Chapter 2

Preliminaries

This chapter presents key definitions and theorems that form the foundation for understanding the concepts discussed in the following chapters.

2.1 Linear Algebra

Linear algebra is fundamental to machine learning, primarily due to the ubiquitous presence of vectors and a comprehensive set of rules for vector manipulation. The resolution of many machine learning tasks, including classifiers and regressors, relies heavily on the principles of linear algebra [19–22].

2.1.1 Quadratic Form

Let $A = [a_{ij}]$ be an $n \times n$ symmetric matrix and $X = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n$. The quadratic form $Q : \mathbb{R}^n \to \mathbb{R}$ defined as [19–22]:

$$\begin{aligned} Q(x) &= x^T Ax \\ &= \begin{pmatrix} x_1 & x_2 & \dots & x_n \end{pmatrix} \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} \\ &= \sum_{i,j=1}^{n} a_{ij} x_i x_j; \quad a_{ij} = a_{ji}, \end{aligned} \tag{2.1}$$

where, A is called the matrix of the quadratic form.

2.1.2 Definite and Semi-definite Quadratic Form

Let $Q(x) = x^T Ax$ be a quadratic form, where A is a symmetric matrix [19–22]. It is said to be

(a) Positive definite if	$x^T Ax > 0;$	$\forall x \neq 0,$
	$x^T Ax = 0;$	$\forall x = 0,$
(b) Positive semi-definite if	$x^T Ax \geq 0;$	$\forall x \neq 0,$
	$x^T Ax = 0;$	$\forall x = 0,$
(c) Negative definite if	$x^T AX < 0;$	$\forall x \neq 0,$

$$\text{(d) Negative semi-definite if} \quad \begin{array}{ll} x^T Ax = 0; & \forall x = 0, \\ x^T AX \leq 0; & \forall x \neq 0, \\ x^T Ax = 0; & \forall x = 0. \end{array}$$

2.1.3 Moore-Penrose Pseudoinverse

Given an $m \times n$ matrix A, the Moore-Penrose pseudoinverse of A is denoted by $A^\dagger$ and defined as $A^\dagger = (A^T A)^{-1} A^T$. This pseudoinverse possesses several key properties [19–22]:

$$\begin{aligned} AA^\dagger A &= A, \\ A^\dagger AA^\dagger &= A^\dagger, \\ (A^\dagger A)^T &= A^\dagger A, \\ (AA^\dagger)^T &= AA^\dagger. \end{aligned} \tag{2.2}$$

2.1.4 Eigenvector and Eigenvalue

For an $n \times n$ matrix A, an eigenvector is a nonzero vector X that satisfies $AX = \lambda X$, where λ is a scalar known as the eigenvalue of A. The equation $AX = \lambda X$ has a non-trivial solution for X if and only if X is an eigenvector corresponding to λ. A scalar λ is considered an eigenvalue of the $n \times n$ matrix A if and only if $(A - \lambda I)X = 0$. [19–22].

2.1.5 Hyperplane

A hyperplane H in $\mathbb{R}^n$ is defined by the set of points $(x_1, x_2, \ldots, x_n)$ that satisfy the linear equation $a_1 x_1 + a_2 x_2 + \cdots + a_n x_n = b$, where the coefficient vector $\mathbf{u} = [a_1, a_2, \ldots, a_n]$ is nonzero. In $\mathbb{R}^n$, this equation represents a line when $n = 2$, a plane when $n = 3$, and a hyperplane in higher dimensions. [19–22].

2.1.6 Gram Matrix

The Gram matrix is a matrix that contains the values of the kernel function evaluated for all pairs within a training set. Given a non-empty set $X \in \mathbb{R}^n$ and a function $k : X \times X \to \mathbb{R}$ (or $\mathbb{C}$), where $x_1, x_2, \ldots, x_n \in X$, the $n \times n$ matrix with elements $K_{ij} = k(x_i, x_j)$ is

known as the Gram matrix, or kernel matrix, corresponding to the points $x_1, x_2, \ldots, x_n$. [19–22].

2.2 Optimisation

2.2.1 Hassian Matrix

The Hessian matrix, a fundamental concept in linear algebra, represents a square matrix composed of second-order partial derivatives of a scalar-valued function. This matrix is pivotal in the analysis of a function's curvature and is extensively utilized to identify local maxima or minima. Specifically, for a function $f : \mathbb{R}^n \to \mathbb{R}$, which possesses second-order partial derivatives that are continuous across its domain, the Hessian matrix, denoted by H, encapsulates the curvature information of f through its entries. The formal definition of the Hessian matrix for function f, considering variables $x_1, x_2, \ldots, x_n$, is given as follows [23, 24]:

$$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{bmatrix} \tag{2.3}$$

This matrix provides a structured way to analyze the curvature and thus the local extremum points of the function, making it an indispensable tool in optimization and analysis within the realm of linear algebra and beyond.

