Ministère de l'Agriculture, des Ressources
hydrauliques et de la Pêche Maritime
–*–
Institution de la Recherche et de l'Enseignement
Supérieur Agricoles

Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique
–*–
Université de Carthage

REPUBLIQUE TUNISIENNE

MOGRANE HIGHER SCHOOL OF AGRICULTURE

LECTURE NOTES
VERSION.2024.BETA

Machine Learning and Deep Learning

Bilel AMMOURI

: ESAM

: ammouri-bilel

: bilelammouri

: Ammouri-Bilel

: 0000-0002-5491-5172

Preface

These are the class notes I took for ESA Mograne : Introduction to Machine Learning and Deep Learning. These notes are part of a course designed for students in agronomy (rural economics and vegetable production). The course, focused on deep learning, assumes that students have little to no background in machine learning or advanced theoretical mathematics. To make these concepts more accessible, I decided to introduce foundational machine learning concepts and then delve into deep learning, describing some algorithms without getting into detailed mathematical demonstrations. Additionally, I have provided a GitHub repository[1] containing code, notebooks, and examples relevant to agronomy, which will help students better understand and apply these concepts.

1. https://github.com/bilelammouri/Machine-Learning-and-Deep-Learning-in-Agronomy

Table of Contents

CHAPTER 1

Introduction to Machine Learning

1.1 Introduction

1.1.1 1.1 What Is Machine Learning?

Machine learning (ML) involves programming computers to enhance their performance using example data or past experiences. This process entails defining a model with specific parameters and employing a computer program to optimize these parameters with training data or prior experiences. The resulting model can be used for making future predictions (predictive) or extracting insights from data (descriptive), or both. The term 'Machine Learning' was introduced by Arthur Samuel, a pioneer in computer gaming and artificial intelligence at IBM, in 1959. He defined it as *'the field of study that gives computers the ability to learn without being explicitly programmed'*. However, there is no universally accepted definition of machine learning, and interpretations vary among scholars.

A computer program is considered to learn from experience (**E**) regarding some tasks (**T**) and performance measure (**P**) if its performance at tasks T, as measured by P, improves with experience E.

Crop Yield Prediction Problem :
T : Predicting crop yield based on various input factors such as soil quality, weather conditions, and farming practices.
P : Accuracy of predicted yield compared to actual yield.
E : Historical data on crop yields with corresponding input factors.

Pest Detection Problem :
T : Identifying the presence of pests in crop fields using image data.
P : Percentage of correctly identified pest occurrences.
E : A dataset of images of crop fields labeled with pest presence or absence.

Robot Driving Learning Problem :
T : Driving on highways using vision sensors.
P : Average distance traveled before making an error.
E : A sequence of images and steering commands recorded while observing a human driver.

A computer program that learns from experience is called a machine learning program or simply a learning program. It is also sometimes referred to as a learner.

1.1.2 Components of Learning

The learning process, for both humans and machines, can be divided into four main components : data storage, abstraction, generalization, and evaluation. Figure 1.1 illustrates these components and the steps involved in the learning process.
The learning process, whether for humans or machines, can be divided into four main components : data storage, abstraction, generalization, and evaluation. Storing and retrieving large amounts of data is crucial for the learning process, as both humans and computers depend on data storage for advanced reasoning. Humans store data in their brains and retrieve it through electrochemical signals, whereas computers use hard drives, flash memory, and RAM, accessing data via cables and other technologies.

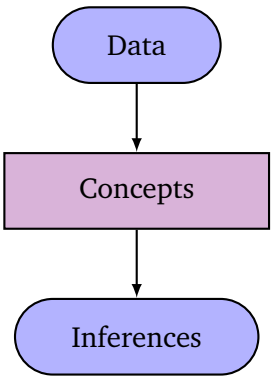


FIGURE 1.1 – Components of learning process

Abstraction involves extracting knowledge from stored data, creating general concepts from the overall data. This process includes applying known models and developing new ones. Training a model to fit a dataset transforms the data into an abstract form that summarizes the original information.

Generalization involves applying the knowledge obtained from stored data to new, similar tasks, aiming to discover the most relevant properties of the data for future applications.

Evaluation provides feedback to measure the usefulness of the learned knowledge. This feedback is used to improve the entire learning process.

1.2 Learning Models

Machine learning involves selecting suitable features to develop models that effectively address specific tasks. These learning models can be categorized into three primary types : Logical models, which utilize logical expressions ; Geometric models, which apply geometric properties of the instance space ; and Probabilistic models, which use probability for classifying instances within the space. Additionally, models may focus on grouping and grading outcomes to enhance predictive accuracy (see Figure 1.2 and Table 1.1).

Geometric models	**Probabilistic models**	**Logical models**
K-nearest neighbors	Naïve Bayes	Decision tree
Linear regression	Gaussian process regression	Random forest
Support vector machine	Conditional random field	...
Logistic regression	...	...

TABLE 1.1 – Examples of different types of learning models

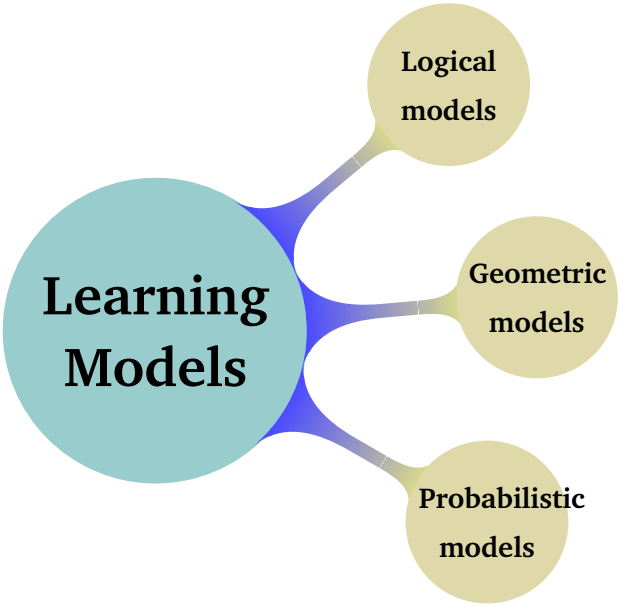


FIGURE 1.2 – Learning Models

1.2.1 Logical Models

Logical models partition the instance space into segments using logical expressions to create grouping models. These expressions yield Boolean values (True or False), facilitating the categorization of data into homogeneous groups based on the specific problem. In classification tasks, all instances within a group belong to the same class. Logical models are primarily divided into two categories : tree models and rule models. Rule models consist of a set of IF-THEN rules, where the 'if-part' defines a segment, and the 'then-part' determines the model's behavior for that segment, following a similar principle in tree-based models. A deeper understanding of logical models requires an exploration of Concept Learning, which involves deriving logical expressions from examples, aligning with the goal of generalizing a function from specific training examples. Concept Learning can be formally defined as the inference of a Boolean-valued function from training examples of its input and output, typically describing only the positive class and labeling everything else as negative. For instance, in a Concept Learning task called "Enjoy Sport," data from various example days is described by six attributes, and the objective is to predict whether a day is enjoyable for sports. This involves formulating hypotheses as conjunctions of constraints on the attributes, with training data containing positive and negative examples of the target function. Each hypothesis might represent a vector of six constraints : Sky, AirTemp, Humidity, Wind, Water, and Forecast, while the training phase focuses on learning the conjunction of attributes for which Enjoy Sport equals "yes." The problem can be articula-

ted as identifying a function that predicts the target variable Enjoy Sport as either yes (1) or no (0) given instances representing all possible days.

1.2.2 Geometric Models

Logical models, such as decision trees, utilize logical expressions to segment the instance space, identifying similarity through logical segments. In contrast, geometric models define similarity based on the geometry of the instance space. Features can be represented as points in two dimensions (x- and y-axis) or three dimensions (x, y, and z). Even if features are not inherently geometric, they can be modeled geometrically ; for example, one could model soil moisture and temperature over time on two axes. There are two primary methods for establishing similarity in geometric models :

— Using geometric concepts like lines or planes to classify the instance space, known as Linear models.

— Using the geometric notion of distance to represent similarity, where proximity implies similar feature values, referred to as Distance-based models.

Linear Models

Linear models are relatively straightforward. They represent the function as a linear combination of inputs. For instance, if x_1 and x_2 are scalars or vectors of the same dimension, and α and β are arbitrary scalars, then $\alpha x_1 + \beta x_2$ represents a linear combination of x_1 and x_2. In its simplest form, $f(x)$ is a straight line, given by the equation $f(x) = \alpha + \beta x$, where α is the intercept and β is the slope (see Figure 1.3).

Distance-based Models

Distance-based models represent the second class of geometric models. Like linear models, distance-based models rely on the geometry of data. As the name suggests, these models operate on the concept of distance. In the context of machine learning, this distance is not solely the physical separation between two points. For example, in agriculture, one might consider the distance between two fields based on the mode of irrigation used—drip irrigation may cover less ground compared to traditional sprinklers. Additionally, the concept of distance can vary depending on the context ; for instance, the distance between crop varieties can influence yield outcomes differently. Commonly used distance metrics include

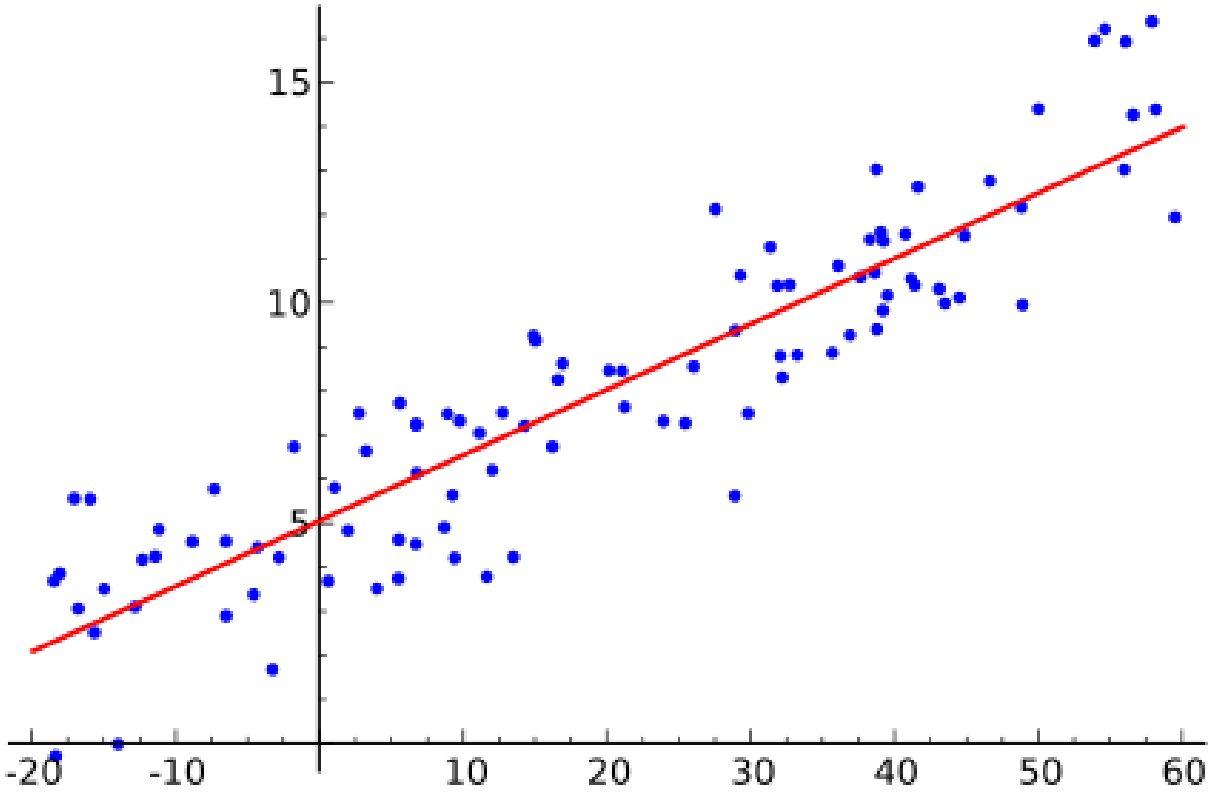


FIGURE 1.3 – Linear Regression

Euclidean, Minkowski, Manhattan, and Mahalanobis (see Figure 1.4). These metrics apply through the concepts of neighbors and exemplars. Neighbors are points in proximity based on the distance measure, expressed through exemplars. Exemplars are either centroids, finding a center of mass according to a chosen distance metric, or medoids, identifying the most centrally located data point. The arithmetic mean is a commonly used centroid, minimizing squared Euclidean distance to all other points.

— A centroid is the geometric center of a plane figure, i.e., the mean position of all points in the figure from the centroid point. This extends to any n-dimensional space object : its centroid is the mean position of all points.

— Medoids are similar to means or centroids but are used when a mean or centroid cannot be defined. They are preferred in contexts where the centroid is not representative of the dataset, such as in image data.

1.2.3 Probabilistic Models

The third category of machine learning algorithms is probabilistic models. Unlike k-nearest neighbor algorithms, which use distance, or logical models, which use logical expressions,

probabilistic models use probability to classify new entities.

Probabilistic models treat features and target variables as random variables. Modeling involves representing and manipulating the uncertainty levels concerning these variables. There are two types of probabilistic models : Predictive and Generative.

1. **Predictive models** use a conditional probability distribution $P(Y|X)$ to predict Y from X.
2. **Generative models** estimate the joint distribution $P(Y, X)$. Once the joint distribution is known, any conditional or marginal distribution involving the same variables can be derived. Generative models can create new data points and their labels based on the joint probability distribution. The joint distribution seeks a relationship between two variables, allowing the inference of new data points once this relationship is known.

Naïve Bayes is an example of a probabilistic classifier, utilizing Bayes' rule, which relies on conditional probability. Conditional probability determines the likelihood of an event given that another event has occurred. The Naïve Bayes algorithm evaluates the evidence to determine the likelihood of a specific class and assigns a label to each entity accordingly.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{1.1}$$

1.2.4 Grouping and Grading

Grouping vs. grading is an orthogonal categorization to geometric-probabilistic-logical-compositional.

— Grouping models divide the instance space into segments or groups and apply a simple method within each segment (e.g., majority class). Examples include decision trees and KNN.

— Grading models form a single global model over the instance space. Examples include linear classifiers and neural networks.

1.3 Learning System

In any learning system, it is essential to understand three key components : T (Task), P (Performance Measure), and E (Training Experience). The learning process can be outlined as follows (see Figure 1.5) : it begins with the task T, performance measure P, and

training experience E, with the ultimate goal of discovering an unknown target function. This target function embodies the knowledge to be acquired from the training experience but remains unknown initially.

For instance, in the context of predicting crop yield, the learning system utilizes historical crop data as its training experience, while the task involves classifying whether a given crop will have a high yield. Here, the training examples can be represented as $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, where X denotes various factors such as soil quality, weather conditions, and irrigation practices, and y signifies the yield status.

The precise knowledge to be gleaned from the training experience pertains to learning the target function, which can be expressed as a mapping function $f : X \rightarrow y$. This function encapsulates the relationship between the input variable X and the output variable y.

1.3.1 Designing a Learning System

Through our exploration of the learning process, we understand that several design choices are fundamental for creating an effective learning system. The critical considerations are :

1. The type of training experience
2. The choice of the target function
3. The way the target function is represented
4. The selection of an approximation algorithm for the target function
5. The overall design

To illustrate these design choices, we will consider the checkers learning problem. The three elements for this scenario are :

To demonstrate these design choices, let's examine a crop yield prediction scenario. The three key elements for this case are :
T : Predicting crop yield
P : The accuracy of the predicted yields compared to the actual yields
E : Historical data on crop yields under varying conditions.

1.3.2 Type of Training Experience

The type of training experience plays a crucial role in determining the success of a learning system. Training experiences can be categorized as follows :

1. **Direct vs. Indirect Training Experience** :
   - **Direct training** provides specific instances along with the correct actions for each instance.
   - **Indirect training** offers sequences of actions and their final outcomes (e.g., yield or no yield) without specifying the correct action for each instance. This introduces the credit assignment problem.
2. **Teacher Involvement** :
   - **Supervised Learning** : The training experience is labeled, meaning each instance is paired with the correct action, requiring guidance.
   - **Unsupervised Learning** : The training experience lacks labels, prompting the learner to make decisions without teacher intervention.
   - **Semi-supervised Learning** : The learner generates instances and consults the teacher only when uncertain about the correct action.
3. **Quality of Training Experience** :
   - The training examples should represent the distribution of instances relevant to the final system's performance measurement. Optimal performance is achieved when the training and test examples share a similar distribution.

In an agricultural context, for example, a crop yield prediction system learns through historical data of crop yields under various conditions, which constitutes indirect training experience. While this experience may not cover conditions commonly found in expert agricultural practices, once an appropriate training experience is established, the next step is to choose the target function.

1.3.3 The choice of the target function

When predicting crop yield, a farmer decides on the best agricultural practices among available options, applying learned experience to enhance their chances of maximizing

yield. This learning process can be formalized through the target function.
Considerations include :

1. **Direct Experience** : The learning system must determine the optimal agricultural practice from a vast search space. This is denoted as the function ChoosePractice : $C \rightarrow P$, where C represents different crop conditions and P denotes possible agricultural practices.
2. **Indirect Experience** : Assigning a score to crop conditions complicates the learning process. We define the function $V : C \rightarrow \mathbb{R}$ to assign scores to crop conditions, with higher scores reflecting more advantageous conditions for high yield.

The target value $V(c)$ for a board state c is defined as follows :

1. If c results in a high yield, then $V(c) = 100$
2. If c results in a low yield, then $V(c) = 0$
3. If c is an average yield, then $V(c) = 50$
4. If c is not a terminal state, then $V(c) = V(c')$, where c' is the best achievable terminal condition from c.

This recursive definition implies that determining $V(c)$ necessitates searching for the optimal agricultural strategy, making it computationally intensive.
The objective of learning is to derive an operational version of V, which the crop yield prediction program can utilize to evaluate crop conditions and select practices efficiently. Perfectly learning this operational form may be challenging ; thus, learning algorithms typically approximate the target function as $\hat{V}$.

1.3.4 The way the target function is represented

With the target function V defined, the subsequent step is to select a suitable representation for the function $\hat{V}$, which the learning algorithm will use. Potential representations include :

— A comprehensive table that stores values for each distinct board state

— A rule-based system that matches board features

— A polynomial function of predefined features

— An artificial neural network.

This selection presents a trade-off : while a more expressive representation can approximate V more accurately, it also requires more training data to differentiate among the various hypotheses.
To simplify, we can represent the function $\hat{V}$ as a linear combination of specific board features :

$$\hat{V} = w_0 + w_1 \cdot x_1(c) + w_2 \cdot x_2(c) + w_3 \cdot x_3(c) + w_4 \cdot x_4(c) + w_5 \cdot x_5(c) + w_6 \cdot x_6(c) \quad (1.2)$$

Here, $w_0, \ldots, w_6$ are numerical coefficients determined by the learning algorithm, and the weights w_1 to w_6 indicate the significance of various board features. These features might include :

— $x_1(c)$ the amount of rainfall during the growing season (mm)

— $x_2(c)$ the average temperature during the growing season (°C)

— $x_3(c)$ the quality of the soil, measured by nutrient content (e.g., pH level)

— $x_4(c)$ the amount of fertilizer used (kg/ha)

— $x_5(c)$ the presence of pests or diseases affecting the crops (binary indicator)

— $x_6(c)$ the type of crop being grown (categorical variable)

1.3.5 The selection of an approximation algorithm for the target function

To train the learning program, a set of training data is necessary, comprising specific board states c and their corresponding training values $V_{\text{train}}(c)$. Each training example is represented as an ordered pair $\langle c, V_{\text{train}}(c)\rangle$.
For instance, a training example might be $\langle(x_1 = 500, x_2 = 22, x_3 = 7.5, x_4 = 100, x_5 = 0, x_6 = 1), 400\rangle$, indicating a win for black (i.e., $x_2 = 22$ degrees Celsius, etc).
While assigning $V_{\text{train}}(c)$ for clear and well-understood conditions is straightforward, it becomes complex for intermediate conditions. In such cases, we utilize temporal difference (TD) learning—a key reinforcement learning concept where iterative corrections improve estimated returns towards accurate targets.
Let Successor(c) denote the subsequent board state following c. The learner's approximation $\hat{V}$ is employed to assign training values for intermediate states as follows :

$$V_{\text{train}}(c) \leftarrow \hat{V}(\text{Successor}(c)) \quad (1.3)$$

Adjusting the Weights

To refine the learning algorithm and optimally fit the training examples, we define the best hypothesis as one that minimizes the squared error E between the training values and the predicted values $\hat{V}$. The algorithm should incrementally adjust weights based on new training examples and remain resilient to errors in the data. The Least Mean Square (LMS) training rule serves as a mechanism to adjust weights minimally in the direction that reduces the error.

The squared error E is calculated as :

$$E = \frac{1}{2}\sum_{i}(V_{\text{train}}(c_i) - \hat{V}(c_i))^2 \tag{1.6}$$

To minimize this error, the LMS rule adjusts the weights as follows :

$$w_j \leftarrow w_j + \eta \cdot (V_{\text{train}}(c) - \hat{V}(c)) \cdot x_j(c) \tag{1.7}$$

where η is the learning rate, and w_j is the weight associated with the feature x_j.

By continuously updating the weights in this manner, the algorithm incrementally improves the accuracy of the yield predictions, ensuring that the model becomes more precise over time in predicting crop yields based on various agricultural conditions.

1.3.6 The overall design

The final design of the checkers learning system can be encapsulated in four distinct program modules that represent fundamental components of many learning systems :

1. **The Performance System** : This module takes a new board state as input and outputs a trace of the game played against itself. It simulates the checkers game using the current hypothesis.

2. **The Critic** : This component processes the trace of the game generated by the performance system. It evaluates the game outcome and produces a set of training examples for the target function. The critic is essential for assessing the effectiveness of moves made during gameplay.
3. **The Generalizer** : This module receives the training examples from the critic and outputs a hypothesis that estimates the target function. Effective generalization is crucial for adapting learned strategies to new, unseen board configurations.
4. **The Experiment Generator** : This component takes the current hypothesis (the function that has been learned so far) as input and generates new problems (initial board states) for the performance system to explore. This iterative process allows the system to continually refine its understanding of the game dynamics.

Together, these modules create a comprehensive framework for the checkers learning system, allowing it to learn from self-play, improve through evaluation, and generalize its knowledge to play effectively against various opponents.

1.4 Types of Learning

Machine learning algorithms can generally be classified into four main types (see Figure 1.6 and 1.7).

1.4.1 Supervised Learning

In supervised learning, a training set containing examples with corresponding correct responses (targets) is provided. The algorithm learns from this training data to generalize and respond accurately to new inputs. This approach is also referred to as learning from exemplars. Specifically, supervised learning involves learning a function that maps inputs to outputs based on example input-output pairs.

Each example in the training dataset consists of an input object (often represented as a vector) and an output value. The supervised learning algorithm analyzes this training data to produce a function capable of mapping new examples. Ideally, this function will accurately determine the class labels for unseen instances. Supervised learning encompasses both classification and regression tasks, and numerous algorithms are available, each with its advantages and disadvantages. There is no single algorithm that excels in all supervised learning scenarios.

The term "supervised learning" arises from the analogy of a teacher overseeing the learning process. The correct answers (outputs) are known, allowing the algorithm to make predictions on the training data and receive corrections from the teacher. Learning continues until the algorithm reaches an acceptable level of performance.

Consider a dataset from an agronomic study that includes various factors affecting crop yield. Each data point in the dataset represents specific agricultural conditions, such as rainfall, temperature, soil quality, fertilizer usage, presence of pests or diseases, and type of crop. Each condition is labeled with the corresponding crop yield.

Here is a detailed example using the previously mentioned variables :

- x_1 — amount of rainfall during the growing season (mm)
- x_2 — average temperature during the growing season (°C)
- x_3 — quality of the soil, measured by nutrient content (e.g., pH level)
- x_4 — amount of fertilizer used (kg/ha)
- x_5 — presence of pests or diseases affecting the crops (binary indicator)
- x_6 — type of crop being grown (categorical variable)

The dataset might look like this :

x_1	x_2	x_3	x_4	x_5	x_6	Yield (units)
500	22	7.5	100	0	Wheat	4000
450	20	6.8	80	1	Corn	3500
...	...	...	...	...	...	...

In this example, the system learns from historical data of crop yields under various conditions, developing a model that can predict future yields based on new sets of conditions. This supervised learning approach helps agronomists and farmers optimize their practices by understanding how different factors contribute to crop productivity.

1.4.2 Unsupervised Learning

This type of machine learning algorithm draws inferences from datasets consisting solely of input data without labeled responses. In unsupervised learning, there are no classifications

or categorizations included in the observations, leading to no output values and, therefore, no function estimations.

Consider data from an agricultural study containing various measurements related to soil properties, crop characteristics, and environmental conditions. Without predefined labels, we aim to discover patterns or groupings in the data that might reveal insights into different types of soil, crop health, or environmental impacts.
Here is a detailed example using the following variables :

- x_1 — soil pH level
- x_2 — soil nitrogen content (mg/kg)
- x_3 — soil phosphorus content (mg/kg)
- x_4 — soil potassium content (mg/kg)
- x_5 — average temperature during the growing season (°C)
- x_6 — average rainfall during the growing season (mm)

The dataset might look like this :

x_1	x_2	x_3	x_4	x_5	x_6
6.5	20	15	50	22	500
5.8	18	10	45	20	450
...	...	...	...	...	...

