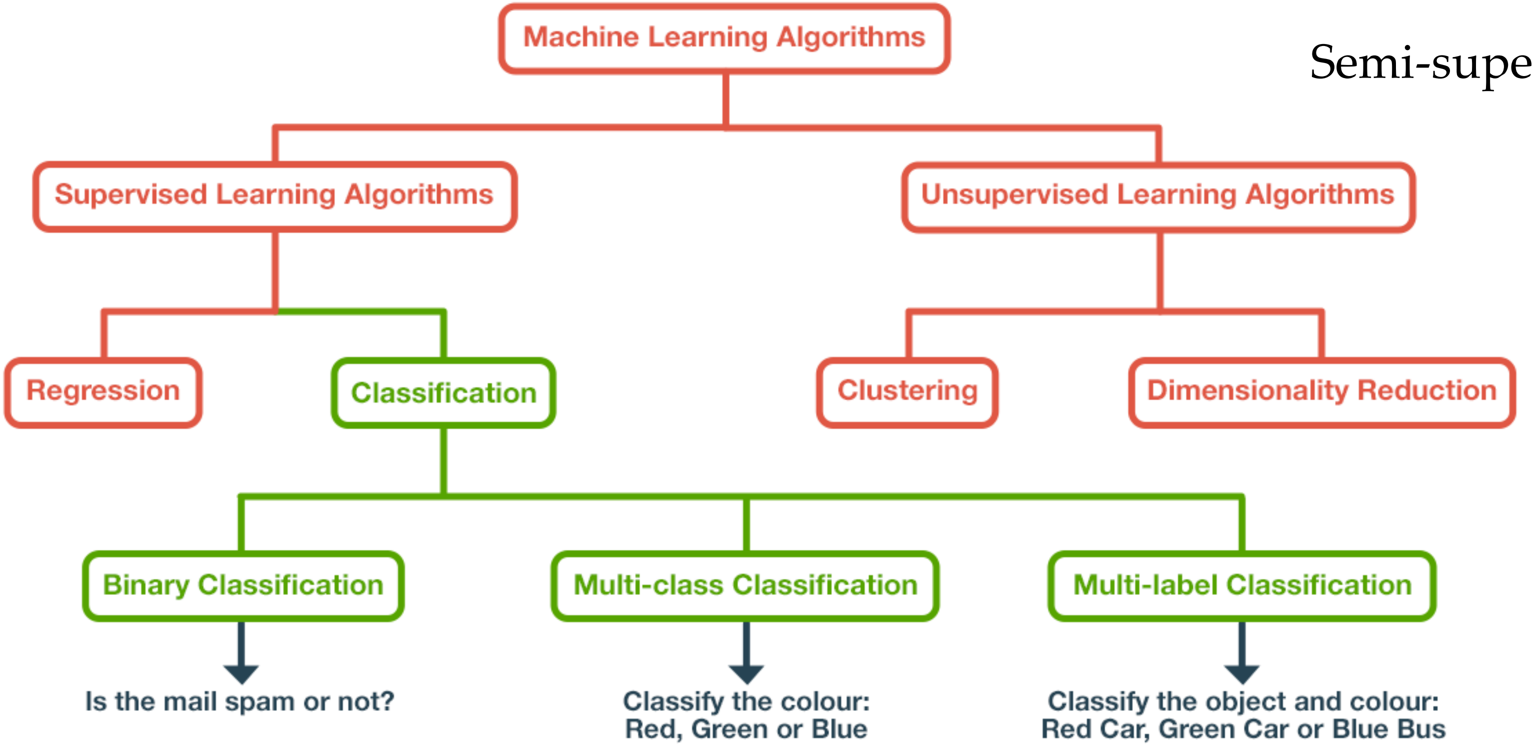
AI & ML Workshop

Introduction to ML Algorithms

Muthukumar

Classification of ML Algorithms



Machine Learning Algorithms

Supervised Learning Algorithms

Unsupervised Learning Algorithms

Regression

Classification

Clustering

Dimensionality Reduction

Binary Classification

Multi-class Classification

Multi-label Classification

Is the mail spam or not?