2.2.2 Karush-Kuhn-Tucker (KKT) condition

Consider the optimization problem:

$$\begin{aligned} &\min_{x \in \mathbb{R}^n} f(x) \\ &\text{Subject to constraints,} \\ &g_i(x) \leq 0; \quad i = 1, 2, \ldots, m, \\ &h_j(x) = 0; \quad j = 1, 2, \ldots, r. \end{aligned} \tag{2.4}$$

Define the general Lagrangian, $L(x,\alpha,\beta) = f(x) + \sum_{i=1}^{m} \alpha_i g_i(x) + \sum_{j=1}^{r} \beta_j h_j(x)$, where α and β are Lagrange's multipliers. The Karush-Kuhn-Tucker (KKT) conditions are as follows [23, 24]:

$$\begin{aligned}
\frac{\partial L}{\partial x_i}(x^*,\alpha^*,\beta^*) &= 0; \quad i = 1,2,\ldots,n \quad \text{(Stationary)},\\
\frac{\partial L}{\partial \beta_i}(x^*,\alpha^*,\beta^*) &= 0; \quad i = 1,2,\ldots,r \quad \text{(Stationary)},\\
\alpha_i^* g_i(x^*) &= 0; \quad i = 1,2,\ldots,m \quad \text{(Complementary slackness)},\\
g_i(x^*) &\leq 0; \quad i = 1,2,\ldots,m \quad \text{(Primal feasibility)},\\
\alpha_i^* &\geq 0; \quad i = 1,2,\ldots,m \quad \text{(Dual feasibility)}.
\end{aligned} \tag{2.5}$$

2.3 Functional Analysis

2.3.1 Inner Product:

Let V be a vector space over the field $K = \mathbb{C}$ or $\mathbb{R}$. An inner product on a vector space V is a function $\langle \cdot,\cdot \rangle : V \times V \to K$, which satisfies the following properties [25–27]:

(a) Symmetry: $\langle u,v \rangle = \langle v,u \rangle$

(b) Additivity: $\langle u+v,w \rangle = \langle u,w \rangle + \langle v,w \rangle$

(c) Homogeneity: $\langle \alpha u,v \rangle = \alpha \langle u,v \rangle$

(d) Positivity: $\langle u,u \rangle \geq 0$

(e) Nondegenerate: $\langle u,u \rangle = 0$ if and only if $u = 0$

2.3.2 Inner Product Space:

Let V over the field K. A vector space V with an inner product i.e. $(V, \langle \cdot,\cdot \rangle)$ is called inner product space [25–27].

2.3.3 Dot Product:

For two vectors $\mathbf{x}$ and $\mathbf{y}$ in an n-dimensional space, the dot product is defined as:

$$\mathbf{x} \cdot \mathbf{y} = \sum_{i=1}^{n} x_i y_i$$

Where:

x_i and y_i are the components of the vectors $\mathbf{x}$ and $\mathbf{y}$, respectively [25–27].

2.3.4 Norm

A norm on a vector space X is a function $\|\cdot\| : X \to \mathbb{R}$ that assigns a non-negative real number $\|x\|$ to each vector $x \in X$, and satisfies the following properties [25–27]:

1. $\|x\| \geq 0$ for all $x \in X$ (Non-negativity).

2. $\|x\| = 0 \iff x = 0$.

3. $\|\alpha x\| = |\alpha|\|x\|$ for all $x \in X$ and $\alpha \in \mathbb{R}$ (or $\mathbb{C}$) (Homogeneity).

4. $\|x+y\| \leq \|x\| + \|y\|$ for all $x, y \in X$ (Triangle Inequality).

2.4 Analysis of Dynamical Systems Through Differential Equations

A key approach in the study of dynamical systems is the formulation of first-order ordinary differential equations (ODEs), which mathematically model the systems behavior by describing the changes in its state over time[28–30].

Framework for General System Dynamics

Consider a dynamical system characterized by n first-order ODEs, formally expressed as [28–30]:

$$\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t), \tag{2.6}$$

where:

- $\mathbf{x} = (x_1, x_2, \ldots, x_n)^T$ denotes the state vector of the system, encapsulating the variables of interest that describe the system's current state.
- $\mathbf{f} = (f_1, f_2, \ldots, f_n)^T$ is a vector-valued function, defining the rate of change of each state variable as a function of both the current state and time t.
- t symbolizes time, serving as the independent variable over which the system evolves.

2.5 Existence and Uniqueness of Solutions

A pivotal aspect of analyzing such systems is determining the conditions under which solutions to the differential equations exist and are unique. The Existence and Uniqueness Theorem, frequently associated with the Picard-Lindelf theorem, plays a crucial role in this context. The theorem asserts:

Given an initial value problem (IVP) formulated as $\frac{d\mathbf{x}}{dt} = \mathbf{f}(\mathbf{x}, t)$ with $\mathbf{x}(t_0) = \mathbf{x}_0$, and assuming that the function $\mathbf{f}$ and its partial derivative with respect to $\mathbf{x}$, $\frac{\partial \mathbf{f}}{\partial \mathbf{x}}$, are continuous in a vicinity around the point $(\mathbf{x}_0, t_0)$, then a unique solution $\mathbf{x}(t)$ exists that passes through the point $\mathbf{x}_0$ at time t_0 [28–30].

