data to a higher dimension if it is not linearly separable.
- (d) **Solve Optimization Problem:** Use quadratic programming to find optimal hyperplane parameters.

8.2 Formula

$$f(x) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$$

- **w:** Weight vector, defining the orientation of the hyperplane.
- **x:** Input vector (data point).
- **b:** Bias term, shifts the hyperplane.
- **sign():** Determines class (+1 or -1).

8.3 Important Properties

- Works well for both linear and non-linear data (with kernel functions).
- Robust to outliers using the soft margin.
- Effective in high-dimensional spaces.
- Relatively memory efficient (uses support vectors only).

9 Supervised Learning - Evaluation Metrics

9.1 Confusion Matrix

A table summarizing the performance of a classification model.

$$\begin{bmatrix} \text{TP} & \text{FP} \\ \text{FN} & \text{TN} \end{bmatrix}$$

- **TP (True Positive):** Correctly predicted positive cases.
- **FP (False Positive):** Incorrectly predicted as positive.
- **FN (False Negative):** Incorrectly predicted as negative.
- **TN (True Negative):** Correctly predicted negative cases.

9.2 Accuracy

$$\text{Accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}$$

9.3 Precision

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

9.4 Recall

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

9.5 Mean Absolute Error (MAE)

$$\text{MAE} = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|$$

- y_i : Actual value.
- \hat{y}_i : Predicted value.
- n : Total number of samples.

9.6 Mean Squared Error (MSE)

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- y_i : Actual value.
- \hat{y}_i : Predicted value.
- n : Total number of samples.

10 Evaluation Metrics for Unsupervised Learning

10.1 1. Silhouette Score

Process:

- (a) For each data point i , calculate:
 - $a(i)$: Average distance to all other points in the same cluster (intra-cluster distance).
 - $b(i)$: Average distance to all points in the nearest different cluster (inter-cluster distance).
- (b) Compute the silhouette score for each point:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

- (c) Calculate the overall silhouette score as the mean of all $s(i)$.

Formula:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

- $s(i)$: Silhouette score for point i ($-1 \leq s(i) \leq 1$).
- $a(i)$: Average distance to points in the same cluster.
- $b(i)$: Average distance to points in the nearest different cluster.

10.2 2. Reconstruction Error (Dimensionality Reduction)

Process:

- (a) Reduce the dimensionality of the data using a method like PCA or autoencoders.
- (b) Reconstruct the data from the reduced dimensions.
- (c) Calculate the error as the difference between the original and reconstructed data.

Formula:

$$\text{Reconstruction Error} = \|X - \hat{X}\|^2$$

- X : Original data matrix.
- \hat{X} : Reconstructed data matrix from reduced dimensions.
- $\|\cdot\|^2$: Squared norm (e.g., Euclidean distance).

10.3 Important Properties

- **Silhouette Score:**
 - Ranges from -1 to 1 .
 - Positive scores indicate well-separated clusters, while negative scores suggest misclassification.
- **Reconstruction Error:**
 - Lower values indicate better preservation of original data in reduced dimensions.
 - Sensitive to noise and dimensionality reduction techniques.

11 K-Means Clustering

11.1 Process

- (a) **Initialize Centroids:** Randomly select k centroids from the dataset.
- (b) **Assign Clusters:** Assign each data point to the nearest centroid.
- (c) **Update Centroids:** Recalculate the centroids as the mean of points in each cluster.
- (d) **Repeat:** Iterate steps 2 and 3 until centroids stabilize or a maximum number of iterations is reached.

11.2 Formula

$$\mu_j = \frac{1}{|C_j|} \sum_{x_i \in C_j} x_i$$

- μ_j : Centroid of cluster j .

- C_j : Set of points in cluster j .
- x_i : Data point assigned to cluster j .
- $|C_j|$: Number of points in cluster j .

Distance Metric: Commonly uses Euclidean distance:

$$d(x, \mu_j) = \sqrt{\sum_{i=1}^n (x_i - \mu_{j,i})^2}$$

11.3 Important Properties

- **Unsupervised Learning:** No labeled data required.
- **Partitioning Algorithm:** Divides data into k clusters.
- **Centroid-Based:** Minimizes within-cluster variance.
- **Sensitive to Initialization:** Results may vary with random centroids.
- **Scalability:** Efficient for large datasets but may struggle with very high dimensions.

12 Regression

12.1 Intro

Logistic Regression is a classification algorithm used to predict categorical outcomes. It estimates the probability that an instance belongs to a particular class. The output is a probability, which is transformed to a binary classification using a threshold (often 0.5).

12.2 Logistic Function

The logistic function (or sigmoid) maps any real-valued number into the range (0, 1).

It is given by:

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

where, e is exponential function

12.2.1 Equation of Logistic Regression

- For a single feature:
 $p(y=1|x) = \sigma(b_0 + b_1x)$
- For multiple features:
 $p(y=1|x) = \sigma(b_0 + b_1x_1 + b_2x_2 + \dots + b_nx_n)$

12.3 Cost Function (Log Loss)

The cost function used in logistic regression is the log loss (or cross-entropy loss):

$$J(\theta) = -\frac{1}{m} \sum [y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})]$$

12.3.1 Updation

The parameters are updated using gradient descent:

$$\theta_j = \theta_j - \alpha \frac{\partial J(\theta)}{\partial \theta_j}$$

where α is the learning rate.

12.4 Assumptions of Logistic Regression

- The dependent variable is binary.
- Observations are independent.
- There is little or no multicollinearity among the independent variables.
- The relationship between independent variables and log-odds is linear.

12.5 Evaluation Metrics for Classification

- Accuracy
- Precision
- Recall
- F1-Score