Using unsupervised machine learning techniques, such as clustering, we can group these data points into clusters that might represent different types of soil, climate conditions, or potential crop yields. For example, we might discover clusters that represent optimal soil conditions for specific crops or identify areas that require soil amendments to improve fertility.

1.4.3 Reinforcement Learning

Reinforcement learning focuses on training an agent to act within an environment to maximize its rewards. Unlike most machine learning approaches, the learner (the program) is not explicitly instructed on which actions to take. Rather, it must discover which actions

yield the highest rewards through experimentation. In many complex scenarios, the effects of actions may not only influence immediate rewards but also impact future situations and subsequent rewards.

Consider an agricultural robot that is tasked with optimizing the watering schedule for a field. While we cannot explicitly instruct the robot on the optimal watering times and amounts, we can provide feedback based on the crop yield and health outcomes. The robot must learn which watering schedules lead to better yields or healthier crops. A similar methodology can be applied to train machines for various tasks, such as pest control, soil treatment, or autonomous harvesting. Reinforcement learning differs from supervised learning, which relies on examples provided by knowledgeable experts.

Here is a detailed description of the reinforcement learning process for the agricultural robot :

- **State** (s) : The current condition of the field, which includes factors such as soil moisture level, weather conditions, and crop growth stage.
- **Action** (a) : The watering decision made by the robot, such as the amount of water to apply and the timing of the application.
- **Reward** (r) : The feedback provided to the robot based on the crop yield and health outcomes. For example, a higher yield or healthier crops result in a higher reward, while poor outcomes result in a lower reward or penalty.
- **Policy** (π) : The strategy that the robot uses to determine its actions based on the current state. The policy is continuously improved as the robot learns from its experiences.

The reinforcement learning process involves the robot interacting with the environment (the field) and receiving feedback to improve its watering policy. The goal is to maximize the cumulative reward over time, leading to optimal watering schedules that enhance crop yield and health.

TABLE 1.2 – Differences between supervised, unsupervised, and reinforcement learning algorithms.

Source : https://datasciencedojo.com/blog/machine-learning-101/

	Supervised Learning	**Unsupervised Learning**	**Reinforcement Learning**
Definition	Makes predictions from data	Segments and groups data	Reward-punishment system and interactive environment
Types of Data	Labeled data	Unlabeled data	Acts according to a policy with a final goal to reach (No or predefined data)
Commercial Value	High commercial and business value	Medium commercial and business value	Little commercial use yet
Types of Problems	Regression and classification	Association and Clustering	Exploitation or Exploration
Supervision	Extra supervision	No	No supervision
Algorithms	Linear Regression, Logistic Regression, SVM, KNN, etc.	K-Means clustering, C-Means, Apriori	Q-Learning, SARSA
Aim	Calculate outcomes	Discover underlying patterns	Learn a series of actions
Application	Risk Evaluation, Forecast Sales	Recommendation System, Anomaly Detection	Self-Driving Cars, Gaming, Healthcare

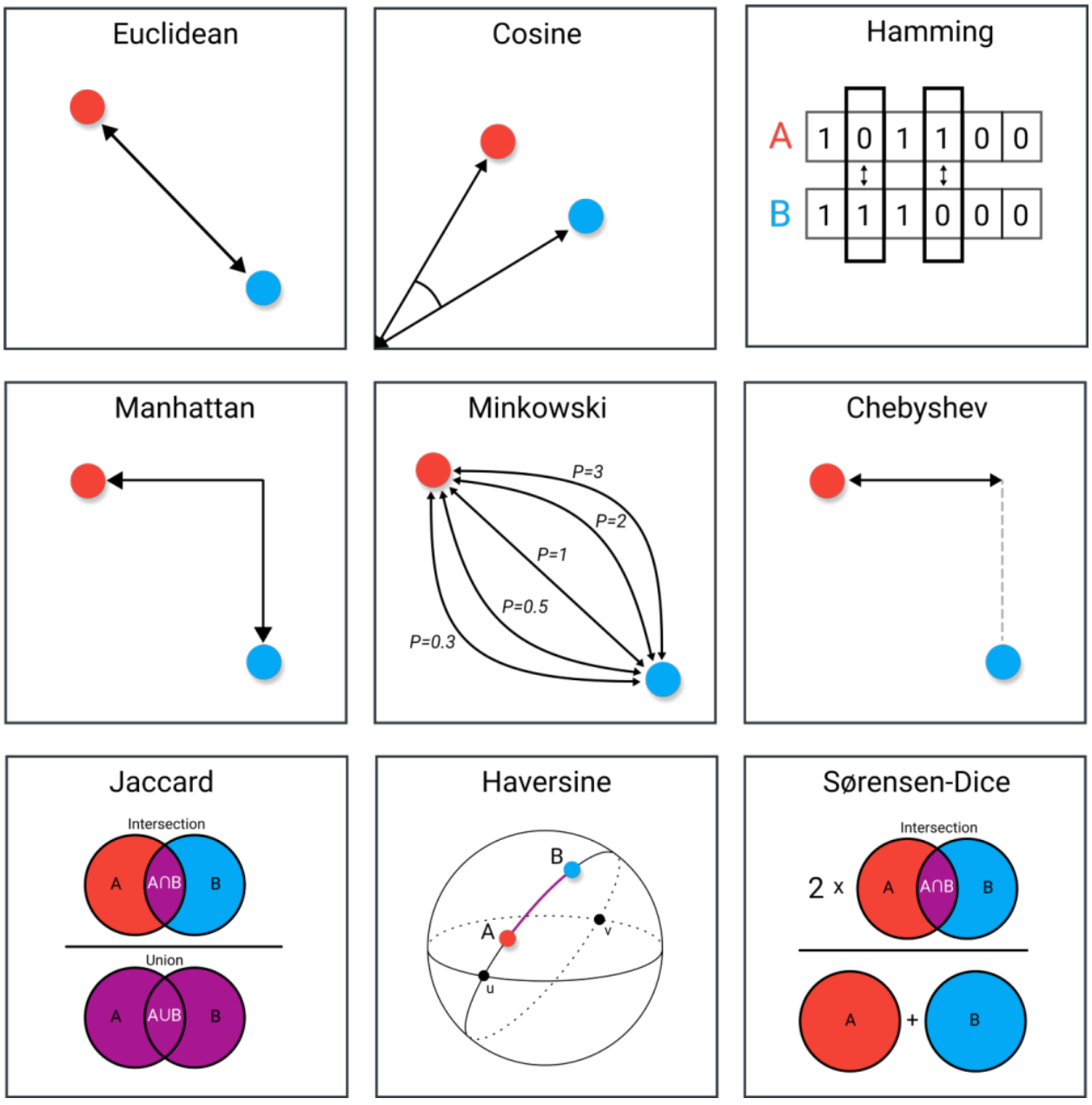


FIGURE 1.4 – Distance metrics.

Source : https://towardsdatascience.com

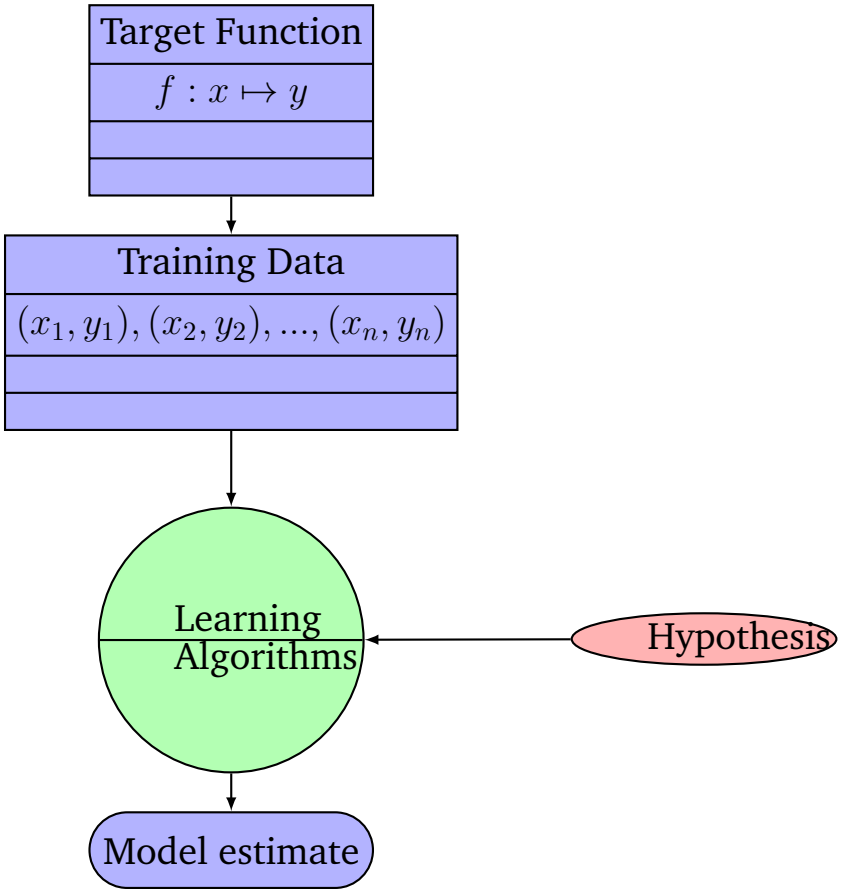


FIGURE 1.5 – Learning System

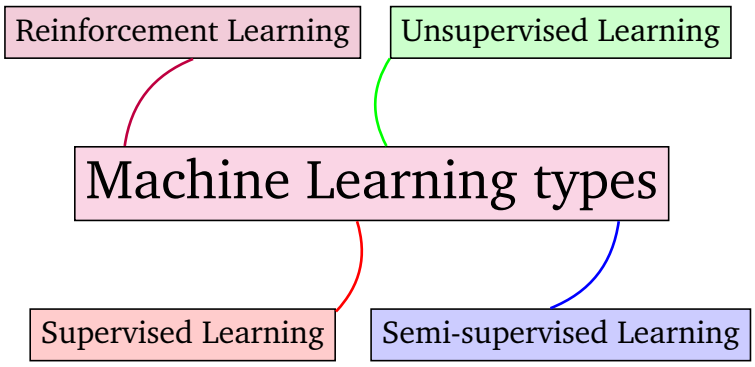


FIGURE 1.6 – Machine Learning types

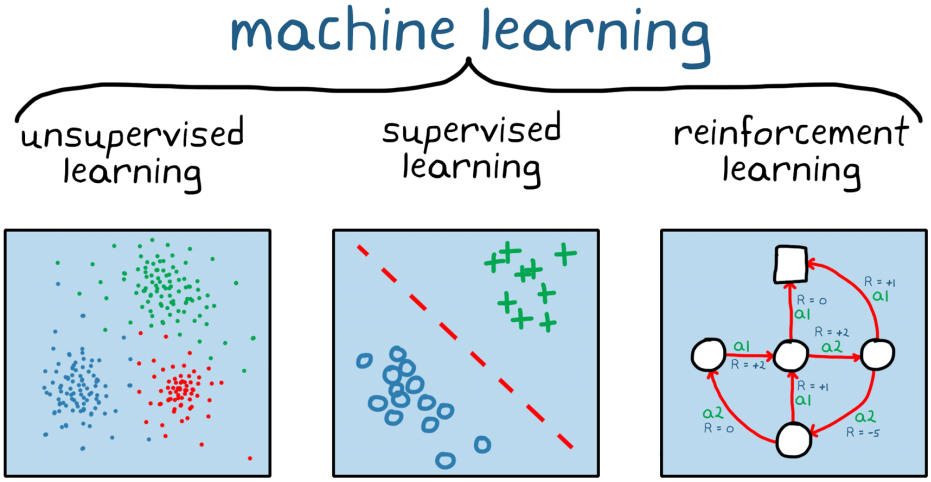


FIGURE 1.7 – Three broad categories of machine learning : unsupervised learning, supervised learning and reinforcement learning

Source : ww2.mathworks.cn

CHAPTER 2

Supervised Learning

Introduction

Supervised learning is a fundamental branch of machine learning where the objective is to learn a mapping from input data to output labels based on a set of labeled training examples. This chapter delves into several key techniques and algorithms that form the backbone of supervised learning. We will start with Decision Trees, exploring algorithms like ID3 and CART (Classification and Regression Trees), which are essential for both classification and regression tasks. The chapter then covers Regression methods, including Linear Regression, Multiple Linear Regression, and Logistic Regression, each offering a unique approach to modeling relationships between variables. Additionally, we will introduce Neural Networks, from the basics of Perceptrons to the complexities of Multilayer Perceptrons, highlighting their powerful ability to learn from data. The discussion extends to Support Vector Machines, examining both Linear and Non-Linear models and the role of Kernel Functions in enhancing model performance. Finally, we will explore K Nearest Neighbors, a simple yet effective method for classification and regression tasks. Through these topics, this chapter aims to provide a comprehensive understanding of supervised learning, equipping readers with the knowledge and skills to apply these techniques to real-world problems.

2.1 Understanding Supervised Learning

Supervised learning is a machine learning approach where the algorithm iteratively learns the dependencies between data points. The desired output is specified in advance, and the learning process is supervised by comparing the algorithm's predictions to actual results. The goal is to optimize the algorithm so it can apply learned patterns to make predictions on new, unseen data. Supervised learning methods can be used for both regression and classification problems (see Figure 2.1).

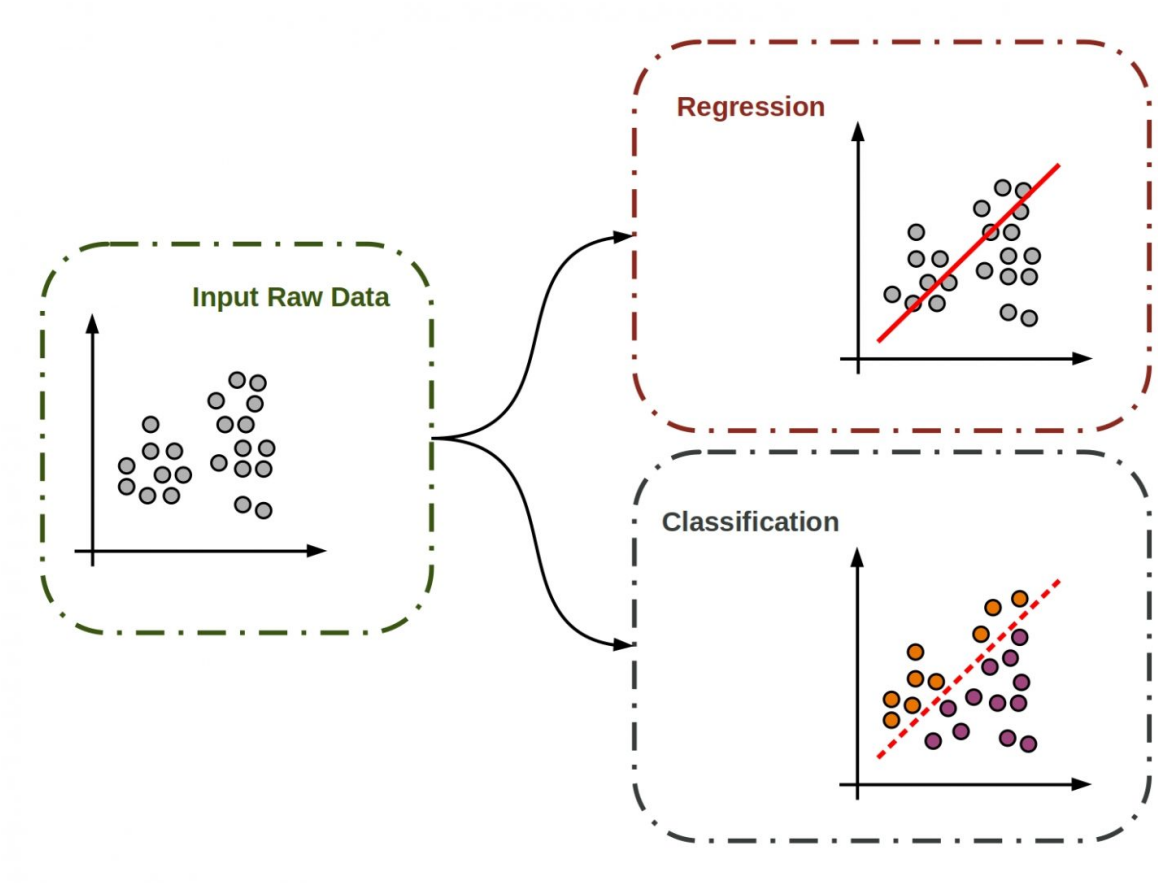


FIGURE 2.1 – Supervised Learning.

Source : starship-knowledge.com

In supervised classification, abstract classes are created to categorize and organize data meaningfully. Objects are grouped based on similar characteristics and structured accordingly. This approach helps in organizing data for better interpretation and analysis.

In contrast, supervised regression algorithms are used to make predictions and infer causal relationships between independent and dependent variables. These algorithms are essential for predictive modeling and understanding the influence of different factors on the outcomes.

The following figure 2.2 lists some of the most important supervised classification algorithms .

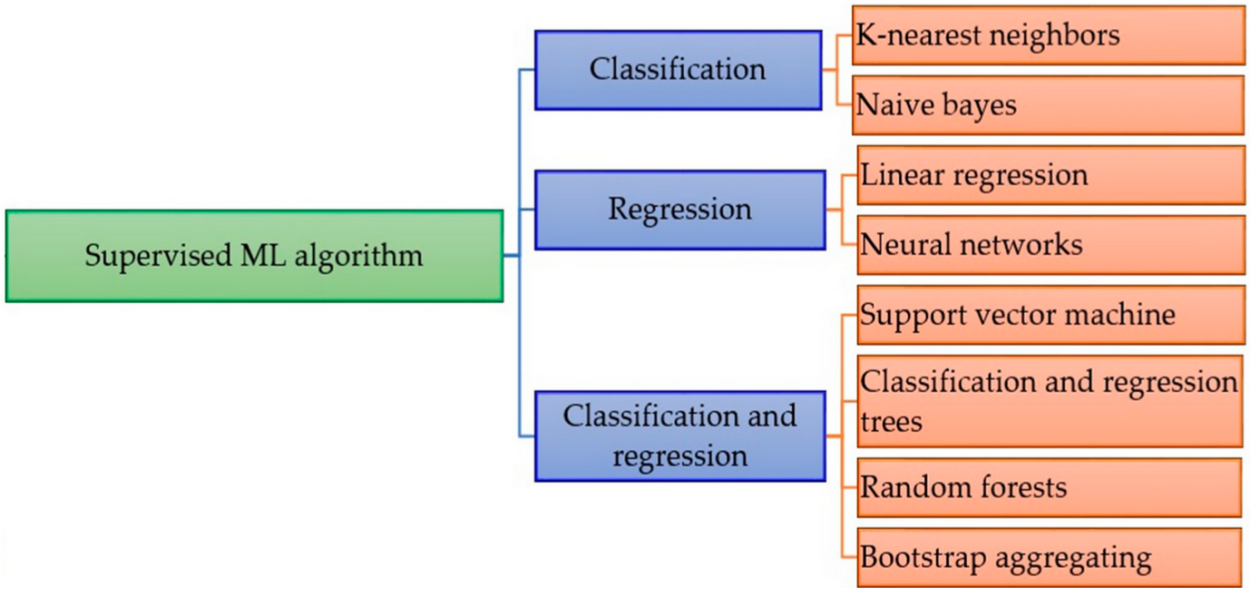


FIGURE 2.2 – Summarized taxonomy of supervised ML algorithms.
Source : www.mdpi.com

2.2 Regression Algorithms

Regression analysis is a statistical approach utilized to model the relationship between a dependent (target) variable and one or more independent (predictor) variables. This method helps in understanding how the dependent variable changes in response to variations in independent variables, while keeping other variables constant. It is primarily used to predict continuous values, such as temperature, crop yield, age, salary, or price.

Regression is a supervised learning technique that identifies correlations between variables and facilitates predictions of continuous output variables based on one or more predictors. It is widely used for tasks such as forecasting, time series analysis, and exploring causal relationships between variables.

Graphically, regression can be represented as a line or curve that best fits the observed data points related to agricultural outcomes. The objective is to minimize the vertical distances between the actual data points and the regression line, which indicates the strength of

the relationship captured by the model. In agronomy, examples of regression applications include predicting crop yields based on factors like fertilizer application and irrigation levels, analyzing the impact of soil quality on plant growth, and forecasting pest outbreaks linked to environmental conditions.

These regression models help agronomists make informed decisions and optimize agricultural practices for better outcomes.

Regression analysis is crucial for predicting continuous variables across various real-world scenarios, including weather forecasting, sales predictions, and market analysis. It employs statistical techniques to enhance prediction accuracy. Other benefits of regression analysis include :

— Estimating relationships between target and independent variables.

— Identifying trends within data.

— Predicting real-valued outcomes.

— Determining the relative importance of different factors and how they interact.

To illustrate regression analysis in the field of agronomy, consider the following example :

A farming cooperative, GreenGrow, has been documenting various factors that influence their crop yields over the past five years. These factors include the amount of fertilizer used, irrigation levels, and weather conditions. For the year 2024, GreenGrow plans to apply 150 kg of fertilizer per hectare and seeks to predict the resulting crop yield. Such prediction tasks are commonly addressed through regression analysis.

Regression analysis allows GreenGrow to create a model that captures the relationship between fertilizer usage and crop yield based on historical data. By inputting the planned fertilizer amount into the model, they can obtain an estimate of the expected yield, helping them make informed decisions about resource allocation and management practices.

There are several types of regression methods employed in data science and machine learning, each suitable for different contexts. Common types include :

— Linear Regression

— Logistic Regression

— Polynomial Regression

— Ridge Regression

— Lasso Regression

— Neural Networks

2.2.1 Linear Regression

Linear regression is a straightforward statistical method used for predictive analysis. It establishes the relationship between continuous variables, making it one of the simplest algorithms for regression tasks. This technique illustrates the linear correlation between the independent variable (X-axis) and the dependent variable (Y-axis) (see Figure 1.3).

When there is a single independent variable, it is termed simple linear regression (2.1) ; when there are multiple independent variables, it is referred to as multiple linear regression (2.2).

$$y_i = \omega_0 + \omega_1 \times x_i + \varepsilon_i \tag{2.1}$$

$$y_i = \omega_0 + \omega_1 \times x_i^1 + \cdots + \omega_p \times x_i^p + \varepsilon_i \tag{2.2}$$

Where Y represents the dependent variable (target), X denotes the independent variable (predictor), and ω_i are the linear coefficients and ε_i are the random error term.

Key assumptions of linear regression include :

- A linear relationship should exist between the target and predictor variables.
- The dependent or target variable (Y) in MLR must be continuous, while the predictor or independent variables may be either continuous or categorical.
- Minimal or no multicollinearity among independent variables.
- The residuals from the regression must be normally distributed.
- Homoscedasticity, indicating consistent error variance across values of independent variables.
- The error terms should be normally distributed, which can be assessed using a Q-Q plot.
- Absence of autocorrelation in error terms, as this could significantly diminish model accuracy.

The primary objective in linear regression is to identify the best fit line, which minimizes the error between predicted and actual values. The accuracy of this line depends on optimizing the coefficients (ω_0 and ω_1), which is achieved through a cost function.
The cost function measures how well the linear regression model performs and helps in estimating the coefficients for the best fit line. For linear regression, the Mean Squared Error (MSE) is commonly used, defined as :

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - (\omega_1 x_i + \omega_0))^2 \tag{2.3}$$

Where :

- N : Total number of observations
- y_i : Actual value
- $(\omega_1 x_i + \omega_0)$: Predicted value

Residuals represent the discrepancies between actual and predicted values; larger residuals indicate a poor fit, while smaller residuals suggest a better model.

2.2.2 Lasso Regression

Lasso regression, short for Least Absolute Shrinkage and Selection Operator, is a linear model that includes a regularization term to prevent overfitting. This technique not only helps in reducing model complexity but also performs feature selection by shrinking some coefficients to zero. The addition of the L1 penalty term differentiates it from standard linear regression (see Figure 2.3).

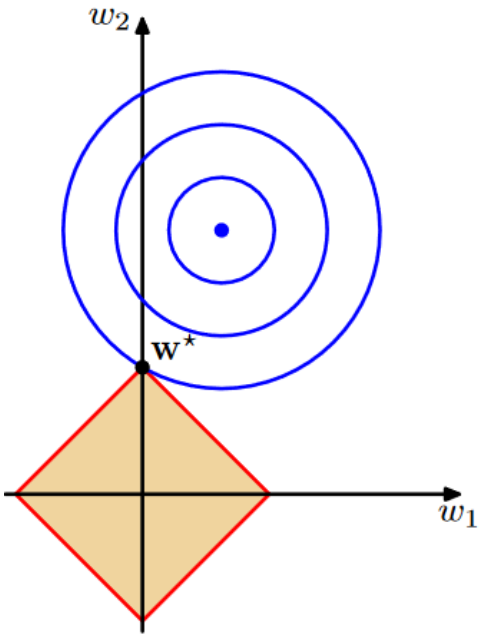


FIGURE 2.3 – Lasso Regression (bishop2006pattern)

Lasso regression modifies the standard linear regression formula by adding a regularization term $\lambda \sum_{j=1}^{p} |\omega_j|$, where λ controls the strength of the penalty (2.4). This helps in addressing issues of multicollinearity and overfitting.

$$\hat{\omega} = \arg\min_{\omega} \left\{ \sum_{i=1}^{n} (y_i - \omega_0 - \sum_{j=1}^{p} \omega_j x_{ij})^2 + \lambda \sum_{j=1}^{p} |\omega_j| \right\} \tag{2.4}$$

Where y represents the dependent variable (target), x denotes the independent variables (predictors), ω are the coefficients, and λ is the regularization parameter. A larger λ leads to more shrinkage of the coefficients.