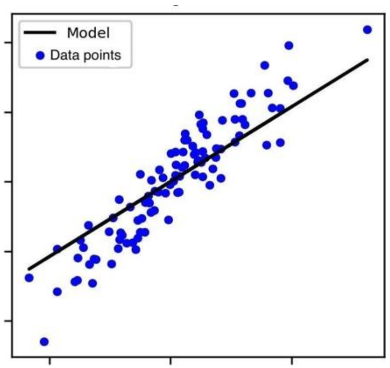
Classify the colour:
Red, Green or Blue

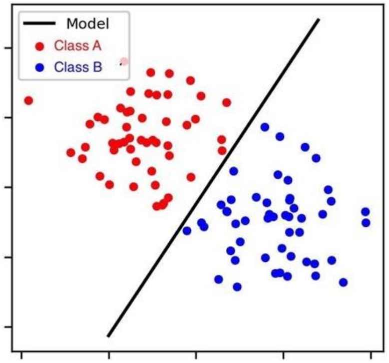
Classify the object and colour:
Red Car, Green Car or Blue Bus

Semi-supervised

Supervised Machine Learning

2 types of problems it tries to solve:





Regression:

- predict numerical (continuous) value
- Linear, Nonlinear Regression

Classification:

- predict categorical (discrete) value
- Naive Bayes Classifier, Support Vector Machines, Logistic Regression, etc ...

Regression & Classification:

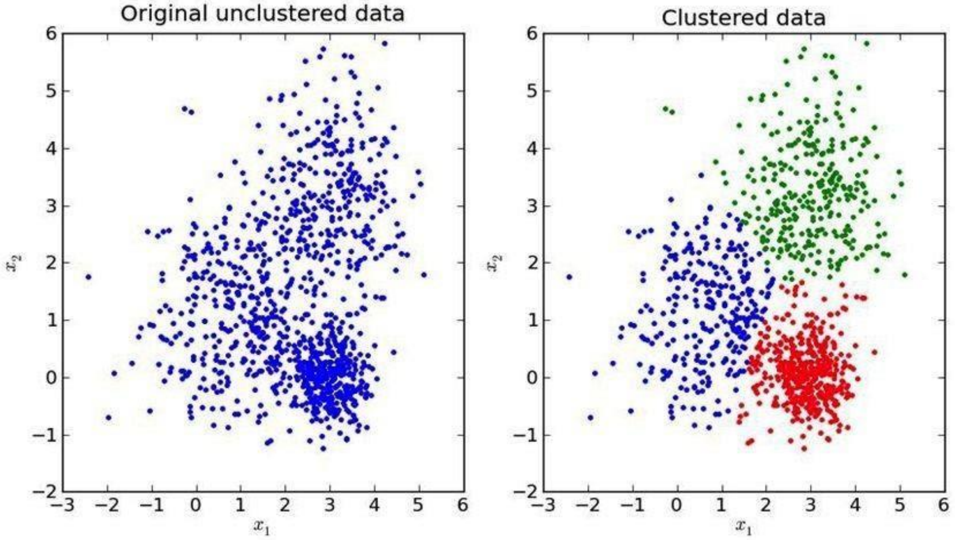
- Decision Tree, Random Forest, k-NN, Neural Networks, etc ... can solve both problems

Unsupervised Machine Learning

Training data: Unlabeled data

Training:

- extract features and patterns from data itself
- clustering: these features used to label and classify the data into clusters
- Example: k-Means clustering, ...



Semi-supervised Machine Learning

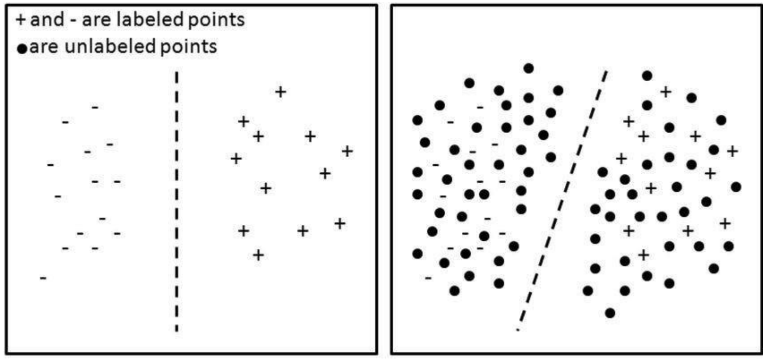
Hybrid learning - between supervised and unsupervised

Solves the problem of having not enough labeled data to train a supervised learning algorithm

Training data: small labeled and large unlabeled data set

Highlights:

- Train model with labeled data
- Trained model used to predict
- labels for unlabeled data => pseudo-labeled data
- Retrain model with both
- pseudo- and labeled data sets



Tools:

- Semi-Supervised Learning in PyTorch - PyTorch implementations of various SSL techniques like pseudo-labeling, consistency regularization, and more.
- Semi-Supervised Active Learning (SEAL) - Active learning-based framework for SSL from Microsoft Research.
- Semi-Supervised VAE (SS-VAE) - Library for semi-supervised learning with variational autoencoders.
- Deep Semi-Supervised Embedding (DS3L) - Library focused on graph-based SSL algorithms.
- Semi-Supervised GAN (SGAN) - GAN-based framework for SSL from MIT.