2.6 Stability Analysis of Equilibrium Points

The analysis of a system's stability, particularly around its equilibrium points, is fundamental to understanding its long-term behavior under small perturbations. Stability considerations focus on the local behavior around equilibrium points, $\mathbf{x}^*$, where the system exhibits no net change [28–30].

Linearization and the Jacobian Matrix

To evaluate the stability of an equilibrium point x^*, the system is usually linearized around x^*, resulting in a linear approximation that reflects the systems behavior in response to small perturbations from equilibrium. The linearized system can be expressed as: [28–30]:

$$\frac{d\mathbf{y}}{dt} = J(\mathbf{x}^*)\mathbf{y}, \tag{2.7}$$

where $\mathbf{y} = \mathbf{x} - \mathbf{x}^*$ signifies a minor perturbation from the equilibrium, and $J(\mathbf{x}^*)$ denotes the Jacobian matrix of $\mathbf{f}$, evaluated at $\mathbf{x}^*$.

Criteria for Determining Stability

The stability of the equilibrium point x^* is determined by the eigenvalues of the Jacobian matrix $J(x^*)$ [28–30]:

- If all eigenvalues possess negative real parts, the equilibrium point $\mathbf{x}^*$ is classified as locally asymptotically stable. This implies that slight deviations from $\mathbf{x}^*$ will diminish over time, returning the system to its equilibrium state.
- Should at least one eigenvalue exhibit a positive real part, the equilibrium point $\mathbf{x}^*$ is deemed unstable. In this scenario, perturbations from $\mathbf{x}^*$ are likely to amplify, driving the system away from its equilibrium.
- In cases where all eigenvalues have non-positive real parts, with at least one having a real part precisely zero, the linearized analysis alone is insufficient to conclusively determine stability. Further investigation is necessitated to ascertain the system's behavior in the vicinity of $\mathbf{x}^*$.

2.7 Optimal Control Theory

Optimal control theory is concerned with finding a control law for a given dynamical system such that a certain performance index is optimized. This involves determining a control input $u(t)$ over a time interval $t \in [t_0, t_f]$ that minimizes (or maximizes) a cost function, while ensuring that the system's dynamics, governed by differential equations, satisfy the desired constraints and initial conditions [31, 32].

Mathematical Formulation of Optimal Control

Consider a dynamical system described by n first-order ordinary differential equations (ODEs):

$$\frac{dx}{dt} = f(x,u,t), \tag{2.8}$$

where:

- $x = (x_1, x_2, \ldots, x_n)^T$ is the state vector of the system, representing the variables of interest.
- $u = (u_1, u_2, \ldots, u_m)^T$ is the control vector applied to the system.
- f is a vector-valued function defining the system's dynamics, a function of the current state, control input, and time.

The goal of optimal control is to find the control $u(t)$ that minimizes (or maximizes) a cost function J defined as [31, 32]:

$$J = \Phi(x(t_f), t_f) + \int_{t_0}^{t_f} L(x(t), u(t), t)\, dt, \tag{2.9}$$

where:

- J is the performance index or cost function to be minimized.
- $\Phi(x(t_f), t_f)$ is the terminal cost, a function of the final state of the system and final time t_f.
- $L(x(t), u(t), t)$ is the running cost, a function representing the cost accumulated over time from t_0 to t_f.

The Hamiltonian and Pontryagin's Minimum Principle

A fundamental concept in optimal control is the Hamiltonian H, defined for this problem as:

$$H(x, u, \lambda, t) = L(x, u, t) + \lambda^T f(x, u, t) \tag{2.10}$$

where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_n)^T$ is the vector of costate variables (also known as Lagrange multipliers or adjoint variables).

Pontryagins Minimum Principle provides necessary conditions for optimality, stating that the optimal control $u^*(t)$ and the optimal trajectory $x^*(t)$ must minimize the Hamiltonian H at each point in time, along with satisfying the system dynamics and boundary conditions. [31, 32].

2.8 Exploration of Z-Control in Continuous Dynamical Systems

In the exploration of control theory for continuous dynamical systems, the refinement of feedback mechanisms is paramount for the regulation and stabilization of system states. This narrative introduces a generalized model that employs a Z-control strategy, enriched with Proportional-Derivative (PD) elements, to meticulously guide the system's state towards a specified target. This advanced control technique is adeptly crafted to manage both the immediate deviation from the desired state, denoted as e_1, and the dynamic rate at which this deviation evolves, denoted as e_2, the derivative of e_1 [33–37].

Z-Control in Continuous Systems: A Refined Approach

Z-control, tailored for continuous systems, presents a sophisticated strategy that synergizes feedback control with the goal of error minimization. This methodology utilizes both the proportional (P) aspect and the derivative (D) component of classical PD control, facilitating precise control over the system's trajectory to align with the desired state [33–37].