2.2.3 Ridge Regression

Ridge regression, also known as Tikhonov regularization, is a linear model that includes a regularization term to prevent overfitting. This technique helps in reducing model complexity and multicollinearity by adding a penalty term that shrinks the coefficients. The addition of the L2 penalty term differentiates it from standard linear regression (see Figure 2.4).

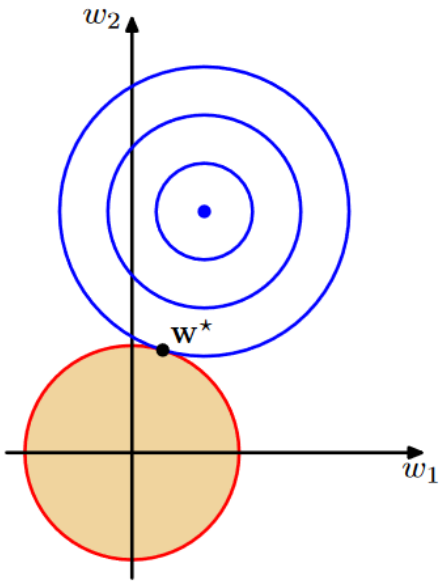


FIGURE 2.4 – Ridge Regression (bishop2006pattern)

Ridge regression modifies the standard linear regression formula by adding a regularization term $\lambda \sum_{j=1}^{p} \omega_j^2$, where λ controls the strength of the penalty (2.5). This helps in addressing issues of multicollinearity and overfitting.

$$\hat{\omega} = \arg\min_{\omega} \left\{ \sum_{i=1}^{n} (y_i - \omega_0 - \sum_{j=1}^{p} \omega_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \omega_j^2 \right\} \tag{2.5}$$

Where y represents the dependent variable (target), x denotes the independent variables (predictors), ω are the coefficients, and λ is the regularization parameter. A larger λ leads to more shrinkage of the coefficients.

2.2.4 Polynomial Regression

Polynomial regression is an extension of linear regression that models the relationship between the independent variable x and the dependent variable y as an n-th degree polynomial. This technique can capture non-linear relationships between variables, making it more flexible than simple linear regression (see Figure 2.5).

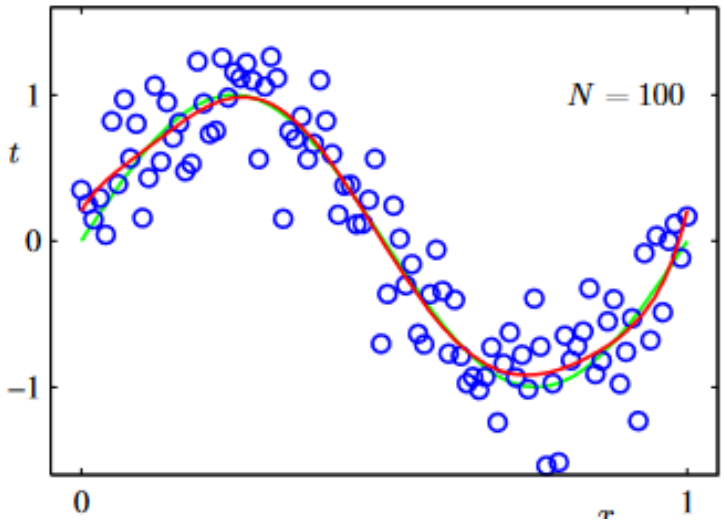


FIGURE 2.5 – Polynomial Regression (bishop2006pattern)

Polynomial regression expands the linear regression model by adding polynomial terms of the independent variable x (2.6). This allows the model to fit curves instead of straight lines, accommodating more complex data patterns.

$$y_i = \omega_0 + \omega_1 x_i + \omega_2 x_i^2 + \cdots + \omega_n x_i^n + \varepsilon_i \tag{2.6}$$

Where y represents the dependent variable (target), x denotes the independent variable (predictor), ω_i are the coefficients, and ε_i are the error terms. By increasing the degree n, the model becomes more flexible but may also risk overfitting.

2.3 Classification Algorithms

Classification is a fundamental technique in machine learning and statistics used to categorize data into predefined classes or categories. Unlike regression, which predicts continuous values, classification algorithms are used to predict discrete outcomes. These algorithms are essential for a wide range of applications, such as spam detection, medical diagnosis, credit scoring, and more.

Classification is a supervised learning technique that assigns labels to data points based on input features. It is widely used for tasks such as image recognition, document categorization, and customer segmentation. The primary goal of classification is to learn a mapping from input features to a set of discrete labels.

Graphically, classification can be represented by decision boundaries that separate data points of different classes in the feature space. These boundaries can be linear or non-linear, depending on the algorithm and the nature of the data. In agronomy, examples of classification applications include distinguishing between different crop types, identifying healthy and diseased plants, and categorizing soil types based on various characteristics. These classification models assist agronomists in making informed decisions, enhancing agricultural productivity, and managing resources effectively.
Classification algorithms are crucial for predicting categorical outcomes in various real-world scenarios, including fraud detection, sentiment analysis, and biometric identification. They employ statistical and computational techniques to maximize classification accuracy. Other benefits of classification analysis include :

— Categorizing data into meaningful classes.

— Identifying patterns and relationships within data.

— Enhancing decision-making processes by providing clear, actionable insights.

— Reducing human error in tasks requiring categorization.

To illustrate classification in the field of agronomy, consider the following example : A research team at AgroTech Institute is developing a system to identify various diseases in crops based on leaf images. By collecting a large dataset of leaf images labeled with the corresponding diseases, the team aims to train a model that can accurately classify new images into specific disease categories. Using classification algorithms, the team can build a model that learns the distinguishing features of each disease from the training data. Once trained, the model can predict the disease class of a new leaf image, helping farmers quickly and accurately diagnose and manage crop health issues.

There are several types of classification methods employed in data science and machine learning, each suitable for different contexts. Common types include :

— Logistic Regression

— k-Nearest Neighbors (k-NN)

— Naive Bayes

2.3.1 Logistic Regression

Logistic regression is a statistical method used for binary classification tasks, where the goal is to predict one of two possible outcomes. It models the probability that a given input belongs to a particular class, making it an essential tool for various classification problems.

Unlike linear regression, which predicts continuous values, logistic regression predicts the probability of an outcome that can only take on two discrete values (e.g., yes/no, true/-false, success/failure). The model uses a logistic function (sigmoid function) to map predicted values to probabilities (see Figure 2.6).

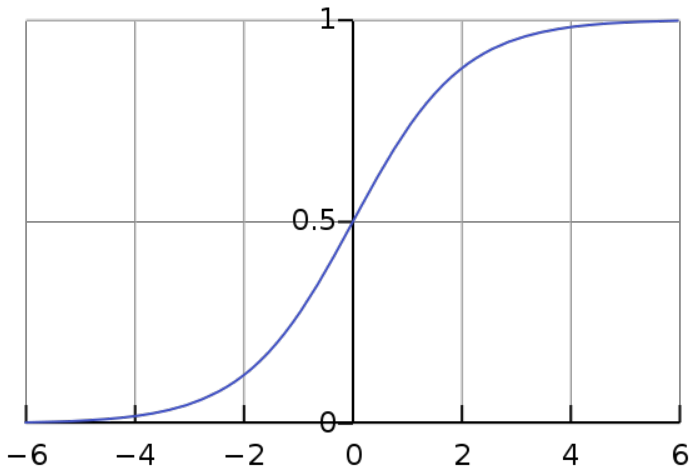


FIGURE 2.6 – Logistic Regression

Logistic regression can be extended to multi-class classification problems through techniques such as one-vs-rest (OvR) and multinomial logistic regression. However, its primary application is in binary classification tasks.

In logistic regression, the relationship between the independent variables (predictors) and the dependent variable (binary target) is modeled using the logistic function :

$$P(Y = 1|X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n)}} \tag{2.7}$$

$$Logit(P) = \ln \frac{p}{(1-p)} = \beta_0 + \beta_1 X_1 + \cdots + \beta_n X_n \tag{2.8}$$

Where :

— $P(Y = 1|X)$ is the probability that the target variable Y equals 1 given the predictors X.

— β_0 is the intercept.

— $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients of the predictor variables $X_1, X_2, \ldots, X_n$.

— e is the base of the natural logarithm.

The logistic function outputs a value between 0 and 1, which can be interpreted as the probability of the target variable being 1.

Key assumptions of logistic regression include :

— The dependent variable is binary.

— Observations are independent of each other.

— There is little or no multicollinearity among the independent variables.

— The relationship between the independent variables and the log odds of the dependent variable is linear.

— Large sample size is required for logistic regression to provide reliable results.

Logistic regression is widely used in various fields, including healthcare for disease prediction, finance for credit scoring, marketing for customer segmentation, and many more, due to its simplicity, interpretability, and effectiveness in binary classification tasks.

2.3.2 k-Nearest Neighbors (k-NN)

k-Nearest Neighbors (k-NN) is a simple, non-parametric classification algorithm used for both classification and regression tasks. In classification, k-NN predicts the class of a data point based on the majority class among its k-nearest neighbors in the feature space.

The k-NN algorithm is particularly intuitive and easy to implement. It does not assume any underlying distribution for the data, making it a versatile method for various classification problems. The choice of k (the number of neighbors) is crucial, as it influences the algorithm's performance. A small k can make the model sensitive to noise, while a large k can smooth out class boundaries excessively.

The k-NN algorithm operates on the principle of similarity, often measured using distance metrics such as Euclidean distance, Manhattan distance, or Minkowski distance. The Euclidean distance between two points $x = (x_1, x_2, \ldots, x_n)$ and $y = (y_1, y_2, \ldots, y_n)$ is given by :

$$d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{2.9}$$

The algorithm can be summarized as follows :

1. Compute the distance between the query instance and all the training samples.
2. Select the k-nearest neighbors to the query instance.
3. Assign the most common class among the k-nearest neighbors to the query instance.

Key considerations for k-NN include :

- Choosing the value of k : The optimal k can be determined using cross-validation.
- Distance metric : The choice of distance metric affects the performance of the algorithm.
- Feature scaling : Features should be normalized or standardized to ensure that each feature contributes equally to the distance calculation.
- Computational efficiency : k-NN can be computationally intensive for large datasets, as it requires calculating the distance to all training samples for each prediction.

Despite its simplicity, k-NN is a powerful tool for classification tasks, especially when the decision boundary is complex and non-linear. It is widely used in fields such as pattern recognition, image classification, and bioinformatics, where it can effectively leverage the similarity between instances to make accurate predictions.

To illustrate k-NN in the field of agronomy, consider the following example :

A team at AgroTech Institute wants to classify different types of crops based on features such as leaf size, plant height, and flowering time. By collecting data on these features from various crop samples, they can use k-NN to classify new samples based on their similarity to known crop types (see Figure 2.7).

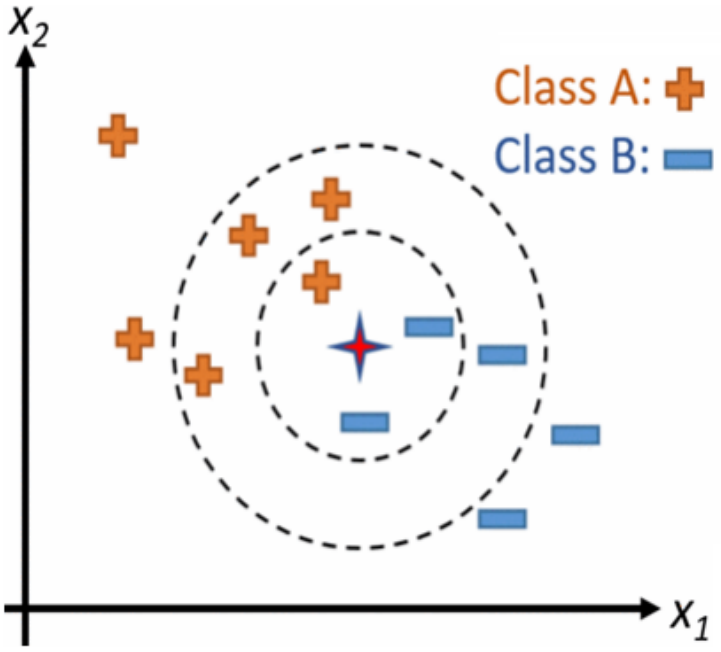


FIGURE 2.7 – A simple KNN model for different values of k

Source : alsharif2020machine

2.4 Regression-classification algorithms

Regression-classification algorithms are versatile models that can be used for both regression (predicting continuous values) and classification (predicting discrete classes) tasks. These algorithms are particularly powerful as they can handle a variety of data types and structures. In this section, we will explore some of the most commonly used regression-classification algorithms, including Decision Trees (DT), Random Forest (RF), Support Vector Machines (SVM), Boosting, and Bagging.

2.4.1 Decision Trees (DT)

Decision Trees are versatile and interpretable models that can be used for both regression and classification tasks. They split the data into subsets based on the value of input features, creating a tree-like structure where each internal node represents a feature, each branch represents a decision rule, and each leaf node represents an outcome.
The diagram below 2.8 illustrates the key terminologies associated with decision trees.

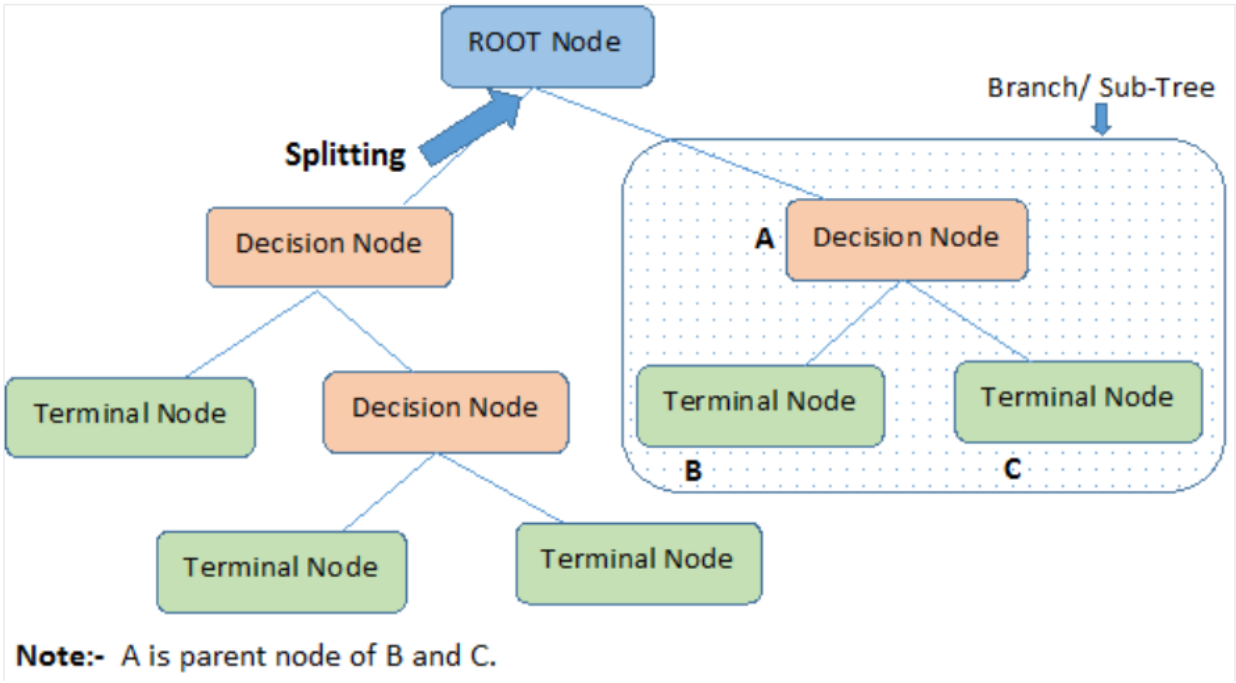


FIGURE 2.8 – Decision Tree Structure

Source : www.datacamp.com

A decision tree begins with a root node representing the entire population or sample, which is then split into two or more homogeneous groups through a process called splitting. Sub-nodes that further divide are known as decision nodes, while those that do not are called

terminal nodes or leaves. A segment of a full tree is referred to as a branch.
Various algorithms exist for constructing Decision Trees :

— **ID3 Algorithm (Iterative Dichotomiser 3) :** ID3 is one of the foundational algorithms for constructing decision trees. It recursively partitions the dataset based on attributes to maximize information gain.

— **C4.5 Algorithm :** An extension of ID3, C4.5 introduces capabilities for handling continuous attributes and missing values, making it more robust in real-world applications.

— **C5.0 Algorithm :** This algorithm is an improvement over C4.5, incorporating boosting techniques and optimizations for better performance and accuracy.

— **CART (Classification and Regression Trees) :** CART is a versatile algorithm that can construct both classification and regression trees. It uses binary splits and measures like Gini index or Mean Squared Error to create robust decision trees.

We have established that decision trees can be used for both regression and classification tasks. Let's delve into the algorithms behind the various types of decision trees.

Classification with Decision Trees

In classification tasks, Decision Trees aim to predict the class label of an instance based on its features (see Figure 2.9). The algorithm recursively splits the dataset into subsets that are as homogeneous as possible concerning the target class. The most common measures for selecting the best split are Gini impurity and information gain.
The classification process can be summarized by the following equation for a leaf node :

$$\hat{y} = \arg\max_{c \in C} P(y = c|X) \tag{2.10}$$

where $\hat{y}$ is the predicted class, C is the set of all classes, and $P(y = c|X)$ is the probability of class c given the features X.

Regression with Decision Trees

In regression tasks, Decision Trees predict continuous values. The tree structure is built in a similar way, but instead of predicting class labels, the algorithm predicts a numerical value at each leaf node (see Figure 2.10). The predicted value for a new instance is the average of the target values in the corresponding leaf node.

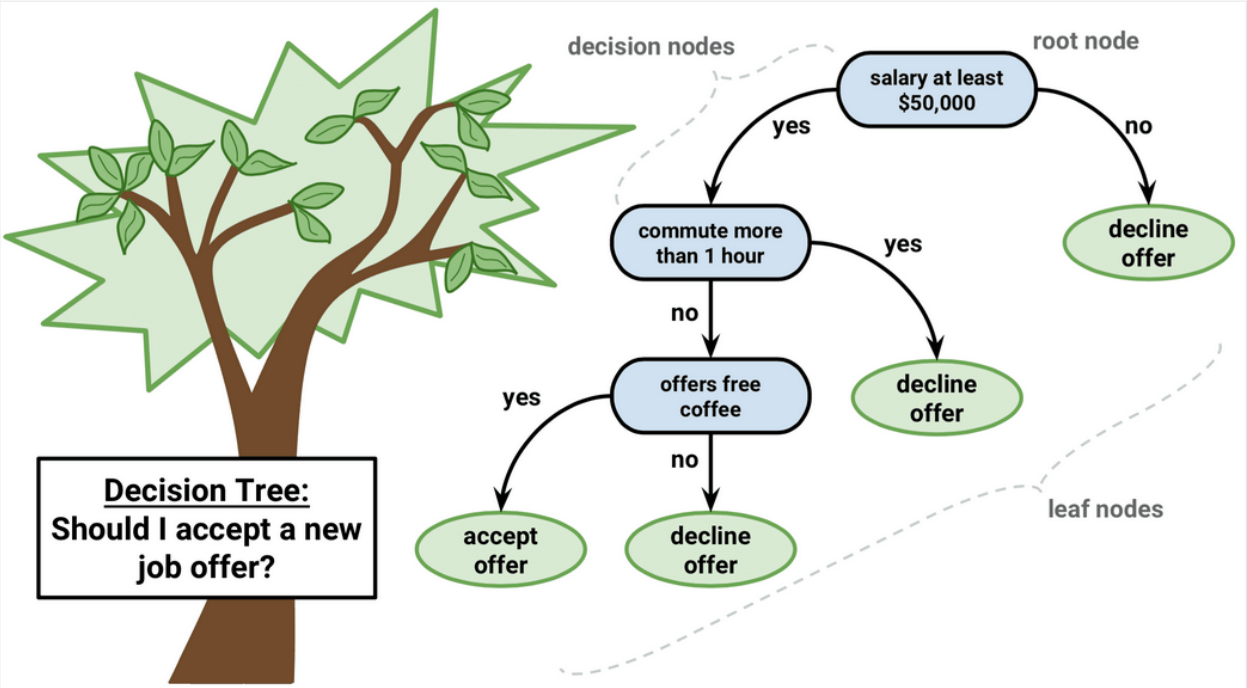


FIGURE 2.9 – Classification Decision Tree Structure

Source : www.datacamp.com

The regression prediction can be expressed as :

$$\hat{y} = \frac{1}{N_t} \sum_{i \in T} y_i \tag{2.11}$$

where $\hat{y}$ is the predicted value, N_t is the number of training instances in the leaf node T, and y_i are the target values of the instances in T.

In agronomy, Decision Trees can be utilized to predict crop yields based on inputs such as fertilizer application, irrigation levels, and climatic conditions.

Decision Trees are easy to interpret and visualize, making them a popular choice for exploratory data analysis and decision support systems. Their ability to provide clear decision rules contributes to their widespread use in various fields, including agronomy.

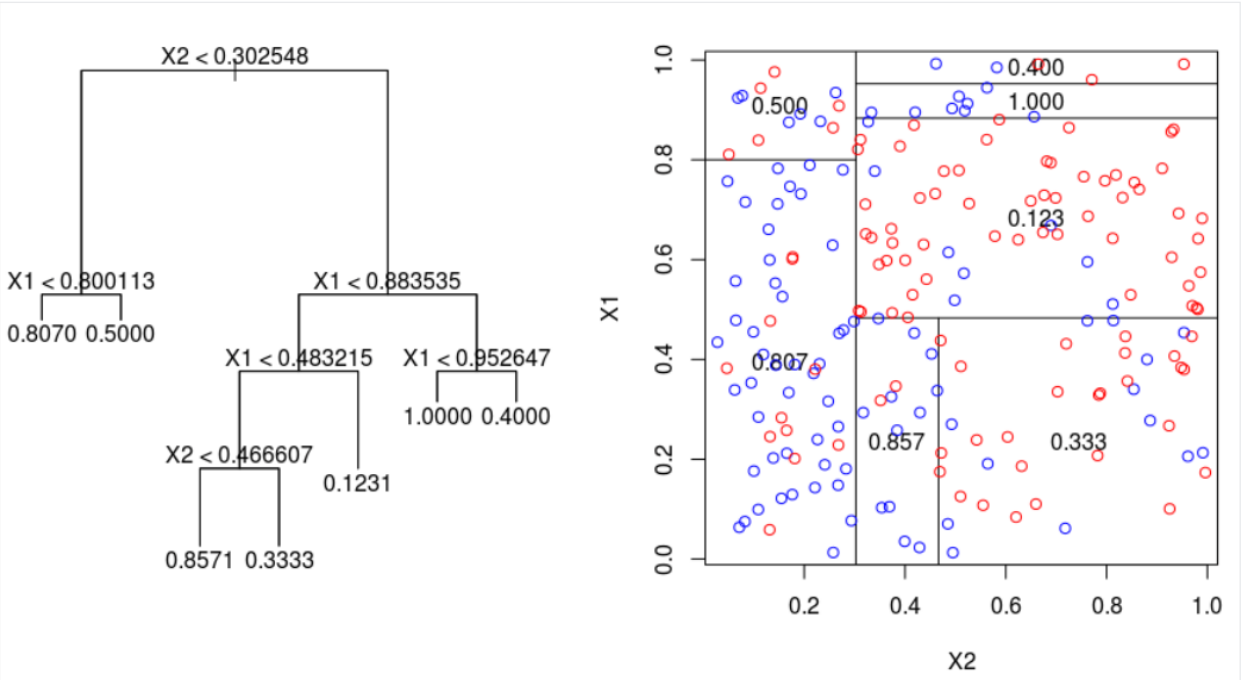


FIGURE 2.10 – Regression Decision Tree Structure

Source : www.datacamp.com

ID3 Algorithm

ID3 Algorithm performs the following tasks recursively :

1. Create a root node for the tree.
2. If all examples are positive, return a leaf node labeled "positive."
3. If all examples are negative, return a leaf node labeled "negative."
4. Calculate the entropy of the current state.
5. For each attribute x, calculate the entropy with respect to the attribute.
6. Select the attribute that maximizes the information gain.
7. Remove the attribute that offers the highest information gain from the set of attributes.
8. Repeat until all attributes are exhausted or the decision tree consists entirely of leaf nodes.

2.4.2 Random Forest (RF)

Random Forest is an ensemble learning method that combines the predictions of multiple decision trees to improve robustness and accuracy (see Figure 2.11). It can be used for both

classification and regression tasks. The algorithm creates a "forest" of decision trees, each trained on a different random subset of the data, and then aggregates their predictions.