Applications of different MLs

ML Paradigm	Application Domain	Sensor/Signal Input	Example ML Task	Embedded/TinyML Approach
Supervised ML	UAV (Drone Navigation)	IMU (accel + gyro), GPS	Predict UAV orientation or classify flight maneuvers (hover, roll, pitch)	Train a CNN/MLP offline on labeled flight data, deploy on FC to run inference for real-time UAV control
	Human Activity Recognition (HAR)	Wearable IMU	Classify activities: walking, sitting, running, falling	TinyML with decision tree / 1D-CNN trained on labeled IMU datasets (e.g., WISDM)
Unsupervised ML	HAR	IMU	Discover latent motion patterns without labels	Use k-means / PCA on-device to cluster unknown user motion signatures
Semi-Supervised ML	HAR	Wearable IMU	Few labeled activities, many unlabeled	Pseudo-labeling or consistency regularization for HAR adaptation across users
	ESC Motor Health	Current/voltage + IMU	Small labeled dataset of faults, large unlabeled runtime data	Semi-supervised anomaly detection to refine ESC diagnostics during operation

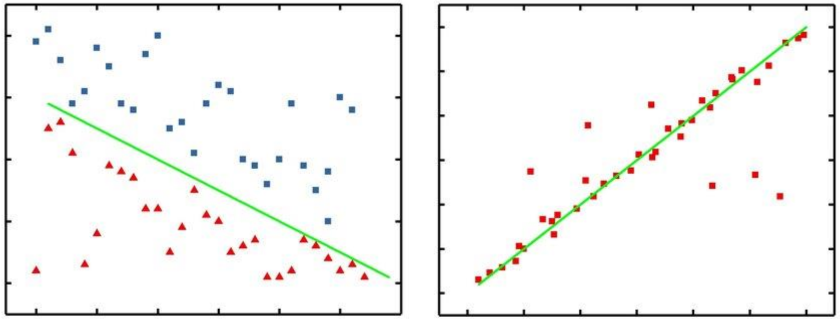
Regression

Regression Models are a class of supervised learning algorithms used for predicting a continuous outcome (target variable) based on one or more independent variables.

Linear regression assumes a linear relationship between the input features and outcome.

Logistic regression is used for binary classification tasks. It models the probability that an instance belongs to a particular class using the logistic function.

Non-linear regression models capture non-linear relationships between input features and the outcome.



(a) Logistic Regression

(b) Linear Regression

Regression for IMU data

Regression models are widely used on Inertial Measurement Unit (IMU) data to predict outcomes like position, orientation, or movement patterns based on sensor readings.

Regression learns the relationship between IMU data and a target variable from a dataset. It is ideal for situations where the system dynamics are complex or unknown.

Sensor error correction: IMU data is prone to noise and drift over time. Regression can be used to model and correct these errors. A time-delayed multiple linear regression model, for instance, has been used to improve the accuracy of low-cost IMU sensors.

Improving analysis of IMU data: A regression-based method has been proposed for Allan variance analysis, which is used to characterize IMU sensor errors.

IMU Roll Prediction

Ground-truth roll angle computed via:

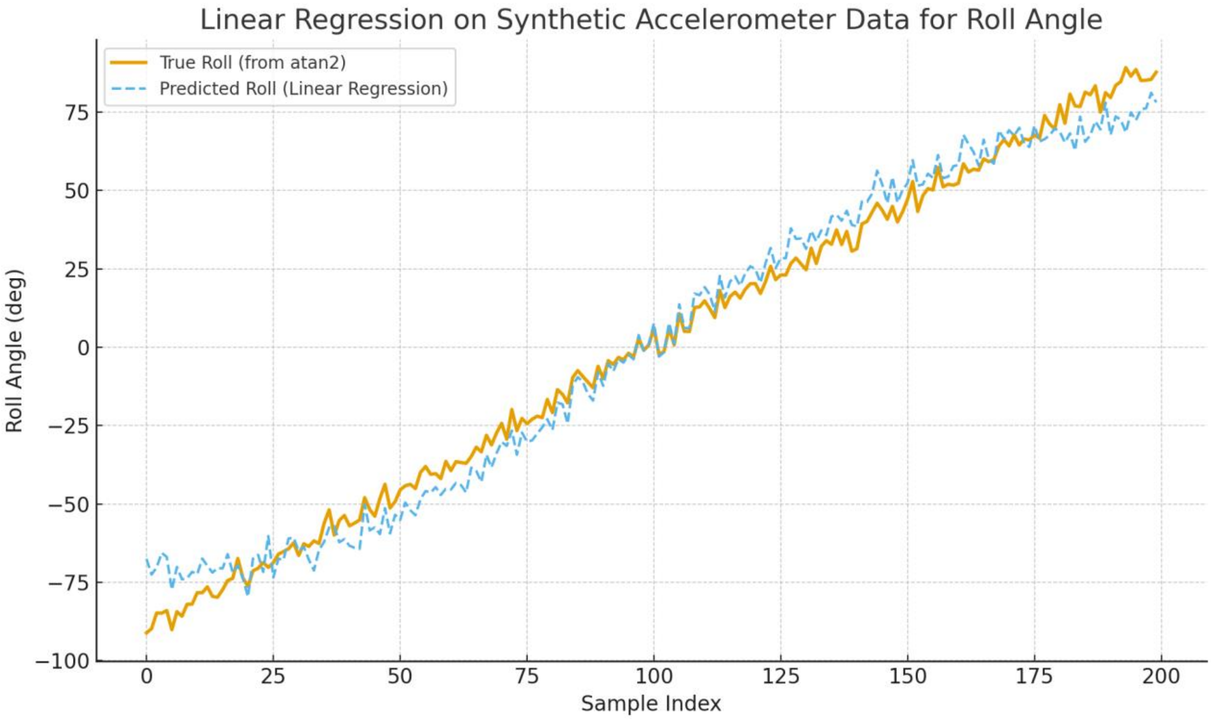
$$\theta_{roll} = \arctan 2\left(a_y, a_z\right)$$

Fitted a **linear regression model**:

$$\hat{\theta}_{roll} = \beta_0 + \beta_1 a_y + \beta_2 a_z$$

regression model is:

$$\hat{\theta}_{roll} \approx -0.44 \; + \; 73.55 \cdot a_y \; + \; 0.60 \cdot a_z$$



Decision Trees

Decision Trees used for both classification and regression tasks.

algorithm builds a tree-like structure by recursively splitting the dataset based on the features that provide the best separation of classes or the best prediction of the outcome/target variable.