Dynamics of Error and Control Objective

The fundamental aim of Z-control within the context of continuous systems is to reduce the error, $e_1 = x^* - x(t)$, where x^* is the desired state. The derivative of this error, $e_2 = \frac{de_1}{dt}$, introduces a forward-looking component to the control strategy, enhancing its predictive capability.

Z-Control Law Formulation

The Z-control law is thus formalized as:

$$u(t) = k_p e_1 + k_d e_2, \tag{2.10}$$

where:

- k_p and k_d are the proportional and derivative gains, respectively. These parameters are meticulously chosen to optimize system stability and responsiveness.
- e_1 is the instantaneous error between the current and desired states.
- e_2 is the rate of change of e_1, offering a glimpse into the system's future behavior under current conditions.

Implementing Z-control in a continuous system requires careful calibration of k_p and k_d to harmonize quick response with stability. Through strategic adjustment of these gains, the control technique can be customized to accommodate varying system dynamics, striking a balance between rapid attainment of the desired state and minimization of overshoot or oscillatory tendencies.

The adoption of Z-control in continuous dynamical systems marks a comprehensive approach to feedback control. By simultaneously addressing the current state deviation and its temporal derivative, this strategy affords a robust mechanism for system regulation. It effectively dampens transient errors while proactively mitigating potential future deviations, ensuring the system's trajectory is congruent with the predetermined target state with heightened precision and stability [33–37].

2.9 Python preliminaries

In the development and implementation of various mathematical algorithms, the Python programming language has been utilized extensively. To develop Kernel-Based models, various Python libraries have been utilized, greatly improving computational efficiency and supporting scientific computing, machine learning, and deep learning tasks. The following Python libraries have played a key role in our programming work. [38, 39]:

1. **NumPy (Numerical Python):** NumPy extends Python by providing robust support for large, multi-dimensional arrays and matrices, along with a wide range of advanced mathematical functions tailored for these data structures. This library is essential for scientific computing, particularly in the fields of machine learning and deep learning.

```
Import numpy as np
```

2. **SciPy (Scientific Python):** Dedicated to scientific and technical computing, SciPy encompasses modules for various computational tasks including optimization, linear algebra, integration, interpolation, and special functions, catering to the diverse needs of science and engineering.

```
Import scipy as sp
```

3. **Pandas:** As an open-source data analysis library, Pandas simplifies data manipulation and analysis in Python by providing intuitive data structures and tools. It is adept at handling numerical tables and time series, making data analysis and manipulation effortlessly manageable.

```
Import pandas as pd
```

4. **Matplotlib:** This library is notable for its ability to create static, animated, and interactive visualizations in Python. Matplotlib is highly versatile, providing a variety of chart types, extensive customization features, and different themes to adjust and personalize plots.

```
From matplotlib import pyplot as plt
```

5. **Seaborn:** Building upon the foundations of Matplotlib, Seaborn excels in data visualization by making statistical graphics more accessible and convenient, especially when working with Pandas data frames. It enhances visualization quality with improved aesthetics and utility functions.

```
Import seaborn as sns
```

6. **Scikit-learn (Sklearn):** A pivotal library in the Python ecosystem for constructing machine learning models.It offers a comprehensive set of efficient tools designed to support machine learning and statistical modeling tasks, such as classification, regression, clustering, and dimensionality reduction.

   (a) **StandardScaler:** It standardizes the features of a dataset by removing the mean and scaling them to have a unit variance.

```
From sklearn.preprocessing import StandardScaler
```

(b) **K-Fold Cross-Validation:** Divides a dataset into K consecutive folds, each serving once as a validation set.

```
From sklearn.model_selection import KFold
```

(c) **Train-Test Split:** Enables random partitioning of a dataset into training and testing subsets.

```
From sklearn.model_selection import train_test_split
```

(d) **Confusion Matrix:** Assesses classification model performance through metrics like accuracy, recall, precision, and F-Score.

```
From sklearn.metrics import confusion_matrix
```

(e) **Classification Metrics:** Functions for computing classification scores and reports.

```
From sklearn.metrics import accuracy_score
From sklearn.metrics import classification_report
```

2.10 Machine Learning

A confusion matrix is an $n \times n$ matrix used to evaluate the performance of a classification model, where n denotes the number of target classes. It facilitates the comparison between the actual target values and the predictions made by the machine learning model. Key performance metrics derived from the confusion matrix include [40, 41]:

1. **Accuracy:** This metric signifies the proportion of true results among the total number of cases examined.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} \times 100\%. \tag{2.11}$$

2. **Precision:** Measures the proportion of positive identifications that were actually correct.

$$\text{Precision} = \frac{TP}{TP+FP} \times 100\%. \tag{2.12}$$

3. **Recall:** Quantifies the proportion of actual positives that were correctly identified.

$$\text{Recall} = \frac{TP}{TP+FN} \times 100\%. \tag{2.13}$$

4. **F-score:** The harmonic mean of precision and recall, providing a balance between the two metrics.

$$F\text{-score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \times 100\%. \tag{2.14}$$

5. **Confusion Matrix**

   A confusion matrix is a tool used to evaluate the performance of a classification model, showing the actual versus predicted classifications. It is structured as follows:

Actual \Predicted	Positive (P)	Negative (N)
Positive (P)	TP	FP
Negative (N)	FN	TN

   Where:

   - **TP (True Positive)**: Number of correctly classified data from the positive class.
   - **FP (False Positive)**: Number of incorrectly classified data from the positive class.
   - **FN (False Negative)**: Number of incorrectly classified data from the negative class.
   - **TN (True Negative)**: Number of correctly classified data from the negative class.