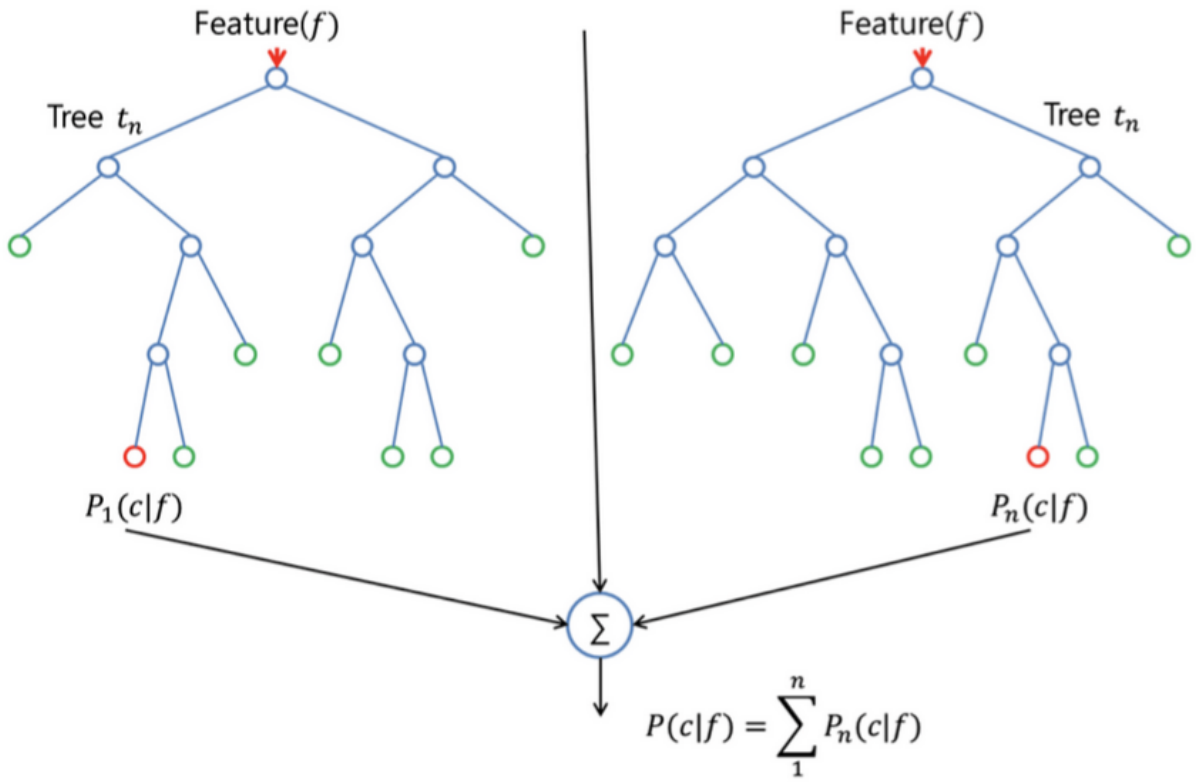


FIGURE 2.11 – Random Forest Structure

Source : www.datacamp.com

Classification with Random Forest

In classification tasks, Random Forest builds multiple decision trees and merges their results to obtain a more accurate and stable prediction. Each tree in the forest outputs a class prediction, and the class with the most votes becomes the final prediction.

The classification prediction for Random Forest can be described by the following equation :

$$\hat{y} = \arg\max_{c \in C} \sum_{b=1}^{B} I(h_b(X) = c) \tag{2.12}$$

where $\hat{y}$ is the predicted class, C is the set of all classes, B is the number of trees in the forest, $h_b(X)$ is the prediction of the b-th tree, and I is an indicator function that equals 1 if $h_b(X) = c$ and 0 otherwise.

In agronomy, Random Forest can be used to classify different types of crops based on features such as soil properties, weather patterns, and plant characteristics.

Regression with Random Forest

In regression tasks, Random Forest predicts continuous values by averaging the predictions of individual trees. Each tree provides a numerical prediction, and the final output is the mean of all the tree predictions.

The regression prediction for Random Forest is given by :

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} h_b(X) \tag{2.13}$$

where $\hat{y}$ is the predicted value, B is the number of trees, and $h_b(X)$ is the prediction of the b-th tree.

In agronomy, Random Forest can be used to predict crop yields based on factors such as fertilizer use, irrigation levels, and weather conditions.

Random Forest is robust to overfitting, especially when dealing with large datasets with many features. It provides an estimate of feature importance, which is valuable for understanding the key factors influencing predictions.

2.4.3 Support Vector Machines (SVM)

Support Vector Machines (SVM) are powerful supervised learning models used for both classification and regression tasks (see Figure 2.12). SVMs work by finding the hyperplane that best separates the data into classes or fits the data in the case of regression. They are particularly effective in high-dimensional spaces and are known for their robustness and accuracy.

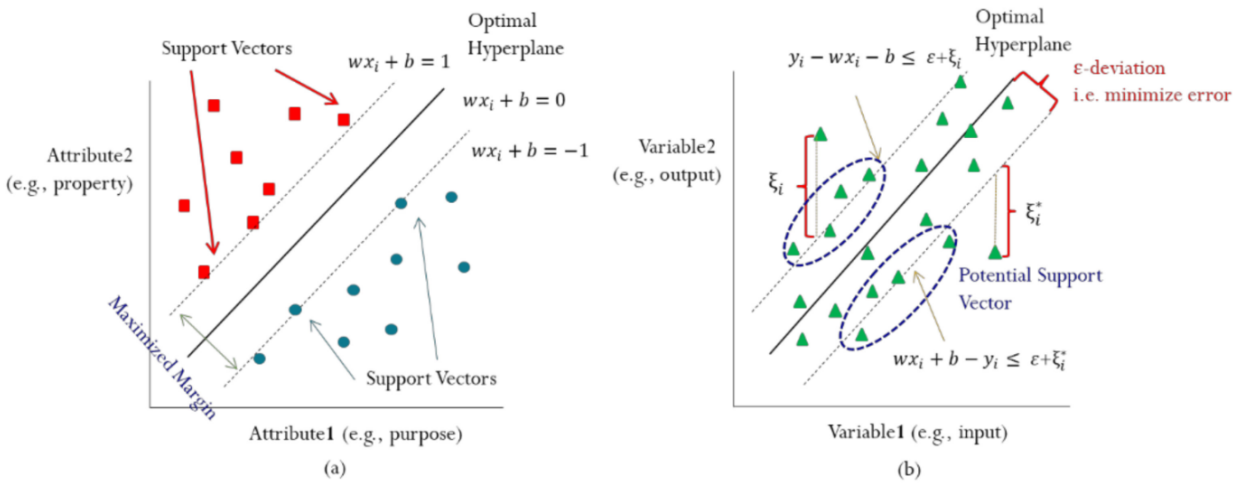


FIGURE 2.12 – Overview of SVM algorithm : (a) SVM for classification ; (b) SVM for regression

Source : liang2020machine

Classification with Support Vector Machines

In classification tasks, SVM aims to find the optimal hyperplane that maximizes the margin between the different classes. The support vectors are the data points that are closest to the hyperplane and influence its position and orientation. The most common kernel functions used in SVM include linear, polynomial, and radial basis function (RBF).

The classification decision function for SVM can be described by the following equation :

$$f(X) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) \tag{2.14}$$

where $\mathbf{w}$ is the weight vector, $\mathbf{x}$ is the input feature vector, and b is the bias term.

> In agronomy, SVM can be used to classify different crop types, detect plant diseases, and predict soil properties based on spectral data.

Regression with Support Vector Machines (SVR)

In regression tasks, SVM, also known as Support Vector Regression (SVR), tries to fit the best line within a threshold value, known as the epsilon (ε) margin. The objective is to ensure that most of the data points lie within this margin.

The regression function for SVR is given by :

$$f(X) = \mathbf{w} \cdot \mathbf{x} + b \tag{2.15}$$

where $\mathbf{w}$ is the weight vector, $\mathbf{x}$ is the input feature vector, and b is the bias term. The goal is to minimize the following loss function :

$$L(y, f(X)) = \max(0, |y - f(X)| - \varepsilon) \tag{2.16}$$

where y is the actual value and ε is the margin of tolerance.

In agronomy, SVR can be used to predict continuous outcomes such as crop yields based on inputs like fertilizer use, irrigation levels, and weather conditions.

SVMs are effective in high-dimensional spaces and are particularly useful when the number of dimensions exceeds the number of samples. They can handle both linear and non-linear relationships through the use of different kernel functions.

2.4.4 Ensemble learning

Ensemble learning enhances machine learning results by combining multiple models, leading to better predictive performance than a single model. The basic concept involves training a set of classifiers (experts) and allowing them to vote. Two key types of ensemble learning are Bagging and Boosting. Both techniques reduce the variance of individual estimates by combining multiple estimates from different models, resulting in a model with greater stability (see Figure 2.13) :

— **Bagging** : This method involves training multiple homogeneous weak learners independently and in parallel, then averaging their predictions to determine the final model output.

— **Boosting** : Unlike Bagging, Boosting trains homogeneous weak learners sequentially, with each learner attempting to correct the errors of its predecessor to improve the overall model predictions.

Next, we will examine Bagging and Boosting in more detail and highlight their differences.

Boosting

Boosting is an ensemble modeling technique that aims to create a strong classifier from several weak classifiers by building models sequentially. Initially, a model is constructed

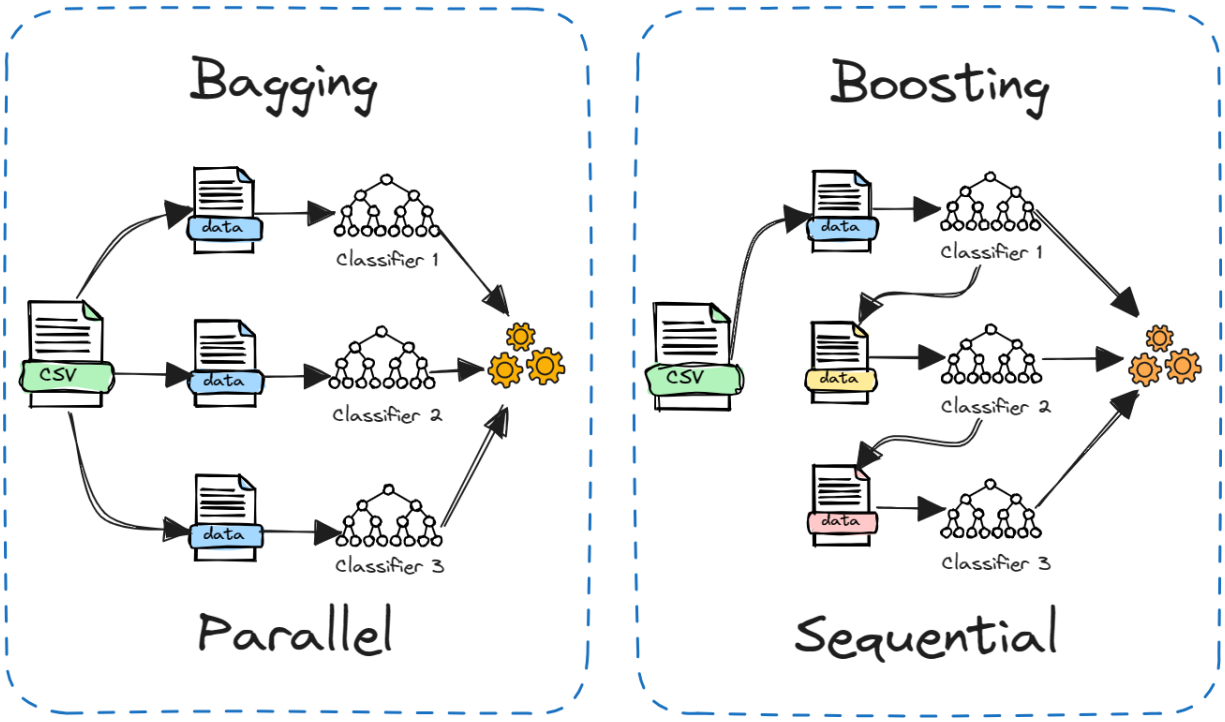


FIGURE 2.13 – Bagging Vs. Boosting

Source : www.datacamp.com

using the training data. The second model is then created to correct the errors present in the first model. This process continues, with each subsequent model focusing on the residuals of the previous one, until the entire training dataset is accurately predicted or the maximum number of models is reached.

There are several boosting algorithms, among which the most notable include AdaBoost, Gradient Boosting, and XGBoost. The original boosting algorithms, proposed by Robert Schapire and Yoav Freund, were not adaptive and could not fully exploit the weak learners. Schapire and Freund then developed AdaBoost, an adaptive boosting algorithm that won the prestigious Gödel Prize. AdaBoost, short for Adaptive Boosting, was the first successful boosting algorithm developed for binary classification, combining multiple “weak classifiers” into a single “strong classifier”.

Boosting Algorithm

Implementation Steps of Boosting (see Figure 2.14) :

1. Initialize the dataset and assign equal weight to each data point.
2. Provide this as input to the model and identify the wrongly classified data points.
3. Increase the weight of the wrongly classified data points and decrease the weights of correctly classified data points. Normalize the weights of all data points.
4. If the desired results are achieved, proceed to the next step. Otherwise, return to step 2.
5. End the algorithm when the required results are obtained or the maximum number of iterations is reached.

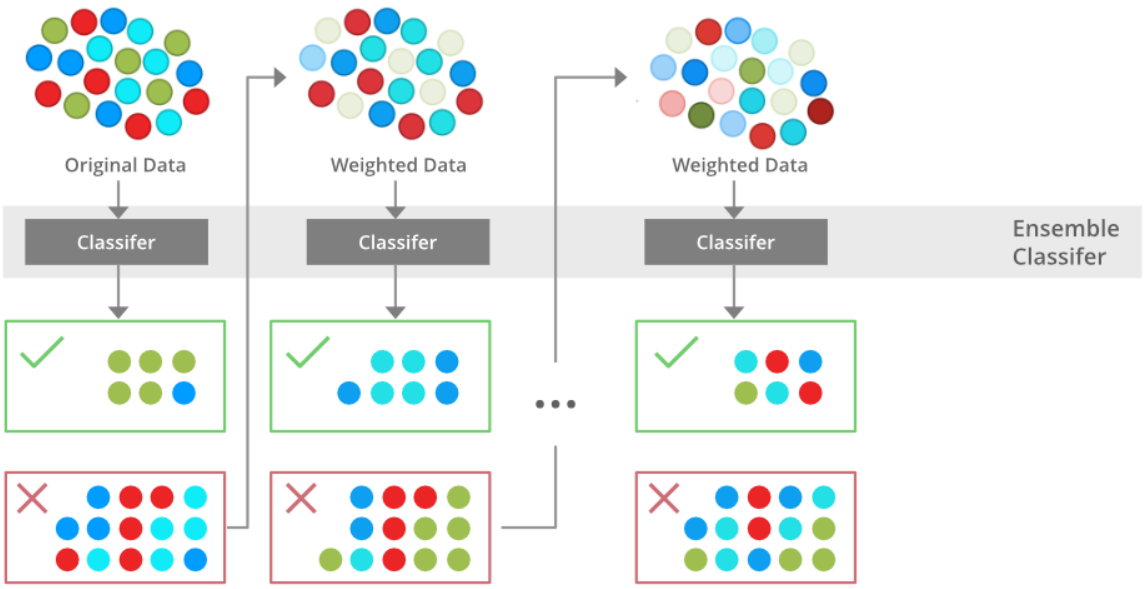


FIGURE 2.14 – An illustration presenting the intuition behind the boosting algorithm, consisting of the parallel learners and weighted dataset

Source : www.geeksforgeeks.org

In agronomy, Boosting can be utilized to classify crop diseases, predict crop types based on various features, and detect anomalies in agricultural data.

In regression tasks, Boosting algorithms improve the model by reducing the residuals of the previous predictions. Popular Boosting algorithms for regression include Gradient Boosting, XGBoost, and LightGBM. These algorithms can be used to predict continuous variables such as crop yields, soil moisture levels, and nutrient content based on inputs like weather data, soil characteristics, and farming practices.

Boosting is a powerful and flexible technique capable of handling various types of data and achieving high accuracy. It is particularly effective in reducing bias and variance, making it a popular choice for many real-world applications, including agriculture.

Bagging

Bootstrap Aggregating, also known as bagging, is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. It decreases the variance and helps to avoid overfitting, making it particularly effective for high-variance models like decision trees. Bagging works by training multiple instances of a base learner on different random subsets of the training data and then averaging their predictions.

In agronomy, Bagging can be used to classify different types of crops based on features such as soil properties, weather patterns, and plant characteristics. Additionally, it can be used to predict continuous outcomes such as crop yields based on inputs like fertilizer use, irrigation levels, and weather conditions.

Imagine we have a dataset D consisting of d tuples. For each iteration i, we create a training set D_i by randomly sampling d tuples from D with replacement (this process is known as bootstrapping, and it allows for duplicate entries in D_i). We then train a classifier model M_i on this bootstrapped training set D_i. Each classifier M_i provides a prediction. To determine the final prediction, the bagged model M^* aggregates these predictions, typically by

voting in the case of classification tasks, and assigns the most common prediction to the unknown sample X.

Bagging Algorithm

Implementation Steps of Bagging (see Figure 2.15) :

1. Generate multiple subsets from the original dataset by randomly selecting observations with replacement, ensuring each subset contains the same number of tuples as the original dataset.
2. Train a base model on each of these bootstrapped subsets.
3. Train each model independently and in parallel, ensuring they do not influence each other during the learning process.
4. Combine the predictions from all the trained models to make the final prediction, typically through methods such as majority voting for classification tasks or averaging for regression tasks.

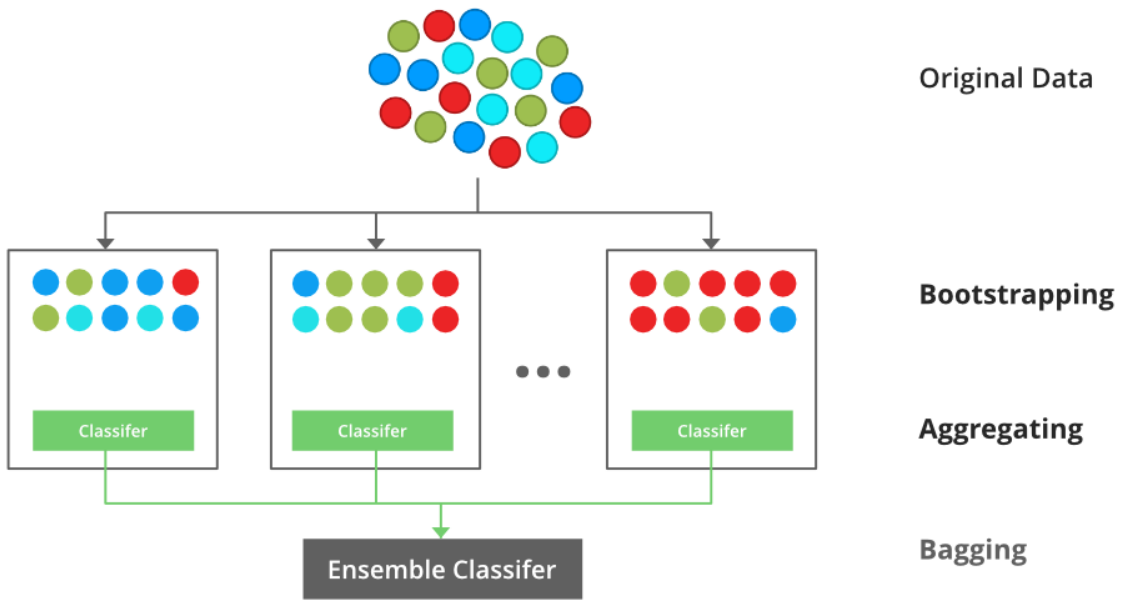


FIGURE 2.15 – An illustration for the concept of bootstrap aggregating (Bagging)
Source : www.geeksforgeeks.org

Bagging is particularly effective in reducing variance and preventing overfitting, especially for high-variance models like decision trees. By aggregating the predictions of multiple models, Bagging creates a more robust and reliable model, making it a popular choice in many real-world applications, including agronomy.

The Random Forest model uses Bagging, where decision tree models with higher variance are present. It makes random feature selection to grow trees. Several random trees make a Random Forest.

2.4.5 Naive Bayes

Naive Bayes is a family of simple and effective probabilistic classification algorithms based on Bayes' Theorem. It assumes that the features are conditionally independent given the class label, which simplifies the computation and makes it a fast and scalable solution for various classification problems.

Naive Bayes is particularly effective for high-dimensional datasets and is widely used in text classification, spam filtering, and sentiment analysis. Despite its simplicity and the strong independence assumption, it often performs surprisingly well in many real-world applications.

Naive Bayes calculates the probability of each class given a set of features using Bayes' Theorem :

$$P(C|X) = \frac{P(X|C) \cdot P(C)}{P(X)} \tag{2.17}$$

Where :

— $P(C|X)$ is the posterior probability of class C given the features X.

— $P(X|C)$ is the likelihood of the features X given class C.

— $P(C)$ is the prior probability of class C.

— $P(X)$ is the marginal probability of the features X.

Given the independence assumption, the likelihood $P(X|C)$ is decomposed into the product of the probabilities of individual features :

$$P(X|C) = \prod_{i=1}^{n} P(x_i|C) \tag{2.18}$$

This simplifies the computation, as the algorithm only needs to estimate the probabilities of individual features given the class.

Key assumptions of Naive Bayes include :

— Conditional independence : Features are assumed to be independent given the class label.

— Data is drawn from a multinomial, Bernoulli, or Gaussian distribution, depending on the type of Naive Bayes used.

Despite its strong independence assumption, Naive Bayes is a powerful and efficient tool for classification tasks, particularly when dealing with large-scale and high-dimensional data. It is widely used in text classification, medical diagnosis, and real-time prediction systems, where its simplicity and speed are valuable advantages.

To illustrate Naive Bayes in the field of agronomy, consider the following example :
Researchers at AgroTech Institute are developing a system to classify soil samples based on chemical composition and other properties. By collecting data on various soil characteristics and their corresponding classifications, they can use the Naive Bayes algorithm to predict the class of new soil samples (see Figure 2.16).

2.4.6 Gaussian Process Regression

Gaussian Process Regression (GPR) is a non-parametric, Bayesian approach to regression that provides a probabilistic prediction of the output. Unlike traditional regression me-

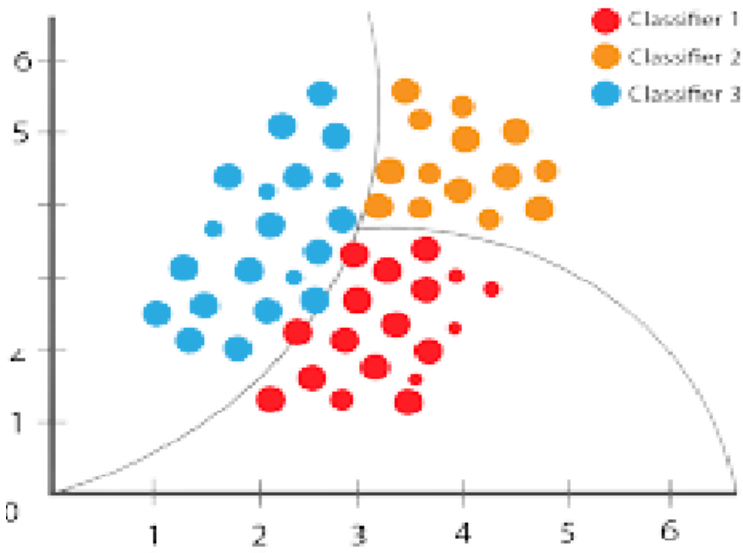


FIGURE 2.16 – Naive Bayes

Source : widyawati2023comparison

thods, GPR does not assume a fixed form for the underlying function and instead defines a distribution over possible functions, making it highly flexible (see Figure 2.17).

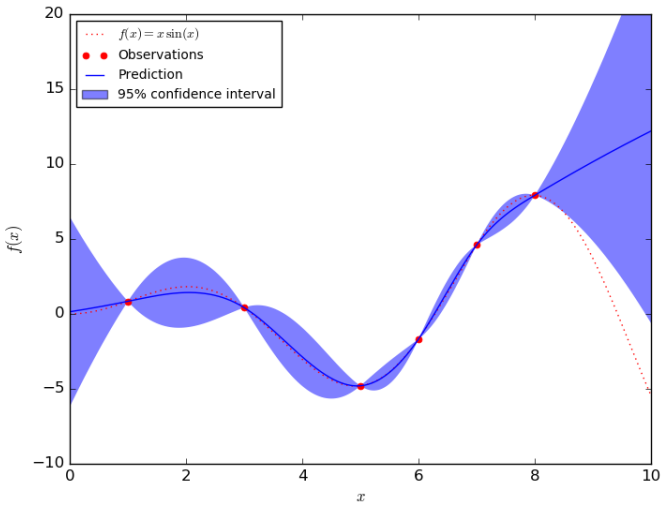


FIGURE 2.17 – Gaussian Process Regression

Gaussian Process Regression models the relationship between the input X and the output y by defining a Gaussian process prior over functions (2.19). This prior is characterized by a mean function $m(X)$ and a covariance function $k(X, X')$, which encodes assumptions about the function's smoothness and other properties.

$$f(X) \sim \mathcal{GP}(m(X), k(X, X')) \tag{2.19}$$

Where $\mathcal{GP}$ denotes the Gaussian process, $m(X)$ is the mean function, and $k(X, X')$ is the covariance function. The choice of the covariance function k significantly affects the model's predictions and flexibility.

2.4.7 Bayesian Linear Regression

Bayesian Linear Regression is a probabilistic approach to linear regression that incorporates prior distributions on the parameters and updates these priors with data to form posterior distributions. This approach provides a full distribution over possible models, offering a measure of uncertainty in the predictions (see Figure 2.18).