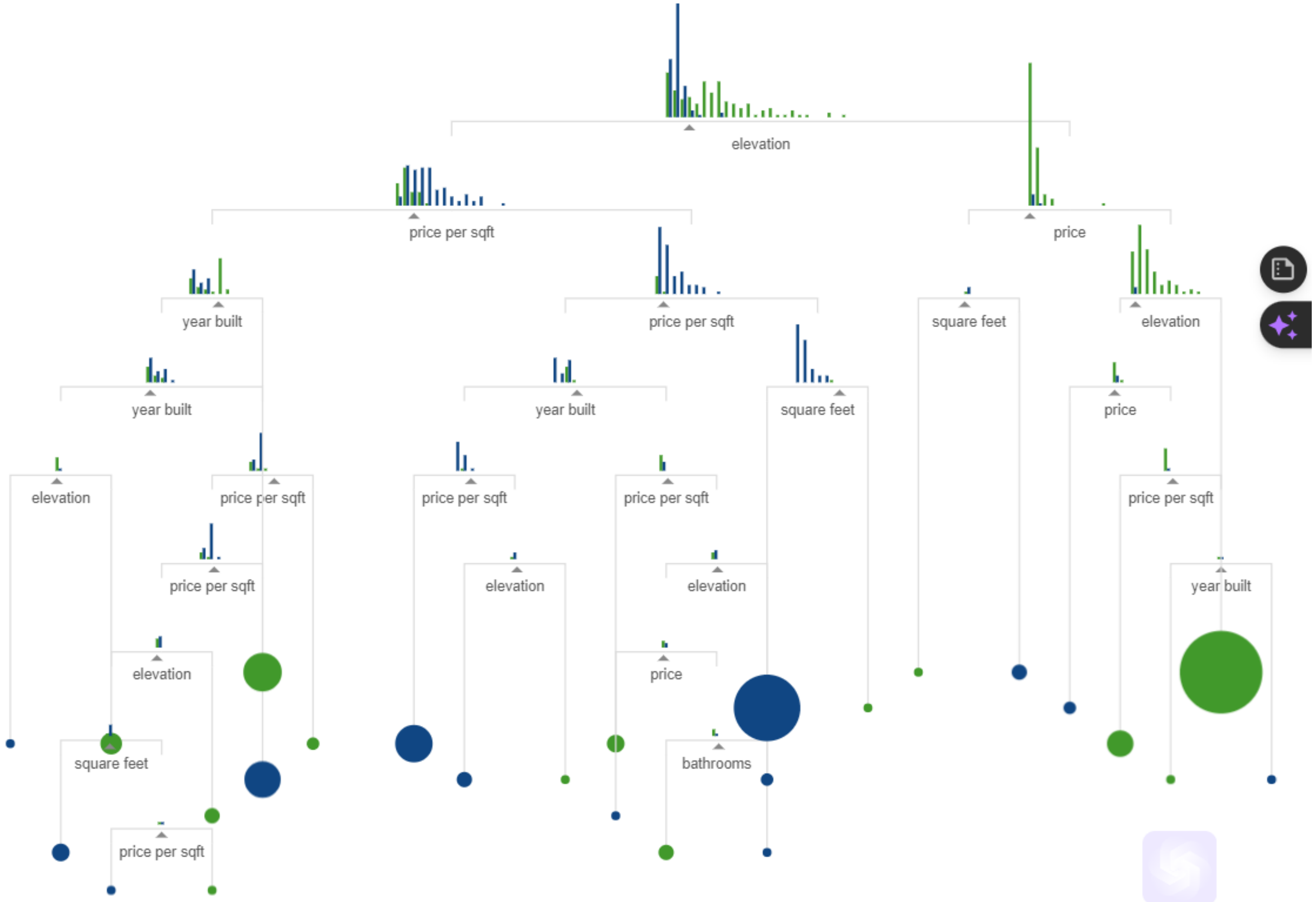
Model:

- Tree-like model with nodes (decision points), branches (possible outcomes), and leaves (final predictions).
- Root node: Represents the entire dataset.
- Internal nodes: Contain decision rules based on feature values.
- Leaf nodes: Represent class labels (for classification) or continuous values (for regression).