2.10.1 Activation Functions

Activation functions are essential components of neural networks, as they introduce non-linear properties that enable the network to capture and learn intricate patterns within the data. Various activation functions are commonly employed, each with unique features and specific use cases. Some of the most widely used activation functions include Sigmoid, Hyperbolic Tangent (Tanh), Rectified Linear Unit (ReLU), Leaky ReLU, and Exponential Linear Unit (ELU).[42, 43].

Sigmoid Activation Function

The Sigmoid function, often referred to as the logistic function, transforms input values into a range between 0 and 1. Its mathematical definition is as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.15}$$

The Sigmoid function is especially useful in binary classification tasks as it can interpret the outputs as probabilities. However, it is susceptible to the vanishing gradient problem, where gradients tend to zero for very high or low input values, leading to slow or stagnant training.

Hyperbolic Tangent (Tanh) Activation Function

The Tanh function, like the Sigmoid function, introduces non-linearity but maps input values to a range of -1 to 1. Its mathematical expression is given as:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.16}$$

Tanh provides better training performance for certain tasks due to its zero-centered output, making gradient updates more efficient. However, like Sigmoid, it also suffers from the vanishing gradient problem.

Rectified Linear Unit (ReLU) Activation Function

ReLU (Rectified Linear Unit) is among the most popular activation functions in deep learning. It returns the input value for positive inputs and outputs zero for

negative inputs. The mathematical definition is as follows:

$$\text{ReLU}(x) = \max(0, x) \tag{2.17}$$

ReLU (Rectified Linear Unit) is computationally efficient and helps address the vanishing gradient issue, enabling faster learning and improved model performance. However, it is susceptible to the "dying ReLU" issue, where certain neurons become inactive and consistently output zero.

Leaky Rectified Linear Unit (Leaky ReLU) Activation Function

Leaky ReLU is a variation of ReLU that allows a small, positive gradient when the input is negative. It is defined as:

$$\text{Leaky ReLU}(x) = \max(\alpha x, x) \tag{2.18}$$

where α is a small constant. Leaky ReLU addresses the dying ReLU problem by allowing a small gradient when the unit is inactive. It provides all the benefits of ReLU and is generally used to avoid the issue of dead neurons.

Exponential Linear Unit (ELU) Activation Function

ELU is designed to combine the benefits of ReLU and mitigate its drawbacks. For positive inputs, ELU acts like ReLU, and for negative inputs, it outputs values smoothly without making them zero. It is mathematically represented as:

$$\text{ELU}(x) = \begin{cases} x, & \text{if } x > 0 \\ \alpha(e^x - 1), & \text{if } x \leq 0 \end{cases} \tag{2.19}$$

where α is a hyperparameter. ELU helps to push the mean of the activations closer to zero, which speeds up learning. It also avoids the dying ReLU problem and ensures a noise-robust deactivation state.

Each of these activation functions has its strengths and trade-offs. The choice of an activation function depends on the specific requirements of the neural network architecture and the nature of the task at hand. Selecting the right activation function is crucial for the learning process and the performance of the neural network.

2.10.2 K-Fold Cross Validation Method

K-Fold Cross-Validation is a resampling method used to assess the performance of machine learning models on a limited dataset. This technique divides the dataset into K groups or folds, where K is a user-defined parameter. As such, the procedure is often called K-fold cross-validation. When a specific value for K is chosen, it may be used in place of K in the reference to the model, such as $K = 5$ becoming 5-fold cross-validation [40, 41].

2.10.3 K-Fold Cross-Validation Procedure

(a) **Shuffle the Dataset Randomly:** The original dataset is randomly shuffled to ensure that the cross-validation process does not exhibit a bias in terms of the order of samples.

(b) **Split the Dataset into K Groups:** The shuffled dataset is split into K groups, or "folds", of approximately equal size. The exact number of samples in each fold may vary if the total number of samples is not evenly divisible by K.

(c) **For Each Unique Group:** For each unique fold:

   i. One fold is set aside to serve as the test dataset, while the remaining K-1 folds are utilized as the training dataset.

   ii. Fit a Model on the Training Set and Evaluate it on the Test Set: A model is trained on the training set and evaluated on the test set. The evaluation score is retained and a new model is discarded.

   iii. Retain the Evaluation Score and Discard the Model: The performance measure of the model on the test set is recorded. After that, the model is discarded as it will no longer be needed.