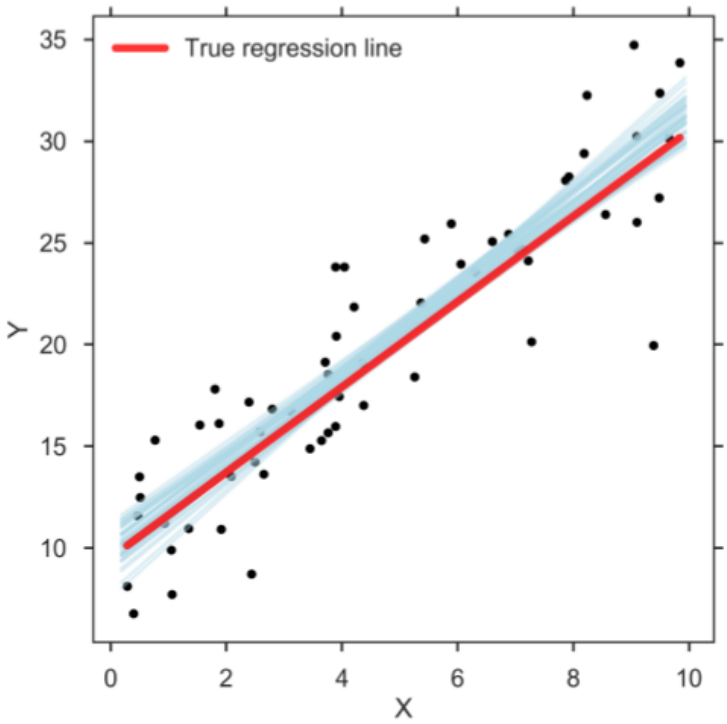


FIGURE 2.18 – Bayesian Linear Regression

Bayesian Linear Regression modifies the standard linear regression by incorporating prior distributions over the model parameters (2.20). The posterior distributions are obtained by combining these priors with the likelihood of the observed data, resulting in a more robust model, especially with limited data.

$$p(\omega|x, y) \propto p(y|x, \omega)p(\omega) \tag{2.20}$$

Where $p(\omega|x, y)$ is the posterior distribution of the parameters given the data, $p(y|x, \omega)$ is the likelihood of the data given the parameters, and $p(\omega)$ is the prior distribution of

the parameters. This Bayesian framework provides a probabilistic interpretation of the regression model.

CHAPTER 3

Unsupervised Learning

Unsupervised learning is a machine learning approach where models are trained without the need for labeled datasets. These models autonomously uncover hidden patterns and insights within the provided data, much like the human brain processes new information. It can be described as follows :

> *Unsupervised learning is a type of machine learning where models are trained on unlabeled datasets and are permitted to operate on this data without supervision*.

Unlike supervised learning, unsupervised learning is not directly applicable to regression or classification problems due to the absence of corresponding output data. The primary objective of unsupervised learning is to reveal the underlying structure of a dataset, group data based on similarities, and represent the dataset in a more compact form.

> Consider an unsupervised learning algorithm applied to a dataset containing various soil samples from different regions. The algorithm, which has no prior knowledge of the features within these samples, is tasked with identifying patterns by clustering the samples based on their similarities, such as nutrient content and pH levels.

Unsupervised learning holds significant importance for several reasons :

— It excels at deriving valuable insights from data, revealing hidden patterns and structures.

— The process mirrors human experiential learning, thereby aligning closely with the core principles of artificial intelligence.

— It effectively handles unlabeled and uncategorized data, which enhances its practical applicability.

— Many real-world datasets lack corresponding output labels, making unsupervised learning essential in such scenarios.

The process of unsupervised learning can be demonstrated as follows : Beginning with unlabeled input data, which is neither categorized nor associated with specific outputs, the machine learning model is trained on this data. Initially, the model analyzes the raw data to detect hidden patterns. Afterward, it applies suitable algorithms, such as k-means clustering or decision trees. These algorithms then group the data objects based on their similarities and differences.

Unsupervised learning algorithms can be broadly divided into two primary categories :

— **Clustering :** This approach involves grouping objects into clusters where items within each cluster exhibit high similarity, while significantly differing from items in other clusters. Cluster analysis uncovers commonalities among data objects and organizes them based on these shared characteristics.

— **Association :** This method focuses on uncovering relationships among variables within large datasets through association rules. It identifies sets of items that frequently occur together, which can be valuable for various applications. For instance, in agronomy, an association rule might reveal that specific soil conditions often coincide with certain crop diseases, aiding in better crop management practices.

Some prominent unsupervised learning algorithms include :

— K-means clustering

— K-nearest neighbors (KNN)

— Hierarchical clustering

— Anomaly detection

— Neural networks

— Principal Component Analysis (PCA)

— Independent Component Analysis (ICA)

— Apriori algorithm

— Singular Value Decomposition (SVD)

3.1 K-Means Clustering

K-Means Clustering is a popular unsupervised learning algorithm used to partition data into k distinct clusters based on their similarities. It aims to minimize the variance within each cluster and maximize the variance between clusters.

The mathematical formulation of K-Means Clustering involves the following steps :

1. Let $\mathbf{X}$ be the $m \times n$ data matrix, where m is the number of samples and n is the number of features.

2. **Initialize the Centroids** : Select k initial centroids $\{\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_k\}$.

3. **Assign Points to Clusters** : For each data point $\mathbf{x}_i$, compute the Euclidean distance to each centroid $\mathbf{c}_j$:

$$d(\mathbf{x}_i, \mathbf{c}_j) = \sqrt{\sum_{f=1}^{n} (x_{i,f} - c_{j,f})^2} \tag{3.1}$$

   Assign $\mathbf{x}_i$ to the cluster with the nearest centroid.

4. **Update Centroids** : Recalculate the centroid of each cluster as the mean of all points assigned to that cluster :

$$\mathbf{c}_j = \frac{1}{|C_j|} \sum_{\mathbf{x}_i \in C_j} \mathbf{x}_i \tag{3.2}$$

   where C_j is the set of points in the j-th cluster and $|C_j|$ is the number of points in cluster j.

5. **Repeat** : Repeat steps 3 and 4 until the centroids do not change significantly :

$$\mathbf{c}_j^{(t+1)} = \mathbf{c}_j^{(t)} \quad \forall j \tag{3.3}$$

K-Means Algorithm

The steps for K-Means Clustering are as follows :

1. **Choose the number of clusters** k : Decide on the number of clusters to create.
2. **Initialize the Centroids** : Select k initial centroids randomly or based on some heuristic.
3. **Assign Points to Clusters** : Assign each data point to the nearest centroid, forming k clusters.
4. **Update Centroids** : Calculate the new centroids as the mean of all points in each cluster.
5. **Repeat** : Repeat the assignment and update steps until the centroids stabilize or a maximum number of iterations is reached.

K-Means Clustering is particularly useful in agronomy for tasks such as grouping similar crop types, soil samples, or weather patterns based on various features.

We have a dataset of soil samples with measurements of different properties such as nitrogen content, phosphorus content, potassium content, and pH level. Our goal is to cluster these soil samples into $k = 3$ clusters based on these properties.

3.2 K-Nearest Neighbors (KNN)

K-Nearest Neighbors (KNN) is a simple, non-parametric, and instance-based learning algorithm used for both classification and regression tasks. In KNN, the classification or prediction for a new data point is based on the k closest data points in the feature space. The mathematical formulation of KNN involves calculating the distance between data points and making predictions based on the k nearest neighbors.

1. Let $\mathbf{X}$ be the $m \times n$ data matrix, where m is the number of samples and n is the number of features. Let $\mathbf{y}$ be the m-dimensional vector of target values.
2. **Distance Calculation** : For a new data point $\mathbf{x}_{\text{new}}$, compute the Euclidean distance

to each existing data point $\mathbf{x}_i$:

$$d(\mathbf{x}_{\text{new}}, \mathbf{x}_i) = \sqrt{\sum_{j=1}^{n} (x_{\text{new},j} - x_{i,j})^2} \tag{3.4}$$

where $x_{\text{new},j}$ is the j-th feature of the new data point and $x_{i,j}$ is the j-th feature of the i-th data point.

3. **Find the Nearest Neighbors** : Identify the k data points with the smallest distances to $\mathbf{x}_{\text{new}}$.

4. **Make a Prediction** :

   For classification : Assign the class label $\hat{y}_{\text{new}}$ based on majority voting among the k nearest neighbors :

$$\hat{y}_{\text{new}} = \text{mode}(y_{i_1}, y_{i_2}, \ldots, y_{i_k}) \tag{3.5}$$

   where $y_{i_1}, y_{i_2}, \ldots, y_{i_k}$ are the target values of the k nearest neighbors.

   For regression : Compute the average target value $\hat{y}_{\text{new}}$ of the k nearest neighbors :

$$\hat{y}_{\text{new}} = \frac{1}{k} \sum_{j=1}^{k} y_{i_j} \tag{3.6}$$

KNN Algorithm

The steps for K-Nearest Neighbors (KNN) are as follows :

1. **Choose the number of neighbors** k : Select the number of nearest neighbors to consider for classification or regression.
2. **Compute the Distance** : Calculate the distance between the new data point and all existing data points using a suitable distance metric (e.g., Euclidean distance).
3. **Find the Nearest Neighbors** : Identify the k nearest neighbors to the new data point based on the computed distances.
4. **Make a Prediction** :
   - For classification : Assign the class label that is most common among the k nearest neighbors (majority voting).
   - For regression : Compute the average (or weighted average) of the target values of the k nearest neighbors.

KNN is widely used in various fields, including agronomy, for tasks such as crop classification, soil property prediction, and disease detection, where the relationship between the features and the target variable may be complex and non-linear.
We have a dataset of soil samples with measurements of different properties such as nitrogen content, phosphorus content, potassium content, and pH level. Our goal is to classify the soil samples into different soil types based on these properties.

3.3 Hierarchical Clustering

Hierarchical clustering is a method of cluster analysis which seeks to build a hierarchy of clusters. This technique can be classified into two types : agglomerative (bottom-up) and divisive (top-down). Agglomerative clustering starts with each observation as its own cluster and iteratively merges them, while divisive clustering starts with all observations in a single cluster and iteratively splits them.
The mathematical formulation of hierarchical clustering involves calculating the distance matrix and updating it as clusters are merged.

1. Let $\mathbf{X}$ be the $m \times n$ data matrix, where m is the number of samples and n is the number of features.
2. Compute the pairwise distance matrix $\mathbf{D}$:

$$d_{ij} = \sqrt{\sum_{k=1}^{n} (x_{ik} - x_{jk})^2} \tag{3.7}$$

   where d_{ij} is the Euclidean distance between sample i and sample j.
3. Start with each data point as its own cluster. Let $\mathcal{C} = \{C_1, C_2, \ldots, C_m\}$ be the initial set of clusters.
4. Find the closest pair of clusters (C_i, C_j) and merge them into a new cluster C_{ij}. Update the set of clusters :

$$\mathcal{C} = (\mathcal{C} \setminus \{C_i, C_j\}) \cup \{C_{ij}\} \tag{3.8}$$

5. Update the distance matrix to reflect the new distances between the merged cluster and the remaining clusters. Common linkage methods include :

Single Linkage (Minimum Distance) :

$$d(C_{ij}, C_k) = \min(d(C_i, C_k), d(C_j, C_k)) \tag{3.9}$$

Complete Linkage (Maximum Distance) :

$$d(C_{ij}, C_k) = \max(d(C_i, C_k), d(C_j, C_k)) \tag{3.10}$$

Average Linkage :

$$d(C_{ij}, C_k) = \frac{|C_i| \cdot d(C_i, C_k) + |C_j| \cdot d(C_j, C_k)}{|C_i| + |C_j|} \tag{3.11}$$

where $|C_i|$ and $|C_j|$ are the sizes of clusters C_i and C_j respectively.

6. Repeat steps 4 and 5 until all data points are in a single cluster.

Hierarchical clustering Algorithm

The steps for hierarchical clustering are as follows :

1. **Calculate the Distance Matrix** : Compute the pairwise distance between all data points using a suitable distance metric (e.g., Euclidean distance).
2. **Merge Clusters** : Starting with each data point as its own cluster, iteratively merge the two closest clusters until all points are in a single cluster.
3. **Update the Distance Matrix** : After each merge, update the distance matrix to reflect the new distances between clusters.
4. **Repeat** : Continue merging clusters and updating the distance matrix until only one cluster remains, creating a dendrogram in the process.

Hierarchical clustering is widely used in various fields, including agronomy applications where it is necessary to group similar soil samples, crops, or environmental conditions. It allows the visualization of the data structure in a dendrogram, which shows the arrangement of the clusters formed by the algorithm.

We have soil samples with measurements of different properties such as nitrogen content (mg/kg), phosphorus content (mg/kg), potassium content (mg/kg), and pH level. Our goal is to group these soil samples based on their similarity. The dataset might be structured as follows :

Sample	Nitrogen	Phosphorus	Potassium	pH
1	30	20	40	6.5
2	25	25	35	6.8
3	22	30	30	7.0
4	35	15	45	6.2
5	40	10	50	6.0

We apply hierarchical clustering to this dataset to group the soil samples based on their properties.

3.4 Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a statistical technique used for dimensionality reduction, data visualization, and feature extraction. It transforms the original features into a new set of uncorrelated features called principal components, which capture the maximum variance in the data.

The mathematical formulation of PCA is as follows :

1. Let $\mathbf{X}$ be the $m \times n$ data matrix, where m is the number of samples and n is the number of features.
2. Standardize the data by centering the mean of each feature to 0 :

$$\mathbf{X}_c = \mathbf{X} - \mu \tag{3.12}$$

   where μ is the mean of each feature.
3. Calculate the covariance matrix $\mathbf{C}$:

$$\mathbf{C} = \frac{1}{m-1}\mathbf{X}_c^T\mathbf{X}_c \tag{3.13}$$

4. Perform eigen decomposition on the covariance matrix to find the eigenvalues and eigenvectors :

$$\mathbf{C}\mathbf{v} = \lambda\mathbf{v} \tag{3.14}$$

where $\mathbf{v}$ are the eigenvectors (principal components) and λ are the eigenvalues (variance explained by each principal component).

5. Sort the eigenvalues in descending order and select the top k eigenvectors corresponding to the largest eigenvalues to form the projection matrix $\mathbf{W}$.

6. Transform the original data to the new subspace :

$$\mathbf{Z} = \mathbf{X}_c\mathbf{W} \tag{3.15}$$

where $\mathbf{Z}$ is the transformed data in the new subspace.

PCA Algorithm

The steps for PCA are as follows :

1. **Standardize the Data** : Scale the data so that each feature has a mean of 0 and a standard deviation of 1.
2. **Compute the Covariance Matrix** : Calculate the covariance matrix to understand the relationships between the features.
3. **Perform Eigen decomposition** : Compute the eigenvalues and eigenvectors of the covariance matrix. The eigenvectors represent the principal components, and the eigenvalues represent the amount of variance captured by each principal component.
4. **Sort and Select Principal Components** : Sort the principal components based on their eigenvalues in descending order and select the top k components that capture the most variance.
5. **Transform the Data** : Project the original data onto the selected principal components to obtain the reduced-dimension dataset.

PCA is particularly useful when dealing with high-dimensional data. By reducing the number of dimensions, it helps simplify the dataset while retaining as much information as possible. This makes it easier to visualize the data and can improve the performance of machine learning algorithms.

PCA is widely used in various fields, including agronomy, for tasks such as soil classification, crop yield prediction, and environmental monitoring.

Consider an example in agronomy : We have a dataset with measurements of various soil properties, such as nitrogen content, phosphorus content, potassium content, and pH level. Our goal is to reduce the dimensionality of this dataset to visualize the relationships between the soil samples.

3.5 Independent Component Analysis (ICA)

Independent Component Analysis (ICA) is a computational method for separating a multivariate signal into additive, independent components. It is particularly useful in scenarios where it is necessary to identify and separate underlying factors that are not directly observable, such as in the analysis of complex agronomic data involving mixed sources of variation.

The mathematical formulation of ICA involves the following steps :

1. **Center the Data** : Subtract the mean of each variable to ensure the data has zero mean.

$$\mathbf{X}_c = \mathbf{X} - \mathbf{E}[\mathbf{X}] \tag{3.16}$$

where $\mathbf{E}[\mathbf{X}]$ is the mean of $\mathbf{X}$.

2. **Whiten the Data** : Transform the centered data so that its covariance matrix is the identity matrix, making the data uncorrelated and with unit variance.

$$\mathbf{X}_{whitened} = \mathbf{V}\mathbf{\Lambda}^{-\frac{1}{2}}\mathbf{V}^T\mathbf{X}_c \tag{3.17}$$

where $\mathbf{V}$ and $\mathbf{\Lambda}$ are the eigenvector and eigenvalue matrices of the covariance matrix of $\mathbf{X}_c$.

3. **Apply ICA** : Decompose the whitened data $\mathbf{X}_{whitened}$ into a product of mixing matrix $\mathbf{A}$ and independent components $\mathbf{S}$:

$$\mathbf{X}_{whitened} = \mathbf{AS} \tag{3.18}$$

The goal is to estimate $\mathbf{A}$ and $\mathbf{S}$ such that the components in $\mathbf{S}$ are as statistically independent as possible.

ICA Algorithm

The steps for Independent Component Analysis are as follows :

1. **Center and Whiten the Data** : Preprocess the data by centering (subtracting the mean) and whitening (decorrelating and scaling) to make the variance equal across dimensions.
2. **Apply ICA** : Use an ICA algorithm to separate the mixed signals into independent components.
3. **Analyze the Components** : Interpret the independent components to understand the underlying factors.

ICA is widely used in various fields, including agronomy, for tasks such as identifying and isolating independent sources in agronomy, such as separating different environmental effects on crop yield from soil properties and other variables.

Consider an example in agronomy : We have a dataset containing mixed signals from multiple sensors measuring soil moisture, temperature, and nutrient levels across different locations. Our goal is to use ICA to separate these mixed signals into independent components that represent distinct environmental factors affecting soil properties.

3.6 Apriori Algorithm

The Apriori algorithm is a popular method used in association rule mining to identify frequent itemsets and generate association rules from a dataset. It is widely utilized in market basket analysis and other fields to discover relationships among variables.

The mathematical formulation of the Apriori algorithm involves the following steps :

1. **Define Support** : The support of an itemset I is the proportion of transactions in the dataset that contain I.

$$\text{Support}(I) = \frac{\text{Number of transactions containing } I}{\text{Total number of transactions}} \tag{3.19}$$

2. **Define Confidence** : The confidence of an association rule $I \rightarrow J$ is the proportion of transactions that contain J among those that contain I.

$$\text{Confidence}(I \rightarrow J) = \frac{\text{Support}(I \cup J)}{\text{Support}(I)} \tag{3.20}$$

3. **Define Lift** : The lift of an association rule $I \rightarrow J$ measures the strength of the rule over random chance.

$$\text{Lift}(I \rightarrow J) = \frac{\text{Support}(I \cup J)}{\text{Support}(I) \times \text{Support}(J)} \tag{3.21}$$

4. **Apriori Property** : Use the property that all non-empty subsets of a frequent itemset must also be frequent. This helps in reducing the search space.

Apriori algorithm Algorithm

The steps for the Apriori algorithm are as follows :

1. **Identify Frequent Itemsets** : Generate itemsets that occur frequently in the dataset, based on a minimum support threshold.
2. **Generate Association Rules** : From the frequent itemsets, derive association rules that meet a minimum confidence threshold.

The Apriori algorithm is widely used in various fields, including agronomy, the Apriori algorithm can be applied to identify patterns and associations among different agricultural practices, crop types, and environmental conditions. This can help in understanding the co-occurrence of certain practices and their impacts on crop yields and soil health.

Consider an example in agronomy : We have a dataset containing records of various agricultural practices (such as irrigation, fertilization, crop rotation) and crop yields. Our goal is to use the Apriori algorithm to identify frequent combinations of practices that are associated with high crop yields.

3.7 Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) is a powerful matrix factorization technique widely used in various applications, including dimensionality reduction, data compression, and noise reduction. In agronomy, SVD can be applied to analyze complex datasets, such as soil properties, crop yields, and environmental factors, by extracting essential features and reducing data dimensionality.

The mathematical formulation of SVD involves the following steps :

1. **Construct the Data Matrix** : Let $\mathbf{X}$ be the $m \times n$ data matrix, where m is the number of samples and n is the number of features.

2. **Perform SVD** : Decompose the matrix $\mathbf{X}$ into three matrices :

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{3.22}$$

   where :
   $\mathbf{U}$ is an $m \times m$ orthogonal matrix whose columns are the left singular vectors of $\mathbf{X}$.
   $\mathbf{\Sigma}$ is an $m \times n$ diagonal matrix whose diagonal elements are the singular values of $\mathbf{X}$.
   $\mathbf{V}$ is an $n \times n$ orthogonal matrix whose columns are the right singular vectors of $\mathbf{X}$.

3. **Truncate the Matrices** : Retain only the top k singular values in $\mathbf{\Sigma}$ and the corresponding columns in $\mathbf{U}$ and $\mathbf{V}$, resulting in the truncated matrices $\mathbf{U}_k$, $\mathbf{\Sigma}_k$, and $\mathbf{V}_k$:

$$\mathbf{X} \approx \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T \tag{3.23}$$

4. **Reconstruct the Approximate Matrix** : Use the truncated matrices to reconstruct an approximation of the original matrix $\mathbf{X}$:

$$\mathbf{X}_k = \mathbf{U}_k\mathbf{\Sigma}_k\mathbf{V}_k^T \tag{3.24}$$

SVD Algorithm

The steps for Singular Value Decomposition are as follows :

1. **Construct the Data Matrix** : Form the $m \times n$ data matrix $\mathbf{X}$, where m is the number of samples and n is the number of features.
2. **Perform SVD** : Decompose the matrix $\mathbf{X}$ into three matrices : $\mathbf{U}$, $\mathbf{\Sigma}$, and $\mathbf{V}^T$.
3. **Truncate the Matrices** : Retain only the top k singular values and corresponding vectors to reduce dimensionality.
4. **Reconstruct the Approximate Matrix** : Use the truncated matrices to reconstruct an approximation of the original matrix.

SVD is particularly valuable for handling large, high-dimensional datasets in agronomy, enabling more efficient storage, soil property analysis, crop yield prediction, and environmental monitoring, processing, and analysis of the data.

Consider an example in agronomy : We have a dataset containing measurements of different soil properties (e.g., nitrogen content, phosphorus content, potassium content, and pH level) across various locations. Our goal is to use SVD to reduce the dimensionality of this dataset while preserving its essential features.

Algorithm	**Advantages**	**Disadvantages**
K-means Clustering	— Simple and easy to implement — Efficient for large datasets — Works well with spherical clusters	— Requires specification of k (number of clusters) — Sensitive to initial centroids — Not suitable for non-spherical clusters
Hierarchical Clustering	— No need to specify number of clusters in advance — Produces a dendrogram for visual analysis	— Computationally intensive for large datasets — Not scalable — Sensitive to noise and outliers
Principal Component Analysis (PCA)	— Reduces dimensionality of data — Captures most of the variance in the data — Improves computational efficiency	— Loses some information in the process — Assumes linearity — Components may be hard to interpret
Independent Component Analysis (ICA)	— Finds statistically independent components — Useful for blind source separation	— Sensitive to noise — Computationally expensive — Assumes non-Gaussian sources

K-nearest Neighbors (KNN)	— Simple and intuitive — Effective with small datasets — Non-parametric	— Computationally expensive for large datasets — Requires selection of k (number of neighbors) — Sensitive to irrelevant features
Apriori Algorithm	— Easy to implement — Provides valuable insights into data patterns	— Computationally intensive — Requires large support and confidence thresholds
Singular Value Decomposition (SVD)	— Effective for dimensionality reduction — Provides optimal low-rank approximations	— Computationally intensive — May be sensitive to noise
Anomaly Detection	— Detects rare and unusual patterns — Useful for fraud detection and fault diagnosis	— May have high false positive rate — Requires well-defined normal behavior
Neural Networks	— Can model complex patterns — Flexible and powerful	— Requires large datasets — Computationally intensive — Prone to overfitting

TABLE 3.1 – Comparison of Unsupervised Learning Algorithms

CHAPTER 4

Neural Networks and Deep Learning

Introduction

In recent years, neural networks and deep learning have revolutionized the field of artificial intelligence, enabling significant advancements in various domains such as image recognition, natural language processing, and autonomous systems. This chapter provides a comprehensive overview of these powerful computational frameworks, exploring both foundational concepts and advanced architectures.

4.1 Neural Networks

The concept of "Artificial Neural Network" (ANN) is inspired by biological neural networks that constitute the human brain's architecture. Just as neurons in the brain are interconnected, artificial neural networks feature nodes (neurons) linked across various layers.

As illustrated in the accompanying figure, biological neural networks have dendrites that represent inputs in ANNs, cell nuclei that correspond to nodes, synapses that signify weights, and axons that represent outputs (see Figure 4.1 and table 4.1).