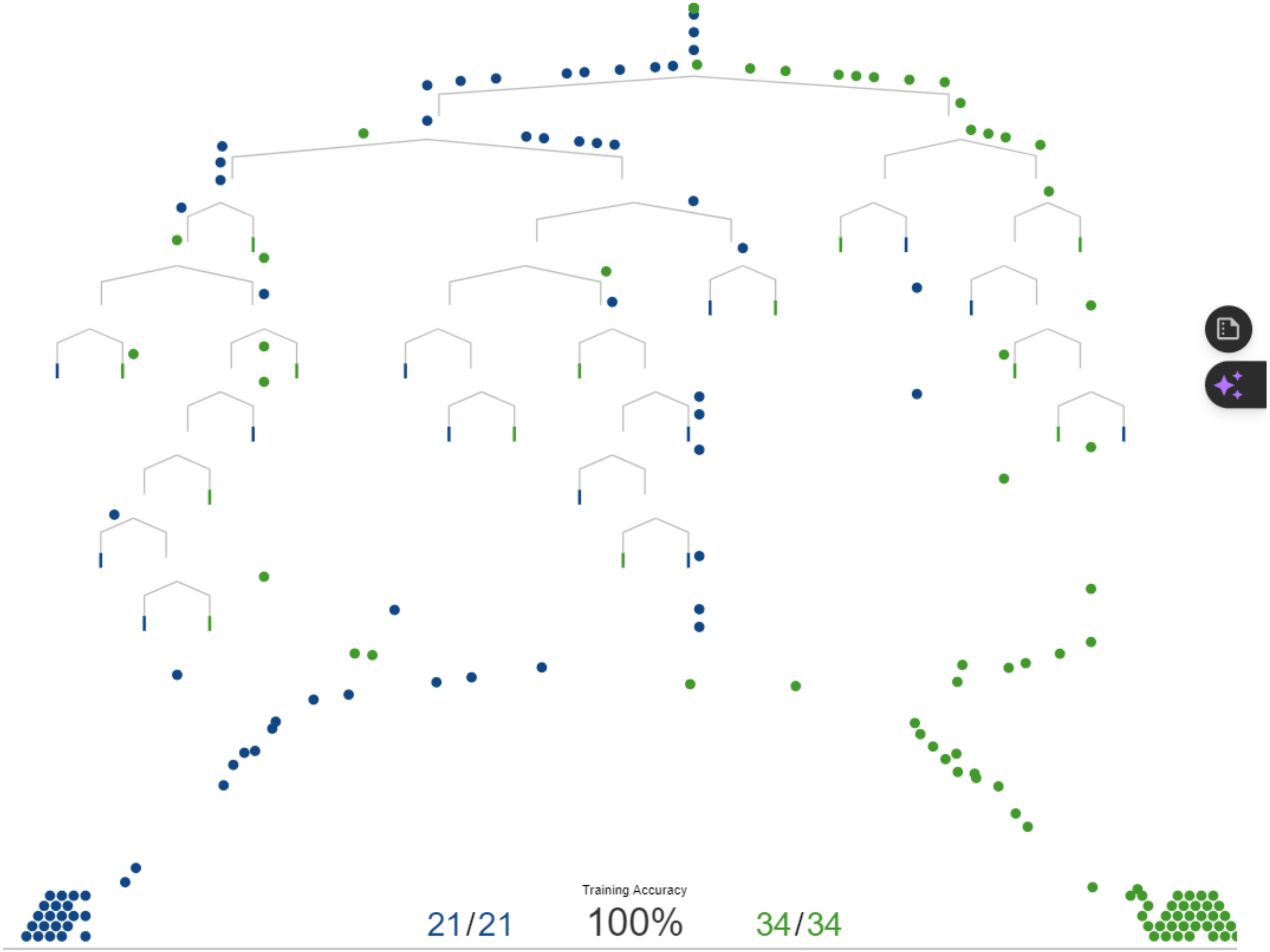
Issues:

Overfitting, and Sensitive to feature scaling: Normalization often required

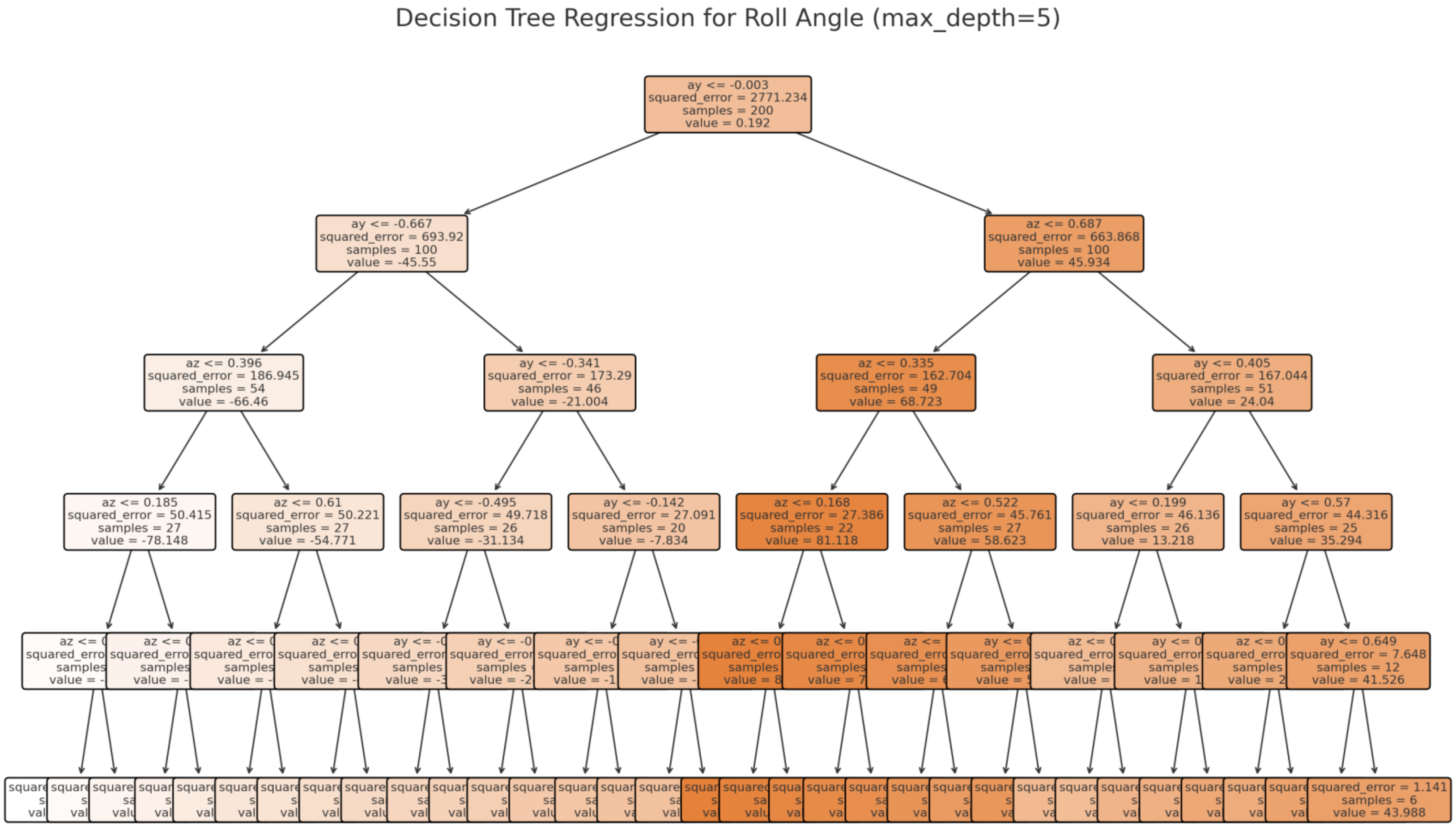
Decision Trees



Decision Trees



Decision Tree for IMU Roll Prediction


Decision Tree Regression for Roll Angle (max_depth=5)
ay <= -0.003 squared_error = 2771.234 samples = 200 value = 0.192
ay <= -0.667 squared_error = 693.92 samples = 100 value = -45.55
az <= 0.687 squared_error = 663.868 samples = 100 value = 45.934
az <= 0.396 squared_error = 186.945 samples = 54 value = -66.46
ay <= -0.341 squared_error = 173.29 samples = 46 value = -21.004
az <= 0.335 squared_error = 162.704 samples = 49 value = 68.723
ay <= 0.405 squared_error = 167.044 samples = 51 value = 24.04
az <= 0.185 squared_error = 50.415 samples = 27 value = -78.148
az <= 0.61 squared_error = 50.221 samples = 27 value = -54.771
ay <= -0.495 squared_error = 49.718 samples = 26 value = -31.134
ay <= -0.142 squared_error = 27.091 samples = 20 value = -7.834
az <= 0.168 squared_error = 27.386 samples = 22 value = 81.118
az <= 0.522 squared_error = 45.761 samples = 27 value = 58.623
ay <= 0.199 squared_error = 46.136 samples = 26 value = 13.218
ay <= 0.57 squared_error = 44.316 samples = 25 value = 35.294
ay <= 0.649 squared_error = 7.648 samples = 12 value = 41.526
squared_error = 1.141 samples = 6 value = 43.988

Random Forests

Random Forests used for both classification and regression tasks.

builds multiple decision trees during training and merges their predictions to improve accuracy and reduce overfitting.

Model:

- Collection of decision trees, each tree is trained independently on a subset of data.
- At each node of a decision tree, a random subset of features is considered for splitting.
- Each tree is trained on a random sample drawn from the original data (bootstrap sample)
- Final decision by majority vote of the trees, robust and avoid overfitting.