(d) **Summarize the Skill of the Model Using the Sample of Model Evaluation Scores:** The performance measures recorded for each fold are then aggregated, typically by taking their mean, providing an estimate of the model's predictive performance. The variance of the performance scores can also be assessed to understand the model's stability.

2.10.4 Key Points of K-Fold Cross-Validation

- **Bias-Variance Trade-Off:** K-fold cross-validation helps in understanding the trade-off between bias and variance in the model. Lower values of K will have a higher bias but lower variance, while higher values of K will have lower bias but higher variance.
- **Model Selection and Configuration:** It is a good practice to perform K-fold cross-validation to confirm the performance of a model and to tune model hyperparameters. It helps in avoiding overfitting and underfitting and ensures that the model generalizes well to unseen data.
- **Computational Cost:** Although K-fold cross-validation provides a robust estimate of the model's performance, it is computationally expensive as the model needs to be trained and evaluated K times.

In conclusion, K-fold cross-validation is an essential technique in machine learning for estimating the performance of predictive models. It provides a solid foundation for model assessment and selection, ensuring that the models not only fit the training data well but also have the generalizability to perform well on unseen data. As with any methodology, it's crucial to understand the context and the specifics of the dataset when applying K-fold cross-validation to ensure meaningful and reliable results.

2.11 Feature Extraction Methods

2.11.1 Grey Level Co-occurrence Matrix

The Gray Level Co-occurrence Matrix (GLCM) method is a statistical approach used in image processing and analysis to examine the texture of an image. GLCM quantifies how often different combinations of pixel brightness values (gray levels) occur in an image, or a portion of an image. It's a method of examining the spatial relationship between pixels and is widely used in various applications, including medical imaging, remote sensing, and machine vision [44–48].

Fundamentals of GLCM:

- The foundation of GLCM is the assessment of how frequently a pixel with intensity (gray level) value i occurs horizontally, vertically, or diagonally adjacent to a pixel with the value j.
- For a given displacement vector $(\Delta x, \Delta y)$, a GLCM matrix $P(x, y, \Delta x, \Delta y)$ is formed, where each element $P(i, j)$ represents the number of occurrences of the pixel with value i in the specified spatial relationship with a pixel with value j.

Matrix Formation:

To create the GLCM, the image is traversed pixel by pixel. For each pixel, the neighboring pixel is identified based on the defined displacement vector. The values of these pixel pairs are then used to increment the corresponding element in the GLCM.

Normalization:

The raw counts in the GLCM are normalized to facilitate comparisons between different images or image regions. This is done by dividing each element in the GLCM by the total number of pixel pairs used to form the matrix, resulting in a matrix where each element represents the probability of the occurrence of a pixel pair.

Feature Extraction:

From the normalized GLCM, various statistical measures can be extracted to describe the texture of the image. Commonly used features include:

- **Contrast:** Measures the local variations in the GLCM.
  Mathematical Equation: $Contrast = \sum_{ij} P(i, j) \cdot (i - j)^2$.
- **Correlation:** Assesses how a pixel is correlated to its neighbor across the whole image.
  Mathematical Equation: $Correlation = \frac{\sum_{ij}(i-\mu_i)\cdot(j-\mu_j)\cdot P(i,j)}{\sigma_i \cdot \sigma_j}$,
  where μ_i, μ_j are the means and σ_i, σ_j are the standard deviations of the row and column sums of the GLCM, respectively.

- **Energy (Angular Second Moment):** Provides the sum of squared elements in the GLCM, indicating the uniformity or homogeneity of the image.
  Mathematical Equation: $Energy = \sum_{ij} P(i,j)^2$.
- **Homogeneity:** Measures the closeness of the distribution of elements in the GLCM to the GLCM diagonal.
  Mathematical Equation: $Homogeneity = \sum_{ij} \frac{P(i,j)}{1+|i-j|}$.
- **Dissimilarity:** Complements the contrast feature by measuring the difference between pairs of elements in the GLCM.
  Mathematical Equation: $Dissimilarity = \sum_{ij} P(i,j) \cdot |i-j|$.

In conclusion, the Gray Level Co-occurrence Matrix method is a robust statistical tool for examining the texture of an image. By analyzing the frequency of co-occurring pixel intensities at a specified spatial relationship, GLCM allows for the extraction of meaningful patterns and features that are fundamental in the interpretation and classification of images in various scientific and industrial applications. The versatility and effectiveness of GLCM make it a staple in the field of image analysis and computer vision [44–48].

2.11.2 Local Binary Pattern (LBP)

The Local Binary Pattern (LBP) methodology is a robust and computationally efficient technique for texture analysis and image classification, often used within the scope of computer vision and image processing. The LBP operator, originally introduced by Ojala, Pietikinen, and Harwood in 1996, has since become a standard method for texture analysis due to its discriminative power and computational simplicity. The method is particularly well-suited for applications requiring real-time processing and has been successfully applied in various domains, including facial recognition, motion analysis, and environment monitoring. This section provides a comprehensive overview of the LBP method, detailing its theoretical underpinnings, algorithmic implementation, and mathematical formulation [49].