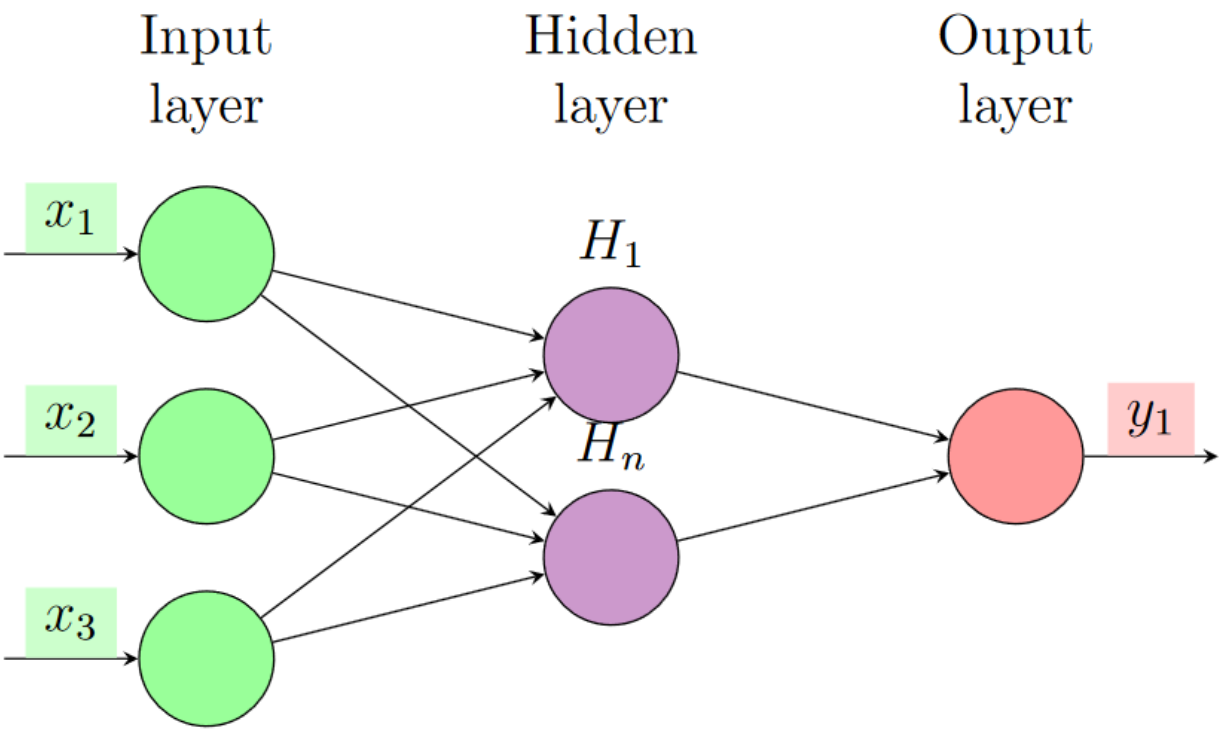


FIGURE 4.1 – Anatomy of a neuron.

Source : towardsdatascience.com

Artificial Neural Network	Biological Neural Network
Inputs	Dendrites
Nodes	Cell nucleus
Weights	Synapse
Output	Axon

TABLE 4.1 – Relationship Between Biological and Artificial Neural Networks

ANNs, part of artificial intelligence, aim to replicate the neuron networks of the human brain, enabling computers to process information and make decisions in a manner akin to human cognition. The design of ANNs involves programming computers to simulate the interconnectivity found in brain cells.

The human brain comprises approximately 100 billion neurons, each linked to between 1,000 and 100,000 other neurons. Data in the brain is stored in a distributed fashion, allowing simultaneous retrieval of multiple data pieces, akin to parallel processing.
To illustrate, consider a digital logic gate, such as an "OR" gate, which yields an "On" output if one or both inputs are "On." Unlike this binary function, our brain's responses adapt through learning.

4.1.1 Architecture of an Artificial Neural Network

An Artificial Neural Network (ANN) is composed of interconnected layers that process input data to produce an output. These layers include the input layer, one or more hidden layers, and the output layer. Each layer plays a specific role in the computation process :

— **Input Layer :** This layer accepts diverse input formats as specified by the programmer.

— **Hidden Layer :** Situated between the input and output layers, the hidden layer performs calculations to uncover latent features and patterns.

— **Output Layer :** After passing through the hidden layer, the transformed inputs yield the final output.

The ANN computes the weighted sum of inputs plus a bias, represented as a transfer function. The weighted total is fed into an activation function to determine node activation, allowing only activated nodes to contribute to the output layer. Various activation functions are available depending on the task (see Figure 4.2).

4.1.2 Operation of Artificial Neural Networks

Artificial Neural Networks (ANNs) can be conceptualized (see Figure 4.3) as weighted directed graphs where neurons serve as nodes, and the connections (edges) between them possess weights. Inputs, represented as patterns or vectors, are received from external sources and mathematically denoted as $x(n)$.
Each input is multiplied by its respective weight, signifying the strength of interconnections. The weighted inputs are then aggregated in a computational unit. If this sum is

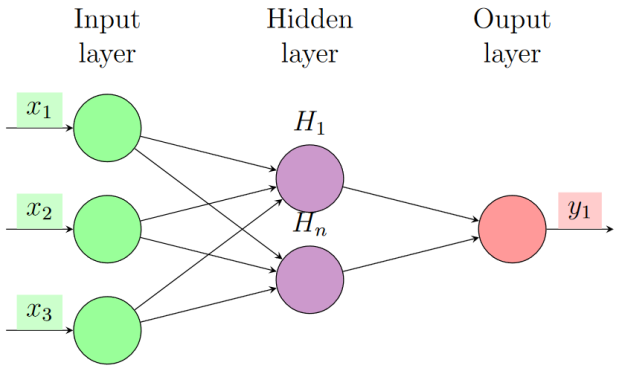


FIGURE 4.2 – Architecture of an Artificial Neural Network

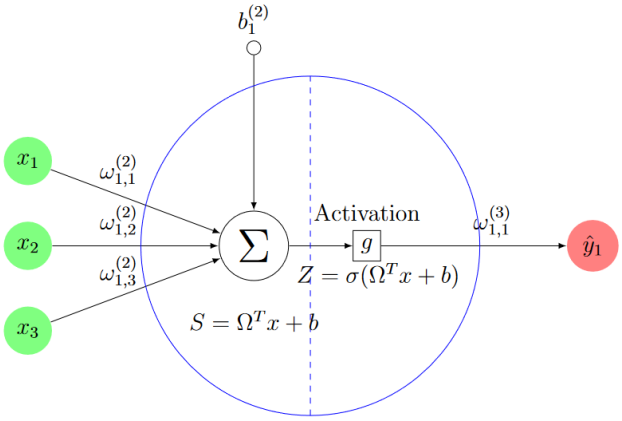


FIGURE 4.3 – Perceptron

zero, a bias is added to prevent a zero output. This bias, with a weight of one, helps keep responses within desired limits.

The weighted inputs are processed through an activation function, which can be linear or non-linear. Common activation functions include binary, linear, and hyperbolic tangent sigmoidal functions (see Figure 4.4).

A specific type of ANN is built around a unit known as a perceptron, as depicted in the accompanying figure. A perceptron processes a vector of real-valued inputs, computes a linear combination of these inputs, and produces an output of 1 if the result exceeds a certain threshold, and -1 otherwise. Formally, given inputs x_1 to x_n, the output $o(x_1, \ldots, x_n)$ from the perceptron can be expressed as :

$$o(x_1, \ldots, x_n) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i > \theta \\ -1 & \text{otherwise} \end{cases} \tag{4.1}$$

Here, each w_i represents a real-valued weight that defines how much influence input x_i has on the output. The term θ serves as the threshold that the weighted sum must exceed

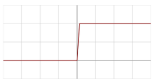
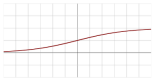
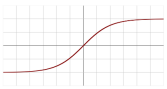
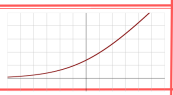
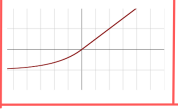
ACTIVATION FUNCTION	PLOT	EQUATION	DERIVATIVE	RANGE
Linear		f(x) = x	f'(x) = 1	(-∞, ∞)
Binary Step		$f(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x \neq 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	{0, 1}
Sigmoid		$f(x) = \sigma(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1-f(x))$	(0, 1)
Hyperbolic Tangent(tanh)		$f(x) = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$	$f'(x) = 1 - f(x)^2$	(-1, 1)
Rectified Linear Unit(ReLU)		$f(x) = \begin{cases} 0 & \text{if } x<0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ \text{undefined} & \text{if } x = 0 \end{cases}$	[0, ∞)
Softplus		$f(x) = \ln(1+e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$	(0, 1)
Leaky ReLU		$f(x) = \begin{cases} 0.01x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0.01 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	(-1, 1)
Exponential Linear Unit(ELU)		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}$	$f'(x) = \begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}$	[0, ∞)

FIGURE 4.4 – Activate function caracteristic

Source : iq.opengenus.org

for the perceptron to output 1.

To streamline notation, we introduce a constant input $x_0 = 1$, allowing us to rewrite the inequality as :

$$\sum_{i=0}^{n} w_i x_i > 0 \tag{4.2}$$

In vector notation, this can be expressed as $\vec{x} \cdot \vec{w} > 0$. For convenience, the perceptron function can sometimes be denoted as $f(\vec{x})$.

Perceptrons can be interpreted as defining a hyperplane in an n-dimensional instance space (i.e., a geometric representation of data points). The perceptron will output 1 for instances on one side of the hyperplane and -1 for those on the other, as shown in the figure below. The equation for this decision boundary is given by $\vec{w} \cdot \vec{x} = 0$. However, not all sets of positive and negative examples can be separated by a hyperplane; those that can are termed linearly separable.

The decision surface illustrated for a two-input perceptron demonstrates a training set where the perceptron classifies correctly (a) and a set that is not linearly separable (b).

In this context, x_1 and x_2 represent the perceptron inputs, with positive examples marked by "+" and negative by "-". The outputs from multiple units can be fed into a subsequent layer, and Boolean functions can be expressed in disjunctive normal form as ORs of conjunctions of inputs and their negations. Negating an input to an AND perceptron can be accomplished by adjusting the sign of the corresponding weight.

4.1.3 The Perceptron Training Rule

To learn how to adjust weights for a perceptron, we start by selecting random initial weights and iteratively applying the perceptron to the training examples, modifying weights when misclassifications occur. This iterative process continues until all training examples are classified correctly. Weights are updated according to the perceptron training rule :

$$w_i \leftarrow w_i + \eta(t - o)x_i \tag{4.3}$$

Here, t is the target output for the current example, o is the perceptron's output, and η is the learning rate, a small positive constant (often around 0.1), which may decrease over time.

The intuition behind this update mechanism is straightforward. If the perceptron correctly classifies a training example, no weight adjustments are needed. Conversely, if the perceptron outputs -1 when the target is $+1$, the weights must be adjusted to increase the output towards the correct classification.

The learning process can be shown to converge within a finite number of iterations of the training rule to a weight vector that accurately classifies all training examples, given that the examples are linearly separable and a sufficiently small η is used. However, if the data is not linearly separable, convergence is not guaranteed.

While the perceptron rule is effective for linearly separable data, it may fail for non-separable examples. The delta rule addresses this issue by using gradient descent to approximate the best-fit solution. The delta rule is significant because it underpins the **Backpropagation** algorithm, which enables the learning of complex networks with many interconnected units.

To derive a weight learning rule for unthresholded perceptrons, we first establish an error measure concerning training examples.

One convenient measure is defined as :

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \tag{4.5}$$

where D represents the set of training examples, t_d is the target output, and o_d is the output from the linear unit for example d. This formulation provides a means to evaluate how well the linear unit's output aligns with the target outputs. Under specific conditions, the hypothesis minimizing E is also the most probable hypothesis given the training data.

4.1.4 Characteristics of an Artificial Neural Network (ANN)

An Artificial Neural Network (ANN) can be designed and implemented in various ways. The following characteristics define different variants of an ANN :

1. **Activation Functions** : The activation function is a crucial component of an artificial neuron, responsible for processing incoming information and transmitting it through the network. Similar to the biological neuron, the activation function in an ANN is modeled after natural processes.

   Given input signals $x_1, x_2, \ldots, x_n$ with associated weights $w_1, w_2, \ldots, w_n$ and a threshold $-w_0$, the input to the activation function can be represented as :

   $$x = w_0 + w_1 x_1 + \cdots + w_n x_n \tag{4.6}$$

   The activation function, applied to this weighted sum, determines the neuron's output. Some commonly used activation functions include sigmoid, tanh, and ReLU.

2. **Network Topology** : Network topology refers to the patterns and structures within a collection of interconnected nodes. It dictates the complexity of tasks that the network can learn, with larger and more complex networks generally capable of identifying more subtle patterns and intricate decision boundaries. However, a network's effectiveness depends not only on its size but also on how the nodes are arranged. Key aspects of network architecture include :

   (a) **The Number of Layers** :

   — *Input Layer* : Nodes in this layer receive unprocessed signals from the input data.

   — *Hidden Layers* : These layers process signals received from previous layers before passing them to the next layer. The network can have multiple hidden layers.

   — *Output Layer* : This layer generates the final predicted values.

   (b) **Direction of Information Flow** :

   — *Feedforward Networks* : In these networks, information flows in one direction, from the input layer to the output layer, without loops. The Multilayer Perceptron (MLP) is a common type of feedforward network.

   — *Recurrent Networks* : These networks allow signals to travel in both directions, forming loops. While theoretically powerful, recurrent networks are less commonly used in practice compared to feedforward networks.

   (c) **The Number of Nodes in Each Layer** :

   — *Input Nodes* : Determined by the number of features in the input data.

   — *Output Nodes* : Determined by the number of outcomes or classes to be modeled.

   — *Hidden Nodes* : The number of hidden nodes is chosen by the user and depends on factors such as the number of input nodes, the size of the training data, the amount of noise in the data, and the complexity of the learning task.

3. **The Training Algorithm** : Training an ANN involves adjusting the connection weights to improve the network's performance. The two primary algorithms for learning a single perceptron are the perceptron rule and the delta rule, used depending on whether the training dataset is linearly separable. The most commonly used algorithm

for training ANNs today is backpropagation, which efficiently updates the weights to minimize the error between the predicted and actual outputs.

4. **The Cost Function** : The cost function, also known as the loss function or error function, quantifies the difference between the network's predictions and the actual target values. It measures the performance of the ANN during training.
   The cost function is a critical component in the optimization process, guiding the adjustment of weights to improve the accuracy of the network's predictions. It may also be referred to as the objective function or scoring function, depending on the context.

— To train the parameters Ω and b, we need to define a **cost function**.

$$\hat{y}^{(i)} = \sigma(\Omega^T x^{(i)} + b) \tag{4.8}$$

— Given $\left\{(x^{(1)}, y^{(1)}), (x^{(2)}, y^{(2)}), ..., (x^{(m)}, y^{(m)})\right\}$ we want $\hat{y}^{(i)} \approx y^{(i)}$

Loss (error) function

computes the error for a single training sample.

$$L(\hat{y}^{(i)}, y^{(i)}) = \frac{1}{2}(\hat{y}^{(i)} - y^{(i)})^2 \tag{4.11}$$

$$L(\hat{y}^{(i)}, y^{(i)}) = -(y^{(i)} \times \ln(\hat{y}^{(i)}) + (1 - y^{(i)}) \times \ln(1 - \hat{y}^{(i)})) \tag{4.12}$$

— If $y^{(i)} = 1$ then : $L(\hat{y}^{(i)}, y^{(i)}) = -\ln(\hat{y}^{(i)}) \Rightarrow$ want $\hat{y}^{(i)}$ to be large or close to 1.

— If $y^{(i)} = 0$ then : $L(\hat{y}^{(i)}, y^{(i)}) = -\ln(1 - \hat{y}^{(i)}) \Rightarrow$ want $\hat{y}^{(i)}$ to be small or close to 0.

Cost function

computes the average error over all training samples.

$$\begin{aligned} J(\omega, b) &= \frac{1}{m}\sum_{i=1}^{m} L(\hat{y}^{(i)}, y^{(i)}) \\ &= -\frac{1}{m}\sum_{i=1}^{m}[y^{(i)} \times \ln(\hat{y}^{(i)}) + (1 - y^{(i)}) \times \ln(1 - \hat{y}^{(i)})] \end{aligned} \tag{4.14}$$

Artificial Neural Networks (ANNs) come in various types, each designed to address specific types of problems and inspired by different aspects of biological neural systems. The most common types include :

— **Feedback ANN :** Outputs are cycled back into the network, optimizing internal results. This type of ANN is particularly effective for solving optimization problems and is often used in recurrent neural networks (RNNs).

— **Feed-Forward ANN :** This type consists of an input layer, an output layer, and at least one hidden layer. It evaluates input patterns to determine outputs and is the simplest form of ANN, often used for tasks like image recognition and classification.

4.1.5 Forward Pass

Forward propagation, also known as the forward pass, is the process of computing and storing intermediate variables, including outputs, within a neural network. This calculation unfolds sequentially from the input layer to the output layer, establishing the foundation for subsequent stages in the neural network's operation.

$$X = \begin{bmatrix} \vdots & \vdots & \vdots \\ x^1 & x^2 & x^m \\ \vdots & \vdots & \vdots \end{bmatrix}; Z = \begin{bmatrix} \vdots & \vdots & \vdots \\ z & z & z^{[1](m)} \\ \vdots & \vdots & \vdots \end{bmatrix}; A = \begin{bmatrix} \vdots & \vdots & \vdots \\ a & a & a^{[1](m)} \\ \vdots & \vdots & \vdots \end{bmatrix} \tag{4.15}$$

$$\varphi_1 = \begin{cases} Z^1 = W^1 X + b^1 & \text{hidden layer} \\ A^1 = \phi(Z^1) & \text{hidden activation vector} \end{cases} \tag{4.16}$$

Output layer variable :

$$\omega = W^2 A^1 \tag{4.17}$$

The loss term for a single data example :

$$\Sigma = l(\omega, y) \tag{4.18}$$

$$J = \Sigma + \theta \tag{4.19}$$

with regularization term :

$$\theta = \frac{\lambda}{2}(\|W^1\|^2 + \|W^2\|^2) \tag{4.20}$$

Algorithm 5.1 Forward Pass of an MLP

The steps are as follows hochreiter2014theoretical :

1. **Initialization** :
   (a) Provide input x.
   (b) For all $i = 1$ to I do :
      — Set $a_i = x_i$.
2. **Forward Pass** :
   (a) For $\nu = 2$ to L do :
      — For all i in layer L_ν do :
         — Compute $\text{net}_i = \sum_{j=0;w_{ij} \text{ exists}}^{N} w_{ij} a_j$.
         — Set $a_i = f(\text{net}_i)$, where f is the activation function.
3. **Output** :
   — Provide output $g_i(x; w) = a_i$, for $N - O + 1 \leq i \leq N$.

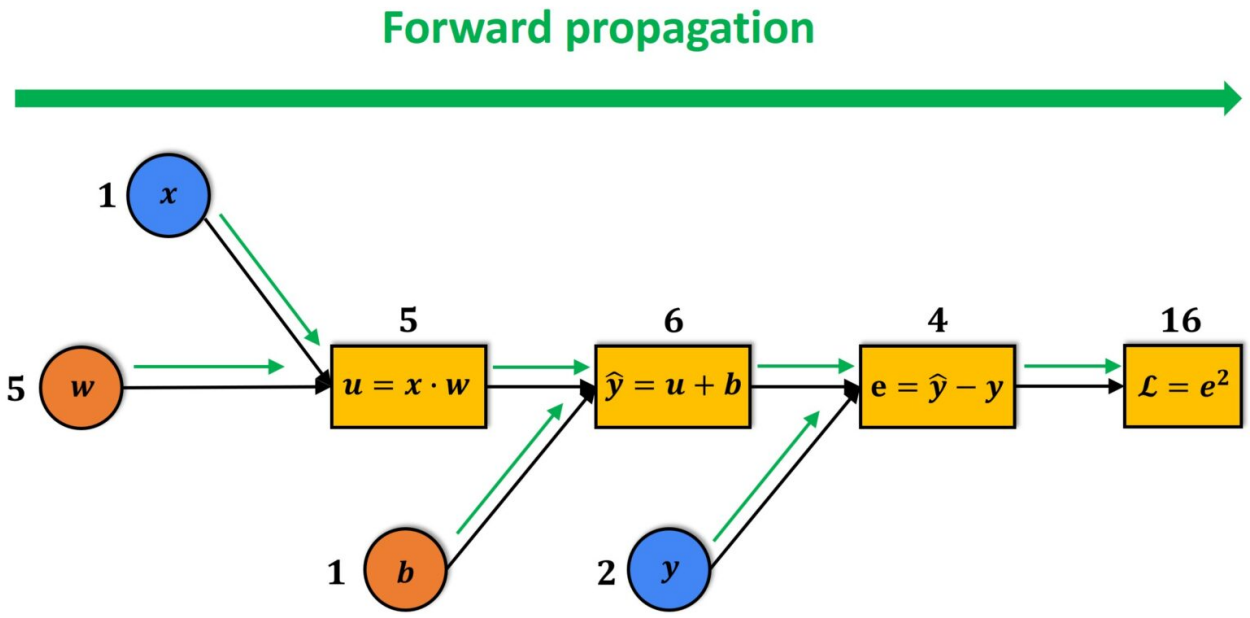


FIGURE 4.5 – Forward Pass.

Source : datahacker.rs

4.1.6 Backpropagation

Backpropagation is a technique employed for computing the gradients of neural network parameters. In essence, this method involves traversing the network in reverse, moving from the output layer to the input layer, utilizing the chain rule from calculus. Throughout this process, the algorithm systematically retains intermediate variables, namely partial derivatives, necessary for the comprehensive calculation of gradients concerning specific parameters.

1. Calculate the gradients of the objective function (loss and regularization term) :

$$\frac{\partial J}{\partial \Sigma} = 1 \tag{4.21}$$

$$\frac{\partial J}{\partial s} = 1 \tag{4.22}$$

2. Compute the gradient of the objective function (output layer) :

$$\frac{\partial J}{\partial \omega} = \frac{\partial J}{\partial \Sigma}\frac{\partial \Sigma}{\partial \omega} = \frac{\partial \Sigma}{\partial \omega} \tag{4.23}$$

3. Calculate the gradients of the regularization term :

$$\begin{cases} \frac{\partial s}{\partial W^1} &= \lambda W^1 \\ \frac{\partial \Sigma}{\partial W^2} &= \lambda W^2 \end{cases} \tag{4.24}$$

4. Calculate the gradient :

$$\begin{aligned} \frac{\partial J}{\partial W^2} &= \frac{\partial J}{\partial \omega}\frac{\partial \omega}{\partial W^2} + \frac{\partial J}{\partial s}\frac{\partial s}{\partial W^2} \\ &= \frac{\partial J}{\partial \omega}{A^1}^T + \lambda W^2 \end{aligned} \tag{4.25}$$

5. The gradient with respect to the hidden layer output :

$$\begin{aligned} \frac{\partial J}{\partial A^1} &= \frac{\partial J}{\partial \omega}\frac{\partial \omega}{\partial A^1} \\ &= {W^2}^T\frac{\partial J}{\partial \omega} \end{aligned} \tag{4.26}$$

6.

$$\begin{aligned} \frac{\partial J}{\partial Z} &= \frac{\partial J}{\partial A^1}\frac{\partial A^1}{\partial Z} \\ &= \frac{\partial J}{\partial A^1}\phi'(Z) \end{aligned} \tag{4.27}$$

7. Finally, the gradient :

$$\begin{aligned} \frac{\partial J}{\partial W^1} &= \frac{\partial J}{\partial Z}\frac{\partial Z}{\partial W^1} + \frac{\partial J}{\partial s}\frac{\partial s}{\partial W^1} \\ &= \frac{\partial J}{\partial Z}X^T + \lambda W^1 \end{aligned} \tag{4.28}$$

Algorithm 5.2 Backward Pass of an MLP

The steps are as follows hochreiter2014theoretical :

1. **Initialization** :
   (a) Provide activations a_i from the forward pass and the label y.
   (b) For $i = N - O + 1$ to N do :
      — Calculate $\delta_i = \frac{\partial L(y,x,w)}{\partial a_i} f'(\text{net}_i)$.
      — For all j in layer L_{L-1} do :
         — Update weight $\Delta w_{ij} = -\eta \delta_i a_j$.
2. **Backward Pass** :
   (a) For $\nu = L - 1$ to 2 do :
      — For all i in layer L_ν do :
         — Calculate $\delta_i = f'(\text{net}_i) \sum_k \delta_k w_{ki}$.
         — For all j in layer $L_{\nu-1}$ do :
            — Update weight $\Delta w_{ij} = -\eta \delta_i a_j$.

4.2 Deep Learning

Deep Architectures (see Figure 4.7) refer to computational models that are composed of multiple layers of interconnected processing units, commonly known as neurons or nodes. These architectures are characterized by their depth, meaning they have a significant number of hidden layers between the input and output layers. Each layer in a deep architecture performs a specific transformation on the data, extracting increasingly abstract and complex features as the data propagates through the network (heaton2015artificial).