Fundamentals of Local Binary Pattern (LBP):

The LBP operator works by comparing each pixel in the image with its surrounding neighbours. It assigns a binary value to the neighbours depending on whether

their intensity is greater than or less than the centre pixel's intensity. The binary values for each neighbour are then concatenated to form a binary number. This number, or its decimal equivalent, is used as a label to characterize the local texture around the central pixel [49].

Algorithmic Implementation:

Consider a pixel at location (x_c, y_c) with intensity I_c in a grayscale image. Define a neighbourhood around this pixel with P equally spaced pixels on a circle of radius R, denoting their intensities as I_p, where $p = 0, 1, \ldots, P-1$. The LBP value for the centre pixel is computed by comparing each neighbours intensity with the centre pixel's intensity. The comparison is formulated as a binary result s, defined as:

$$s(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{if } x < 0 \end{cases}. \tag{2.20}$$

The binary results for all neighbours are then combined to form the LBP code for the center pixel, mathematically expressed as:

$$LBP_{P,R} = \sum_{p=0}^{P-1} s(I_p - I_c) \cdot 2^p. \tag{2.21}$$

Rotational Invariance and Uniform Patterns:

To achieve rotational invariance, the LBP code is rotated to its smallest numerical value. This allows the LBP operator to be insensitive to the rotation of textures in the image. A uniform LBP pattern is defined as an LBP code with at most two bitwise transitions from 0 to 1 or vice versa when the bit pattern is considered circular. Uniform patterns account for a vast majority of texture patterns and can be used to reduce the dimensionality of the feature vector, enhancing computational efficiency [49].

Feature Vector Construction:

The histogram of the LBP codes computed for the pixels in the image or a region of interest serves as a feature descriptor. This histogram encapsulates the distribution of local micro-patterns over the image and is robust to monotonic grayscale changes [49].

LBP Energy and LBP Entropy:

In the realm of texture analysis and pattern recognition using Local Binary Patterns (LBP), two significant derived measures, LBP Energy and LBP Entropy, play a pivotal role in providing comprehensive textural and structural information about images. These measures, extracted from the LBP histograms, offer deep insights into the uniformity and complexity of local patterns, making them invaluable for various applications in image processing and computer vision.

LBP Energy: LBP Energy is a measure that quantifies the uniformity and regularity of local binary patterns within an image. Mathematically, for a given LBP histogram H with n bins, the LBP Energy E_{LBP} can be computed as the sum of the squares of the histogram values:

$$E_{LBP} = \sum_{i=1}^{n} H(i)^2. \tag{2.22}$$

High values of LBP Energy indicate regions with prevalent and repetitive local binary patterns, signifying uniform and consistent textures.

LBP Entropy: LBP Entropy measures the randomness or complexity of local binary patterns in the image. It is a statistical measure that quantifies the unpredictability or disorder in the distribution of LBP codes. For the LBP histogram H with n bins, the LBP Entropy S_{LBP} is given by the Shannon entropy formula:

$$S_{LBP} = -\sum_{i=1}^{n} p(i) \log_2 p(i), \tag{2.23}$$

where $p(i)$ is the normalized histogram value or the probability of occurrence of the i^{th} LBP pattern. High values of LBP Entropy correspond to images or regions with complex and varied texture patterns, indicating the presence of diverse and irregular local structures.

Role in Texture Analysis and Feature Extraction: LBP Energy and LBP Entropy serve as complementary measures to the basic LBP histogram. While the histogram provides a distribution of local binary patterns, LBP Energy and LBP Entropy offer a condensed and insightful summary of the textural properties. Incorporating LBP Energy and LBP Entropy into the feature vector enhances the

discriminative power of the texture descriptor. It allows for a more nuanced differentiation between images or regions with varying degrees of texture uniformity and complexity. The combined use of LBP Energy, LBP Entropy, and the LBP histogram facilitates robust image classification, texture segmentation, and pattern analysis tasks. It provides a multifaceted understanding of the textural characteristics, crucial for applications in medical imaging, material inspection, and environmental monitoring [49].

In summary, LBP Energy and LBP Entropy are crucial metrics that augment the Local Binary Pattern methodology, offering a holistic perspective on image texture. By quantifying the uniformity and complexity of local patterns, these measures contribute significantly to the robustness and descriptiveness of texture analysis and image classification systems. Their integration with the basic LBP features paves the way for advanced applications in computer vision, reinforcing the role of Local Binary Patterns as a powerful tool in image analysis and pattern recognition [49].

2.11.3 Gabor Filter

The Gabor filter, named after Dennis Gabor, is a linear filter used in image processing for edge detection, texture analysis, and feature extraction. It is particularly effective due to its capability to capture both spatial and frequency information from an image. The Gabor filter is widely recognized for its biological relevance and computational properties, as it closely models the response of neurons in the visual cortex of mammalian brains. This section presents an in-depth exploration of the Gabor filter, including its theoretical foundation, mathematical formulation, and practical applications in image analysis [50].

Theoretical Background:

The Gabor filter is fundamentally a sinusoidal plane wave (a cosine or sine function) modulated by a Gaussian envelope. This combination allows the filter to capture local spatial frequency characteristics while maintaining spatial locality. The response of the Gabor filter is maximum at specific orientations and frequencies in the image, making it highly suitable for edge and texture analysis [50].

Mathematical Formulations:

The Gabor function is defined as:

$$g(x,y;\lambda,\theta,\psi,\sigma,\gamma) = \exp\left(-\frac{x'^2+\gamma^2 y'^2}{2\sigma^2}\right)\cdot\cos\left(2\pi\frac{x'}{\lambda}+\psi\right), \tag{2.24}$$

where,

$$\begin{aligned} x' &= x\cos\theta + y\sin\theta \\ y' &= -x\sin\theta + y\cos\theta \end{aligned} \tag{2.25}$$

λ denotes the wavelength of the cosine factor, θ represents the orientation of the normal to the parallel stripes of a Gabor function, ψ is the phase offset, σ is the standard deviation of the Gaussian envelope, and γ is the spatial aspect ratio.

Parameter Selection:

The parameters of the Gabor filter (λ, θ, ψ, σ, γ) are critical as they determine the filter's sensitivity to specific spatial frequencies and orientations. By varying these parameters, a family of Gabor filters can be constructed to analyze the image at different scales and orientations, making it a versatile tool for image processing tasks [50].

Feature Extraction with Gabor Filters:

In practice, Gabor filters are often applied in a bank, covering various orientations and wavelengths. The output of these filters provides a comprehensive representation of the image's texture and edge information. The convolution of an image with a Gabor filter bank results in a set of filtered images, from which features such as mean and variance can be extracted for texture classification and pattern analysis [50].

In conclusion, Gabor filters are a powerful tool in the field of image processing and analysis, offering a unique blend of spatial and frequency representation. Their ability to model the visual perception mechanisms of the human brain further underscores their significance in computer vision tasks. By carefully tuning the parameters of the Gabor filter, it can be adapted to various applications, providing insightful features for image analysis and contributing to the advancement of pattern recognition and machine learning systems.

Gabor Energy and Gabor Entropy

In the context of image processing and analysis using Gabor filters, two important derived measures often utilized for texture and pattern analysis are Gabor Energy and Gabor Entropy. These measures are extracted from the outputs of the Gabor filter bank and provide significant information about the textural properties of the image. The inclusion of Gabor Energy and Gabor Entropy enhances the feature set derived from the Gabor filter bank, offering more nuanced and discriminative details for various image analysis tasks.

Gabor Energy: Gabor Energy is a measure that encapsulates the sum of squared elements in the filtered image, providing a quantification of the texture's consistency and the spatial frequency components' strength. Mathematically, for a given filtered image I, the Gabor Energy E can be computed as:

$$E = \sum_{x,y} |I(x,y)|^2 \tag{2.26}$$

In the context of a bank of Gabor filters, the energy feature can be calculated for each filter response, leading to a feature vector representing the energy distribution across different orientations and scales.

Gabor Entropy: Gabor Entropy measures the randomness in the distribution of the filter response values, providing an insight into the complexity and variability of the image texture. Entropy is calculated based on the histogram of the filtered image. For a given histogram H with n bins, the Gabor Entropy S is given by:

$$S = -\sum_{i=1}^{n} p(i) \log_2 p(i) \tag{2.27}$$

where $p(i)$ is the probability of the intensity value in the i^{th} bin. Similar to Gabor Energy, the entropy can be computed for each response in the filter bank, providing a comprehensive set of features that describe the textural information present in the image.

Integration in Feature Extraction with Gabor Filters: In the feature extraction phase using Gabor filters, after applying the filter bank and obtaining a set of filtered images, both Gabor Energy and Gabor Entropy can be computed for each

filtered image. The calculated energy and entropy values enrich the feature set derived from the Gabor filter outputs, offering more detailed information about the image's texture. This includes the intensity of texture patterns (captured by energy) and the complexity or randomness of these patterns (captured by entropy). The inclusion of Gabor Energy and Gabor Entropy in the feature set makes it more robust and informative for subsequent image analysis tasks, such as texture classification, pattern recognition, and biometric authentication [50].

In conclusion, Gabor Energy and Gabor Entropy are valuable metrics that complement the spatial and frequency information captured by the Gabor filter bank. These metrics contribute to a more comprehensive and discriminative feature set, enhancing the performance of image analysis and pattern recognition systems. The integration of Gabor Energy and Gabor Entropy with the traditional Gabor filter outputs facilitates a deeper understanding of the textural properties of images, paving the way for advancements in computer vision and image processing applications [50].