Deep architectures are a hallmark of deep learning, a subfield of machine learning. They are particularly well-suited for handling high-dimensional data and complex patterns, making them ideal for tasks such as image and speech recognition, natural language processing, and game playing. The depth of these networks allows them to learn hierarchical

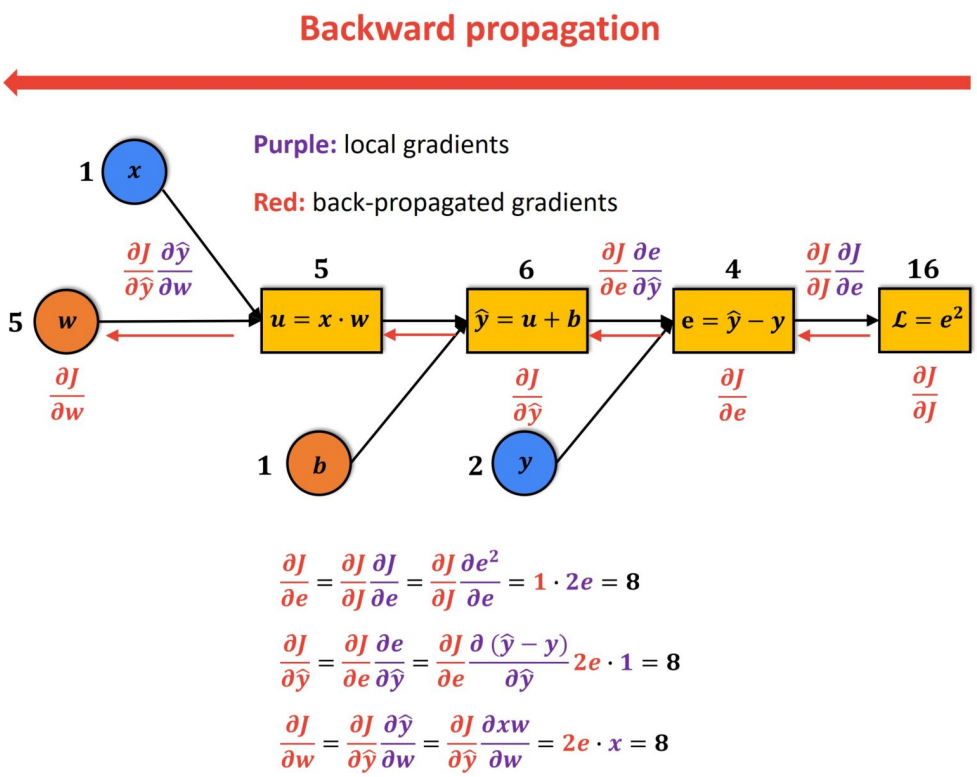


FIGURE 4.6 – Backpropagation

representations, where each successive layer captures more sophisticated features or representations of the input data.

The most common types of deep architectures include Convolutional Neural Networks (CNNs), which are widely used in image and video processing; Recurrent Neural Networks (RNNs) and their variants like Long Short-Term Memory (LSTM) networks, which are effective in modeling sequential data; and Deep Belief Networks (DBNs) and Autoencoders, which are used for unsupervised learning and feature extraction.

The training of deep architectures typically involves sophisticated optimization techniques and regularization methods to address challenges such as overfitting, vanishing gradients, and computational efficiency. Despite these challenges, the ability of deep architectures to automatically discover relevant features from raw data has led to significant breakthroughs in various AI applications.

4.2.1 Convolutional Neural Networks (CNNs)

Convolutional Neural Networks, commonly known as CNNs or ConvNets, are a class of deep learning architectures particularly well-suited for processing grid-like data structures, such as images. Inspired by the visual cortex of animals, CNNs utilize layers of neurons that

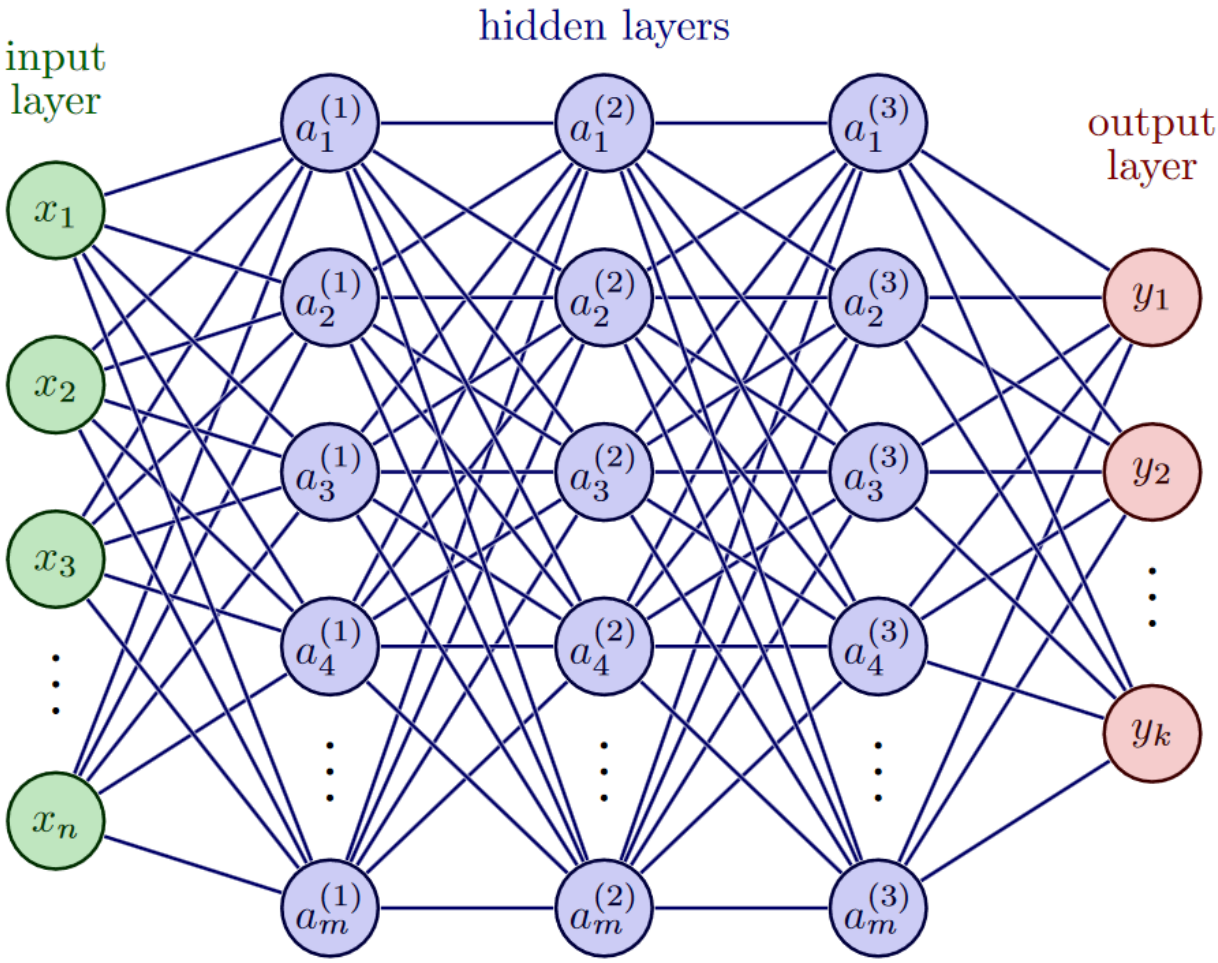


FIGURE 4.7 – Deep Architectures

respond to overlapping regions of the visual field (see Figure 4.8).

Core Components of CNNs

1. **Convolutional Layers** : The convolutional layer is the core building block of a CNN. It consists of a set of learnable filters (or kernels) that slide over the input data. The operation can be mathematically described by :

$$(f * x)(i, j) = \sum_m \sum_n x(m, n)w(i - m, j - n) \tag{4.29}$$

   where x is the input, w is the filter (kernel), and (i, j) are the coordinates of the output feature map. Each filter learns to detect specific features such as edges, textures, or colors.

2. **Activation Functions** : After convolution, the feature map is passed through an activation function to introduce non-linearity into the model. The most commonly used

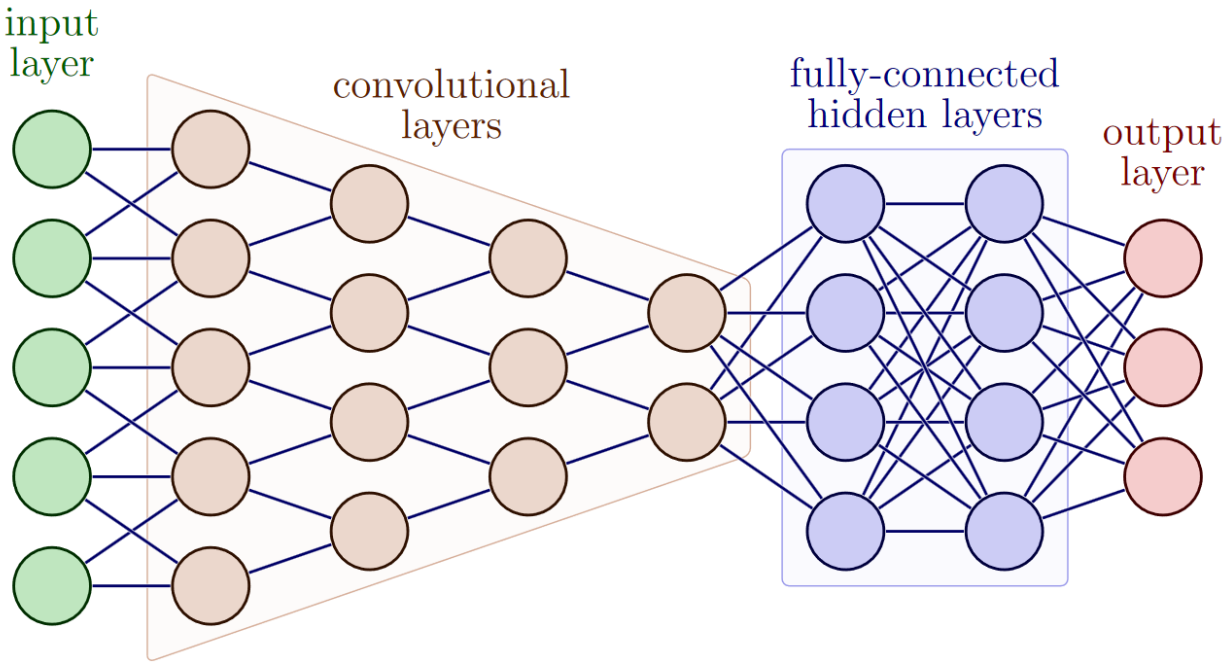


FIGURE 4.8 – CNN Architectures

activation function is the Rectified Linear Unit (ReLU), defined as :

$$f(x) = \max(0, x) \tag{4.30}$$

This function helps the network to learn complex patterns.

3. **Pooling Layers** : Pooling layers reduce the spatial dimensions (width and height) of the feature maps while retaining the most important information. A common operation is max pooling, which is defined as :

$$y(i, j) = \max_{m,n} x(i + m, j + n) \tag{4.31}$$

where $y(i, j)$ is the pooled output and $x(i, j)$ is the input feature map. This operation helps to achieve spatial invariance.

4. **Fully Connected Layers** : In the final stages of a CNN, fully connected layers are used to combine the features learned by the convolutional layers across the entire image. The output from the previous layers is flattened into a vector and fed into the fully connected layers, leading to the final classification output. The output layer uses a softmax activation function for multi-class classification, which can be represented as :

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_j e^{z_j}} \tag{4.32}$$

where z represents the inputs to the output layer.

Operation of CNNs

The operation of a CNN involves a forward pass where an input image is passed through the network layers, undergoing convolution, activation, and pooling operations, followed by fully connected layers. The final layer produces class scores or probabilities, from which the network's prediction is derived.
During training, a loss function, such as cross-entropy loss, measures the discrepancy between the predicted labels and the true labels. The weights of the network are then adjusted using backpropagation and optimization algorithms like Stochastic Gradient Descent (SGD) to minimize this loss function.

CNNs are particularly powerful in tasks involving visual data due to their ability to learn spatial hierarchies of features. They are widely used in applications such as image and video recognition, object detection, and facial recognition. CNNs have also been adapted for other domains like speech recognition and natural language processing, showcasing their versatility and robustness.

4.2.2 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a class of artificial neural networks designed to recognize patterns in sequences of data, such as time series, natural language, and more. Unlike feedforward neural networks, RNNs have connections that form directed cycles, allowing them to maintain a 'memory' of previous inputs (see Figure 4.9). This makes RNNs particularly well-suited for tasks where context and sequence order are important.

Core Components of RNNs

1. **Hidden States and Recurrence** : The fundamental feature of an RNN is its hidden state, which captures information from the sequence of inputs. The hidden state at time step t, denoted as h_t, is a function of the input at the current time step x_t and the hidden state from the previous time step h_{t-1}. This relationship can be expressed as :

$$h_t = f(W_{xh}x_t + W_{hh}h_{t-1} + b_h) \tag{4.33}$$

where :

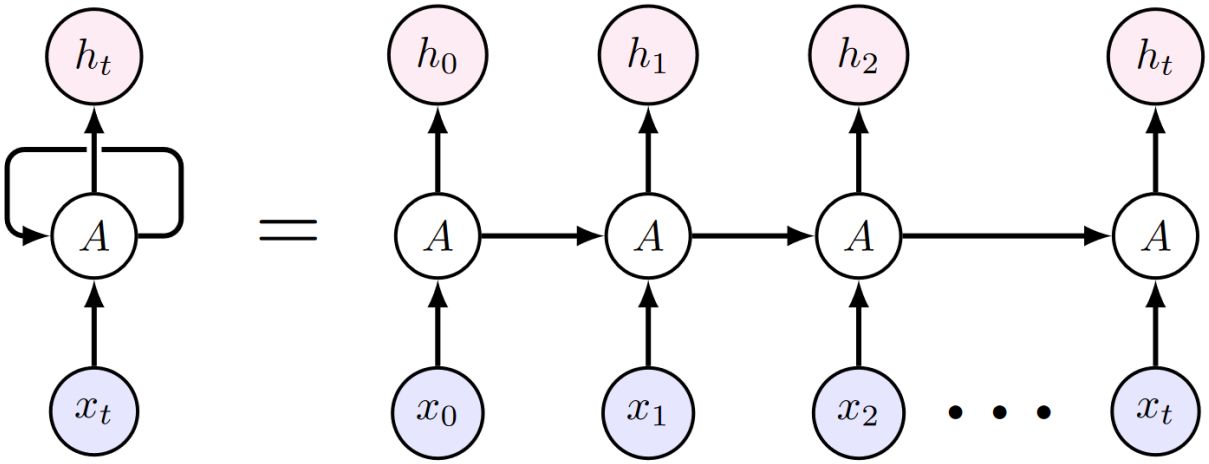


FIGURE 4.9 – Recurrent Neural Network (RNN)

— W_{xh} is the weight matrix for the input to hidden state,

— W_{hh} is the weight matrix for the hidden state to hidden state,

— b_h is the bias term, and

— f is the activation function, typically a non-linear function like *tanh* or ReLU.

2. **Output Layer** : The output at each time step t, denoted as y_t, is typically computed using the hidden state h_t. The output can be expressed as :

$$y_t = g(W_{hy}h_t + b_y) \tag{4.34}$$

where :

— W_{hy} is the weight matrix from the hidden state to the output,

— b_y is the bias term for the output, and

— g is an activation function, often a softmax function for classification tasks.

Training RNNs

RNNs are trained using the backpropagation through time (BPTT) algorithm, a variant of the backpropagation algorithm adapted for handling sequential data. The BPTT algorithm involves unfolding the network through time and applying backpropagation to calculate gradients. The gradients are then used to update the network's weights to minimize the loss function.

Applications of RNNs

RNNs have a wide range of applications, particularly in areas involving sequential data. They are used in language modeling, speech recognition, machine translation, and time series prediction, among others. However, standard RNNs can suffer from issues such as vanishing and exploding gradients, which limit their ability to learn long-term dependencies in sequences.

Recurrent Neural Networks (RNNs) are a powerful tool for processing sequential data, thanks to their ability to maintain information over time. By leveraging hidden states and recurrent connections, RNNs can capture the temporal dynamics of sequences, making them suitable for a wide range of applications in machine learning.

4.2.3 Long Short-Term Memory (LSTM) Networks

Long Short-Term Memory (LSTM) networks (see Figure 4.10) are a type of Recurrent Neural Network (RNN) specifically designed to overcome the limitations of traditional RNNs, such as the vanishing and exploding gradient problems. LSTMs achieve this by introducing a more sophisticated memory cell structure, allowing them to maintain and manipulate information over longer periods. This makes LSTMs particularly effective for tasks that require learning long-term dependencies, such as language modeling and time series forecasting.

Core Components of LSTM Networks

LSTM networks consist of a series of cells, each containing three main components : a cell state, an input gate, a forget gate, and an output gate. These components work together to control the flow of information within the network.

1. **Cell State** : The cell state, denoted as C_t, serves as a memory that carries information across different time steps. It can be modified by the gates to retain or forget information as needed. The cell state can be updated using the following equation :

$$C_t = f_t \odot C_{t-1} + i_t \odot \tilde{C}_t \tag{4.35}$$

where :

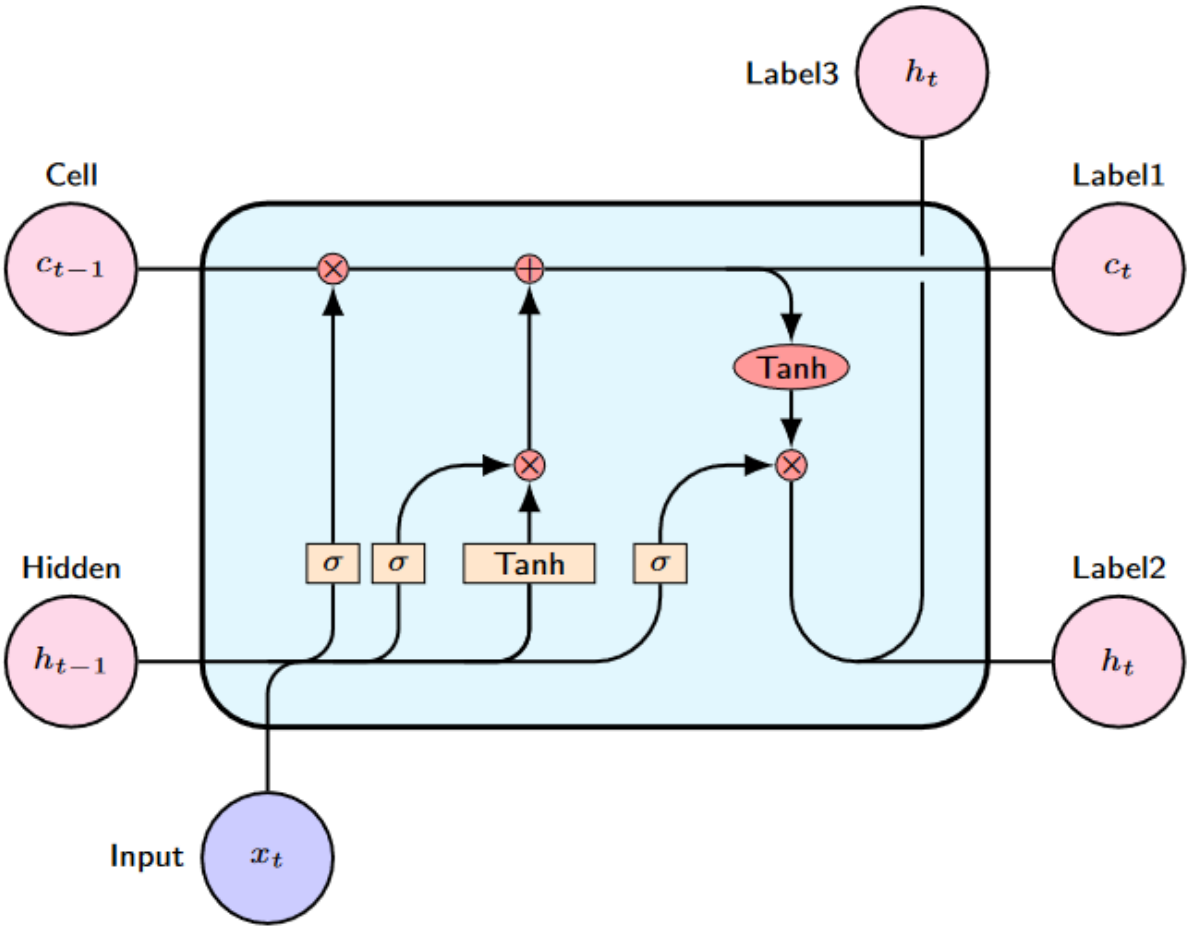


FIGURE 4.10 – Long Short-Term Memory networks

— f_t is the forget gate activation,

— i_t is the input gate activation,

— $\tilde{C}_t$ is the candidate cell state,

— $\odot$ represents element-wise multiplication.

2. **Gates in LSTM** : The three gates in an LSTM—input gate, forget gate, and output gate—are crucial for controlling the flow of information.

   (a) **Forget Gate** : The forget gate decides which information from the previous cell state should be discarded. It is defined as :

   $$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{4.36}$$

   where σ is the sigmoid function, W_f is the weight matrix, h_{t-1} is the previous hidden state, x_t is the input at the current time step, and b_f is the bias term.

   (b) **Input Gate** : The input gate controls how much of the new information should

be added to the cell state. It is given by :

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{4.37}$$

The candidate cell state $\tilde{C}_t$ is computed using :

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C) \tag{4.38}$$

(c) **Output Gate** : The output gate determines the output of the LSTM cell based on the cell state. It is defined as :

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \tag{4.39}$$

The hidden state h_t is then calculated as :

$$h_t = o_t \odot \tanh(C_t) \tag{4.40}$$

Training LSTM Networks

LSTM networks are trained using backpropagation through time (BPTT), similar to standard RNNs. However, due to the gating mechanisms, LSTMs can learn to retain relevant information and forget irrelevant information over longer sequences, making them more robust in handling long-term dependencies.

Applications of LSTM Networks

LSTMs have been widely used in various applications, particularly where sequence prediction and long-term context are crucial. Notable applications include natural language processing (NLP) tasks like language translation and sentiment analysis, speech recognition, time series prediction, and anomaly detection in sequential data.

Long Short-Term Memory (LSTM) networks represent a significant advancement in the field of recurrent neural networks, providing a robust solution to the challenges posed by long-term dependencies. Through the use of gates and cell states, LSTMs can effectively manage the flow of information, making them indispensable for a wide range of applications that involve sequential data.

CHAPTER 5

Evaluation of Machine Learning Algorithms

In machine learning, there are numerous algorithms available for both regression and classification tasks. Given a specific problem, multiple algorithms may be applicable, making it essential to evaluate their effectiveness. This chapter focuses on the evaluation of machine learning algorithms, exploring methods to assess the performance of both regression and classification models. We will also discuss how to compare the performance of different algorithms to select the most suitable one for practical applications. These evaluation techniques are crucial for ensuring that we choose the right model that meets the desired criteria and performs well on the given data.

5.1 Methods of Evaluation

In practical applications of machine learning, whether for classification or regression tasks, it is essential to evaluate the performance of algorithms accurately. Typically, a small subset of data is reserved as a validation set, while the rest is used for training (see Figure 5.1). The model, developed using the training set, is then evaluated on the validation set to assess its accuracy or error metrics. However, relying solely on a single validation set does not provide a complete picture of the model's performance. Moreover, single validation set evaluations are insufficient for making meaningful comparisons between different algorithms. This necessitates the use of multiple validation sets.

When a machine learning model is trained on a dataset, whether it is a classifier or a

regressor, it produces a specific outcome based on the validation set. To account for variations due to randomness in training data, initialization, and other factors, multiple models can be generated using the same algorithm. These models are then tested on various validation sets, producing a range of error measurements. The statistical distribution of these errors provides valuable insights into the expected performance of the algorithm for the given problem and allows for a more comprehensive comparison with other algorithms.

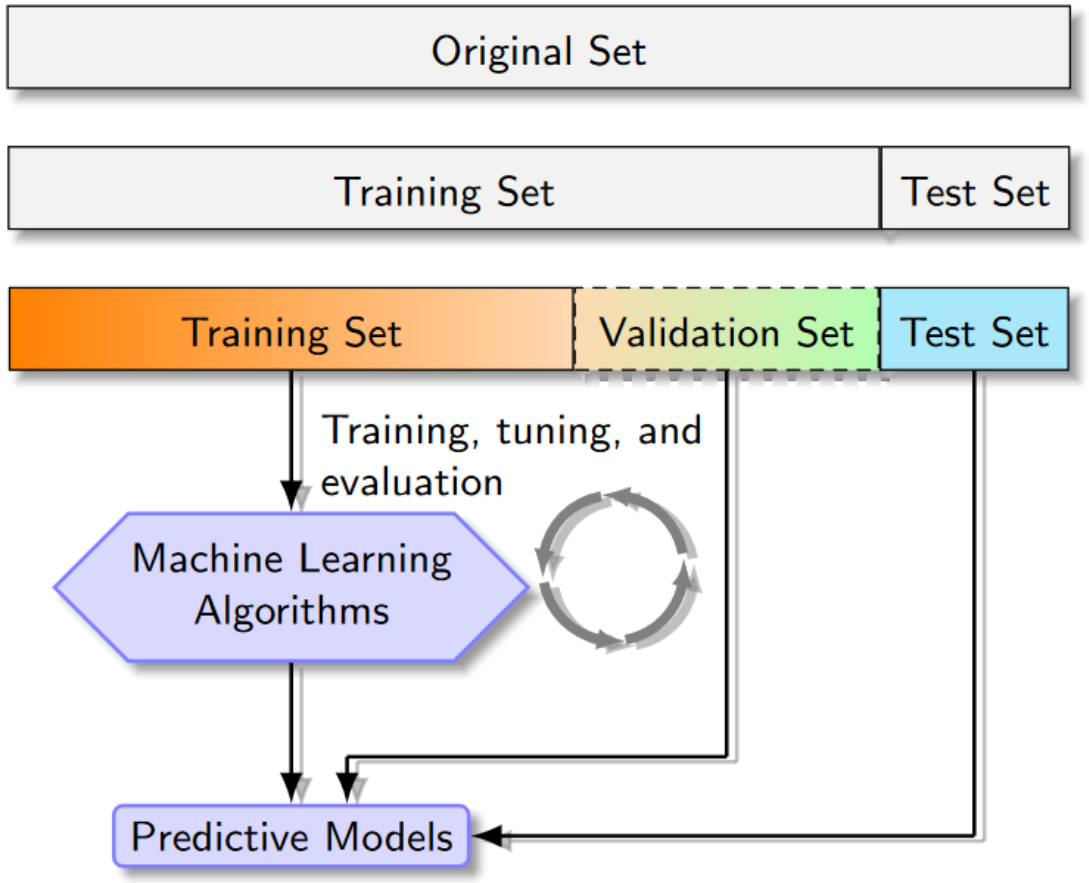


FIGURE 5.1 – Multiple Validation Sets

Cross-validation, particularly k-fold cross-validation, is a commonly used method for generating multiple training-validation sets from a given dataset, providing a more robust assessment of model performance.

Evaluation of machine learning algorithms, be it for classification or regression, should consider factors beyond traditional error metrics, including :

— **Risks associated with errors :** Generalized using loss functions, which may vary significantly depending on the application.

— **Training time and space complexity :** The resources required during the training phase, which can affect scalability.

— **Testing time and space complexity :** The efficiency of the model during deployment and prediction.

— **Interpretability :** The ability of the model to provide insights that can be understood and verified by experts.

— **Ease of implementation :** The complexity involved in programming and deploying the algorithm, which may influence the choice of model in practical scenarios.

5.2 Cross-Validation

Cross-validation is a vital technique in machine learning for evaluating the performance of predictive models. It involves partitioning the original dataset into different sets for training and validation multiple times to ensure a robust assessment. This section explores various cross-validation methods, including K-fold cross-validation, leave-one-out cross-validation, and bootstrapping.

5.2.1 K-Fold Cross-Validation

In K-fold cross-validation, the dataset X is divided randomly into K equal-sized parts, X_i, where $i = 1, \ldots, K$. For each iteration, one of the K parts is used as the validation set V_i, and the remaining $K - 1$ parts are combined to form the training set T_i. This process is repeated K times, ensuring that each part is used once as the validation set (see Figure 5.2). The configuration for each iteration is as follows :

$$
\begin{aligned}
V_1 &= X_1, & T_1 &= X_2 \cup X_3 \cup \ldots \cup X_K \\
V_2 &= X_2, & T_2 &= X_1 \cup X_3 \cup \ldots \cup X_K \\
&\vdots & &\vdots \\
V_K &= X_K, & T_K &= X_1 \cup X_2 \cup \ldots \cup X_{K-1}
\end{aligned}
$$

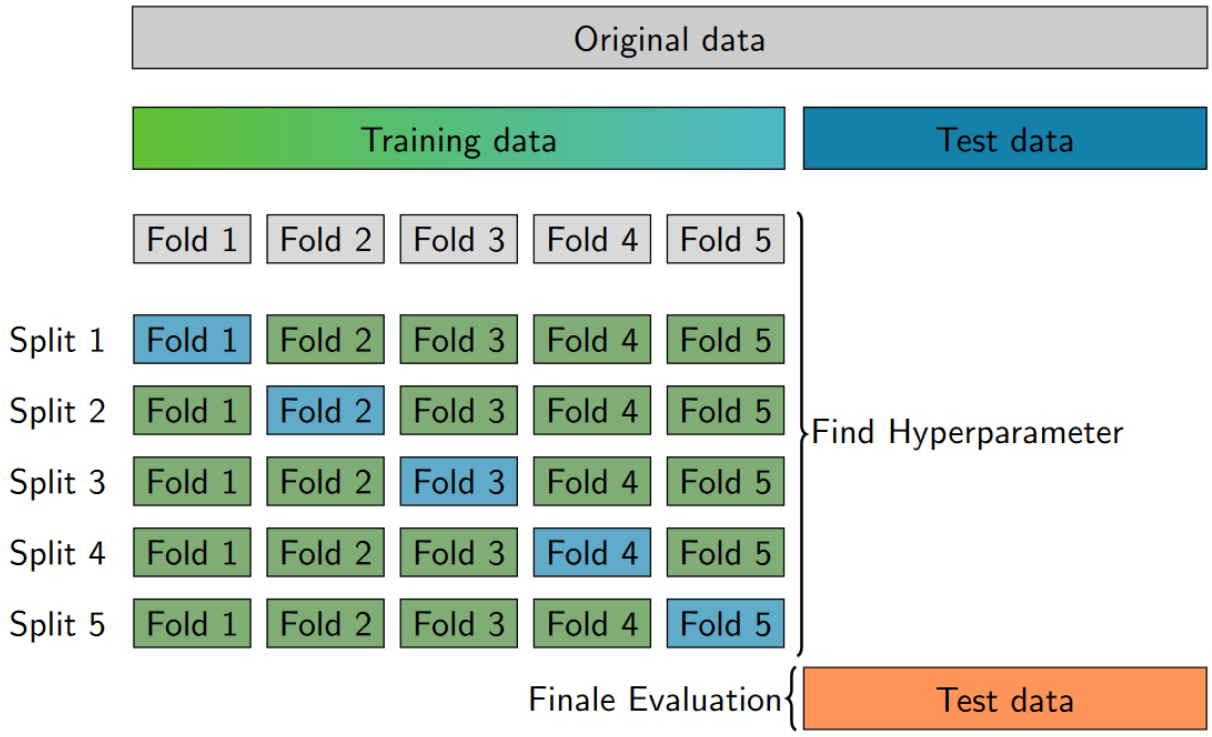


FIGURE 5.2 – K-Fold Cross-Validation

1. A key limitation of K-fold cross-validation is the potential small size of the validation sets, especially when K is large. This may not accurately reflect the model's performance on unseen data.
2. There is a significant overlap between training sets, as each training set shares $K - 2$ parts with others.
3. Typically, K is chosen as 10 or 30. Increasing K provides more robust estimations by increasing the percentage of training data, but it also reduces the size of each validation set and increases computational costs, as the model must be trained K times.

5.2.2 Leave-One-Out Cross-Validation (LOOCV)

Leave-one-out cross-validation is an extreme case of K-fold cross-validation, where K equals the number of instances N in the dataset. In this method, each instance in the dataset is used as a single-instance validation set, with the remaining $N - 1$ instances

forming the training set. This results in N separate evaluations, each with a unique validation instance. LOOCV is particularly useful in applications such as medical diagnosis, where labeled data is scarce.

5.2.3 5 × 2 Cross-Validation

The 5 × 2 cross-validation method involves dividing the dataset X into two equal parts $X_1^{(1)}$ and $X_1^{(2)}$. The model is first trained on $X_1^{(1)}$ and validated on $X_1^{(2)}$, and then the roles are reversed, with $X_1^{(2)}$ as the training set and $X_1^{(1)}$ as the validation set. This procedure is repeated five times with different random splits, resulting in ten pairs of training and validation sets. The pairs are as follows :

$$
\begin{aligned}
T_1 &= X_1^{(1)}, & V_1 &= X_1^{(2)} \\
T_2 &= X_1^{(2)}, & V_2 &= X_1^{(1)} \\
&\vdots & &\vdots \\
T_9 &= X_5^{(1)}, & V_9 &= X_5^{(2)} \\
T_{10} &= X_5^{(2)}, & V_{10} &= X_5^{(1)}
\end{aligned}
$$

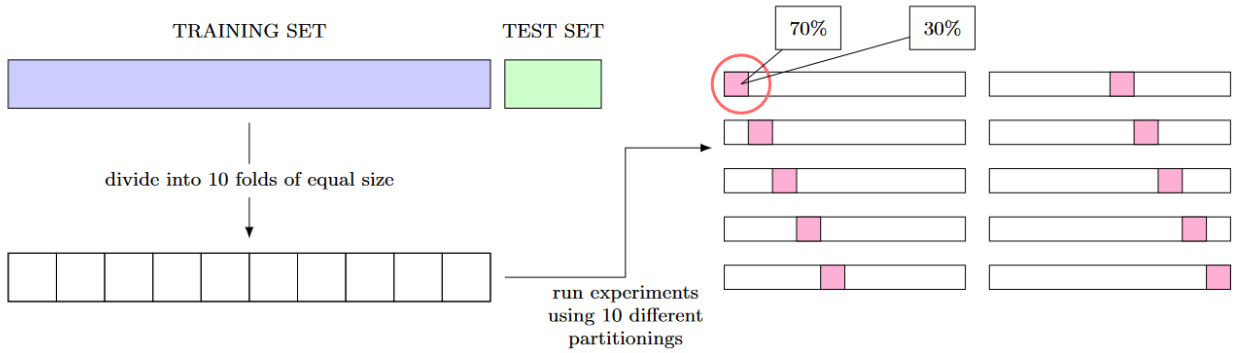


FIGURE 5.3 – N × 2 Cross-Validation

5.2.4 Bootstrapping

Bootstrapping is a statistical resampling technique where datasets are sampled with replacement. In the context of machine learning, bootstrapping involves creating multiple new training datasets by randomly sampling from the original dataset, with some data points potentially being sampled multiple times (see Figure 5.4). The corresponding test datasets are formed from the instances not included in the training sets.

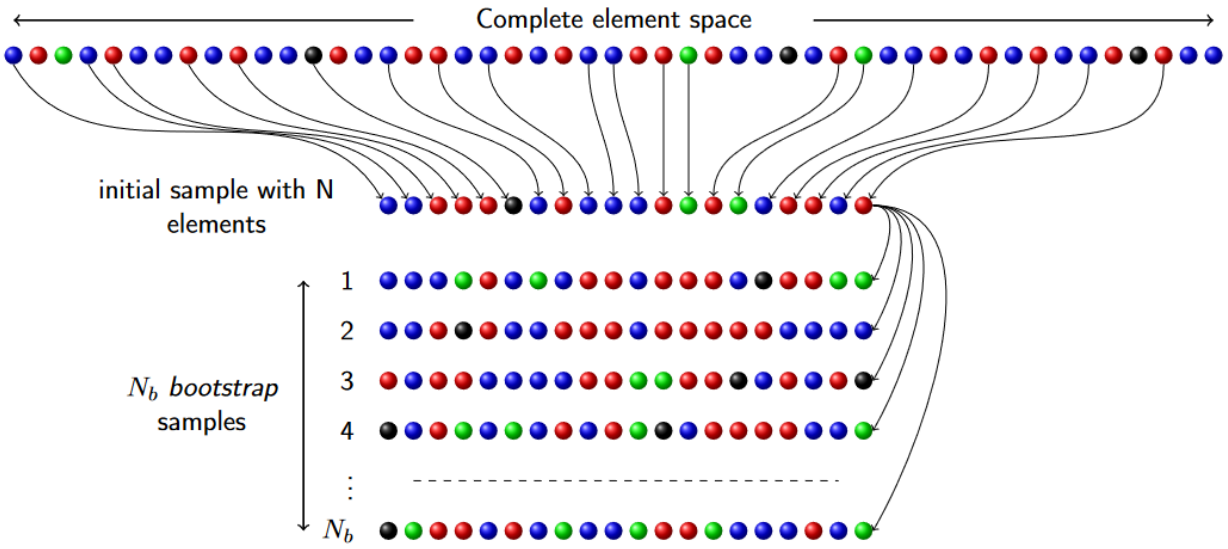


FIGURE 5.4 – Bootstrapping.

Source : texample.net

For example, consider an urn with five labeled balls : A, B, C, D, and E. To select samples containing two balls, we might :

1. Draw two balls (e.g., A and E), record the labels, and return them to the urn.
2. Draw two more balls (e.g., C and E), record the labels, and return them to the urn.

This process, known as sampling with replacement, is repeated as needed to generate multiple samples.

In machine learning, bootstrapping helps estimate model performance by providing several randomly selected training and test datasets. The performance measures from these datasets are averaged to give a more robust estimate. Bootstrapping is particularly useful for small datasets, as it maximizes the use of available data.

5.3 Measuring Error

Accurately measuring the error of a machine learning model is crucial for evaluating its performance. Different metrics are used depending on whether the model is a regression or

a classification model. This section covers various error measures for both types of models.

5.3.1 Error Measures for Regression Models

In regression, the goal is to predict a continuous output. The following metrics are commonly used to measure the accuracy of regression models :

Mean Absolute Error (MAE)

The Mean Absolute Error (MAE) measures the average magnitude of the errors in a set of predictions, without considering their direction. It is calculated as :

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i| \tag{5.1}$$

where y_i are the actual values, $\hat{y}_i$ are the predicted values, and n is the number of observations.

Mean Squared Error (MSE)

The Mean Squared Error (MSE) measures the average of the squares of the errors. It gives more weight to larger errors, which can be useful for identifying large outliers. The formula is :

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{5.2}$$

Root Mean Squared Error (RMSE)

The Root Mean Squared Error (RMSE) is the square root of the MSE. It is in the same units as the target variable, which can make interpretation easier. It is calculated as :

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2} \tag{5.3}$$

Mean Absolute Percentage Error (MAPE)

The Mean Absolute Percentage Error (MAPE) expresses the accuracy as a percentage of the error. It is calculated as :

$$\text{MAPE} = \frac{100\%}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \tag{5.4}$$

However, MAPE can be sensitive to very small actual values, potentially leading to large percentage errors.

Coefficient of Determination (R^2)

The R^2 score, or Coefficient of Determination, indicates the proportion of the variance in the dependent variable that is predictable from the independent variables. It is calculated as :

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{5.5}$$

where $\bar{y}$ is the mean of the actual values y_i. An R^2 value of 1 indicates perfect prediction, while an R^2 value of 0 indicates that the model does not explain any of the variability in the target variable.

Theil's U Statistic

Theil's U Statistic is a relative measure of accuracy that compares the predictive accuracy of a forecasting model to that of a naïve model, which simply uses the last observed value as the forecast for the next period. It is defined as :

$$U = \frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}}{\sqrt{\frac{1}{n}\sum_{i=1}^{n} y_i^2} + \sqrt{\frac{1}{n}\sum_{i=1}^{n} \hat{y}_i^2}} \tag{5.6}$$

A Theil's U value less than 1 indicates that the model has better predictive accuracy than the naïve model, whereas a value greater than 1 indicates worse predictive performance. The table below 5.1 compares several commonly used regression error metrics in terms of their description, interpretability, handling of errors, and unit of measure.

5.3.2 Error Measures for Classification Models

Classification models predict categorical outcomes. The following metrics are commonly used to assess the performance of these models :

Metric	Description	Interpretability	Handling of Errors	Unit of Measure
Mean Absolute Error	Average absolute difference between forecast and actual values	Easy to interpret	Considers magnitude only	Original data unit
Root Mean Squared Error	Square root of MSE	Easy to interpret	Gives more weight to larger errors	Original data unit
Mean Absolute Percentage Error	Average percentage difference between forecast and actual values	Easy to interpret	Considers relative difference	Percentage
Theil's U statistic	Relative measure comparing forecast model with a benchmark	Easy to interpret	Compares performance to benchmark	Ratio

TABLE 5.1 – Metrics comparison [Bil24]

Confusion Matrix

A Confusion Matrix provides a detailed breakdown of the classification results (see Figure 5.5). It consists of the following components :

— **True Positives (TP)** : The number of correctly predicted positive cases.

— **False Positives (FP)** : The number of incorrectly predicted positive cases.

— **True Negatives (TN)** : The number of correctly predicted negative cases.

— **False Negatives (FN)** : The number of incorrectly predicted negative cases.

Precision and Recall

Precision and Recall are key metrics derived from the confusion matrix :

— **Precision** : The ratio of correctly predicted positive observations to the total predic-

		Condition Phase (worst case)		
		Condition Positive/ Shaded	Condition Negative/ Unshaded	**Actual**
Testing Phase (best case)	Test Positive/ Shaded	True positive shaded T_p *(Correct)*	False positive shaded F_p *(Incorrect)*	Precision/Positive Predictive Value (PPV) $\frac{T_p}{T_p+F_p} \times 100\%$
	Test Negative/ Unshaded	False negative unshaded F_n *(Incorrect)*	True negative unshaded T_n *(Correct)*	Negative Predictive Value (NPV) $\frac{T_n}{T_n+F_n} \times 100\%$
		Sensitivity/Recall Rate (RR) $\frac{T_p}{T_p+F_n} \times 100\%$	Specificity Rate (SR) $\frac{T_n}{T_n+F_p} \times 100\%$	

FIGURE 5.5 – Confusion Matrix

ted positives. It is given by :

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.7}$$

— **Recall** : (Sensitivity or True Positive Rate) : The ratio of correctly predicted positive observations to all observations in the actual class. It is calculated as :

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.8}$$

F1 Score

The F1 Score is the harmonic mean of Precision and Recall, providing a balance between the two. It is calculated as :

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.9}$$

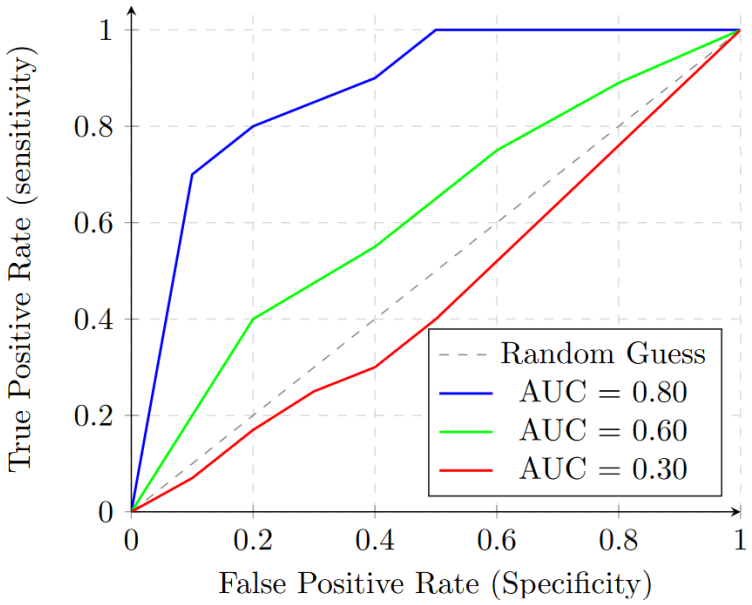


FIGURE 5.6 – ROC curve

Receiver Operating Characteristic (ROC) Curve and AUC

The ROC curve plots the True Positive Rate (Recall) against the False Positive Rate (FPR) (see Figure 5.6), which is defined as :

$$\text{FPR} = \frac{FP}{FP + TN} \tag{5.10}$$

The area under the ROC curve (AUC) provides a single scalar value to compare models. An AUC of 1 represents perfect classification, while an AUC of 0.5 indicates no discriminative ability.

Other Measures of Performance

Other metrics include :

— **Accuracy** : The ratio of correctly predicted instances (both positive and negative) to the total instances.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{5.11}$$

— **Specificity (True Negative Rate)** : The ratio of correctly predicted negative observations to all actual negatives.

$$\text{Specificity} = \frac{TN}{TN + FP} \tag{5.12}$$

Appendix

5.4 Publicly-available datasets related to agriculture

No.	Organization/Dataset	Description of dataset	Source
1	Image-Net Dataset	Images of various plants (trees, vegetables, flowers)	http://image-net.org/explore?wnid=n07707451
2	ImageNet Large Scale Visual Recognition Challenge (ILSVRC)	Images that allow object localization and detection	http://image-net.org/challenges/LSVRC/2017/#det
3	University of Arcansas, Plants Dataset	Herbicide injury image database	https://plants.uaex.edu/herbicide/ http://www.uaex.edu/yard-garden/resource-library/diseases/
4	EPFL, Plant Village Dataset	Images of various crops and their diseases	https://www.plantvillage.org/en/crops
5	Leafsnap Dataset	Leaves from 185 tree species from the Northeastern United States.	http://leafsnap.com/dataset/
6	LifeCLEF Dataset	Identity, geographic distribution and uses of plants	http://www.imageclef.org/2014/lifeclef/plant
7	PASCAL Visual Object Classes Dataset	Images of various animals (birds, cats, cows, dogs, horses, sheep etc.)	http://host.robots.ox.ac.uk/pascal/VOC/
8	Africa Soil Information Service (AFSIS) dataset	Continent-wide digital soil maps for sub-Saharan Africa	http://africasoils.net/services/data/
9	UC Merced Land Use Dataset	A 21 class land use image dataset	http://vision.ucmerced.edu/datasets/landuse.html

No.	Organization/Dataset	Description of dataset	Source
10	MalayaKew Dataset	Scan-like images of leaves from 44 species classes.	http://web.fsktm.um.edu.my/~cschan/downloads_MKLeaf_dataset.html
11	Crop/Weed Field Image Dataset	Field images, vegetation segmentation masks and crop/weed plant type annotations.	https://github.com/cwfid/dataset https://pdfs.semanticscholar.org/58a0/9b1351ddb447e6abded7233a4794d538155.pdf
12	University of Bonn Photogrammetry, IGG	Sugar beets dataset for plant classification as well as localization and mapping.	http://www.ipb.uni-bonn.de/data/
13	Flavia leaf dataset	Leaf images of 32 plants.	http://flavia.sourceforge.net/
14	Syngenta Crop Challenge 2017	2,267 of corn hybrids in 2,122 of locations between 2008 and 2016, together with weather and soil conditions	https://www.ideaconnection.com/syngenta-crop-challenge/challenge.php

TABLE 5.2 – Datasets related to agriculture [KP18]

Bibliographie

AGGARWAL, Charu C (2014). « An Introduction to Data Classification. » In : *Data classification : algorithms and applications* 125.3, p. 142.

AGGARWAL, Charu C et Chandan K REDDY (2014). « Data clustering ». In : *Algorithms and applications. Chapman&Hall/CRC Data mining and Knowledge Discovery series, Londra*.

ALSHARIF, Mohammed H et al. (2020). « Machine learning algorithms for smart data analysis in internet of things environment : taxonomies and research trends ». In : *Symmetry* 12.1, p. 88.

APOLO-APOLO, Orly Enrique et al. (2020). « A mixed data-based deep neural network to estimate leaf area index in wheat breeding trials ». In : *Agronomy* 10.2, p. 175.

BELLIDO-JIMÉNEZ, Juan Antonio et al. (2022). « AgroML : An open-source repository to forecast reference evapotranspiration in different geo-climatic conditions using machine learning and transformer-based models ». In : *Agronomy* 12.3, p. 656.

BILEL, Ammouri (2024). « Forecasting agricultures security indices : Evidence from transformers method ». In : *Journal of Forecasting*.

BISHOP, Christopher M (1995). *Neural networks for pattern recognition*. Oxford university press.

BISHOP, Christopher M et Nasser M NASRABADI (2006). *Pattern recognition and machine learning*. T. 4. 4. Springer.

ELAVARASAN, Dhivya et PM Durairaj VINCENT (2020). « Crop yield prediction using deep reinforcement learning model for sustainable agrarian applications ». In : *IEEE access* 8, p. 86886-86901.

GARCIA-PEDRERO, Angel et al. (2019). « Deep learning for automatic outlining agricultural parcels : Exploiting the land parcel identification system ». In : *IEEE access* 7, p. 158223-158236.

GARCÍA-VÁZQUEZ, Fabián et al. (2023). « Prediction of internal temperature in greenhouses using the supervised learning techniques : Linear and support vector regressions ». In : *Applied Sciences* 13.14, p. 8531.

GHOSH, Dibyendu et al. (2022). « Application of machine learning in understanding plant virus pathogenesis : trends and perspectives on emergence, diagnosis, host-virus interplay and management ». In : *Virology Journal* 19.1, p. 42.

GOMES, Jacó C et Díbio L BORGES (2022). « Insect pest image recognition : A few-shot machine learning approach including maturity stages classification ». In : *Agronomy* 12.8, p. 1733.

HAN, Jiawei, Jian PEI et Hanghang TONG (2022). *Data mining : concepts and techniques*. Morgan kaufmann.

HEATON, Jeff (2015). « Artificial Intelligence for Humans, Volume 3 : Neural Networks and Deep Learning ». In : *Heaton Research Inc, Chesterfield, ABD* 30, p. 55.

HERTZMANN, Aaron, David FLEET et Marcus BRUBAKER (2014). « Machine learning and data mining lecture notes ». In : *Department of Computer and Mathematical Sciences, University of Toronto Scarborough*.

HOCHREITER, Sepp (2013). « Basic methods of data analysis ». In : *Institute of Bioinformatics, Johannes Kepler University Linz, statistics. Austria : Johannes Kepler University Linz*.

— (2014). « Theoretical concepts of machine learning ». In : *Lecture Notes] Linz, AUT : Institute of Bioinformatics, Johannes Kepler University Linz. Available at :< http ://www. bioinf. jku. at/teaching/current/ss vl tcml/ML theoretical. pdf>[Accessed 26/07/2016]*.

— (s. d.). « Bioinformatics III ». In : ().

KAMILARIS, Andreas et Francesc X PRENAFETA-BOLDÚ (2018). « Deep learning in agriculture : A survey ». In : *Computers and electronics in agriculture* 147, p. 70-90.

KRISHNACHANDRAN, VN (s. d.). « Lecture Notes in ». In : ().

LIANG, Yun-Chia et al. (2020). « Machine learning-based prediction of air quality ». In : *applied sciences* 10.24, p. 9151.

MURPHY, Kevin P (2012). *Machine learning : a probabilistic perspective*. MIT press.

SHAFAGH-KOLVANAGH, Jalil et al. (2022). « Machine learning-assisted analysis for agronomic dataset of 49 Balangu (Lallemantia iberica L.) ecotypes from different regions of Iran ». In : *Scientific Reports* 12.1, p. 19237.

WIDYAWATI, Dewi, Amaliah FARADIBAH et Poetri Lestari Lokapitasari BELLUANO (2023). « Comparison Analysis of Classification Model Performance in Lung Cancer Prediction Using Decision Tree, Naive Bayes, and Support Vector Machine ». In : *Indonesian Journal of Data and Science* 4.2, p. 78-86.