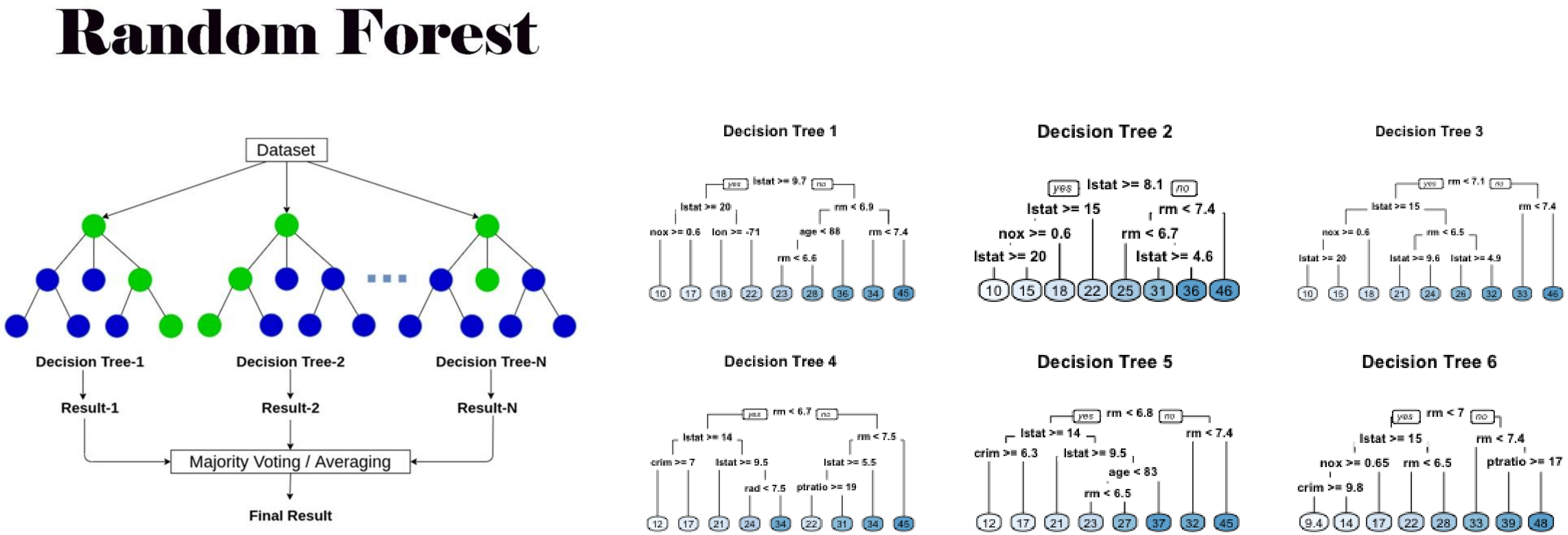
Random Forests

ensemble learning method

constructing a multitude of decision trees during training

outputting the class (for classification) or

mean prediction (for regression) of the individual trees.



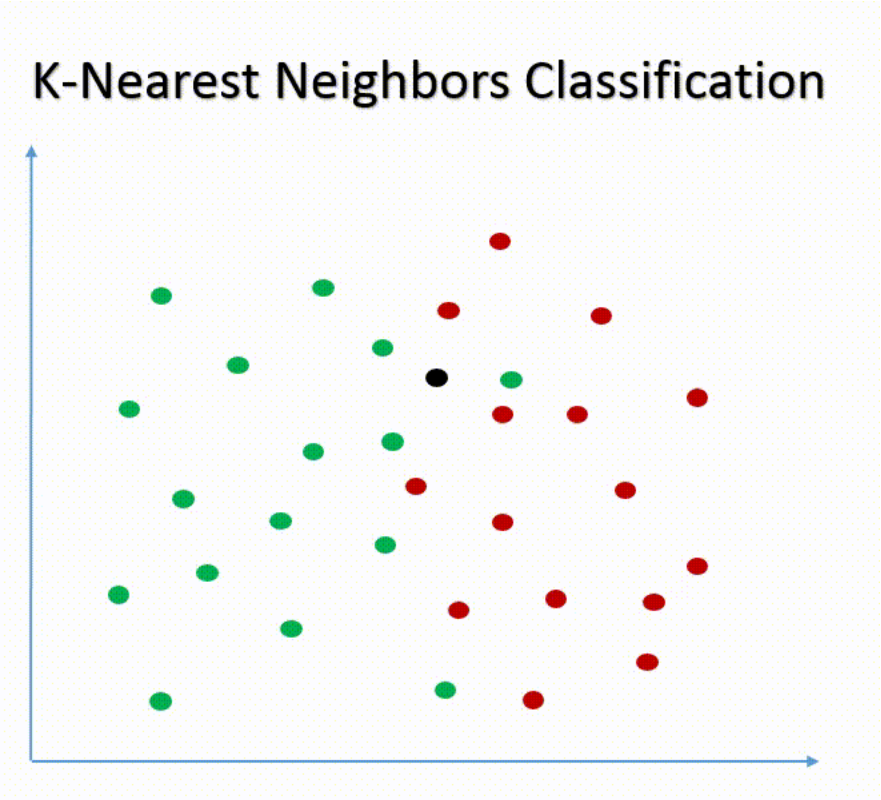
k-Nearest Neighbors (k-NN)

k-NN model - used for both classification and regression tasks.

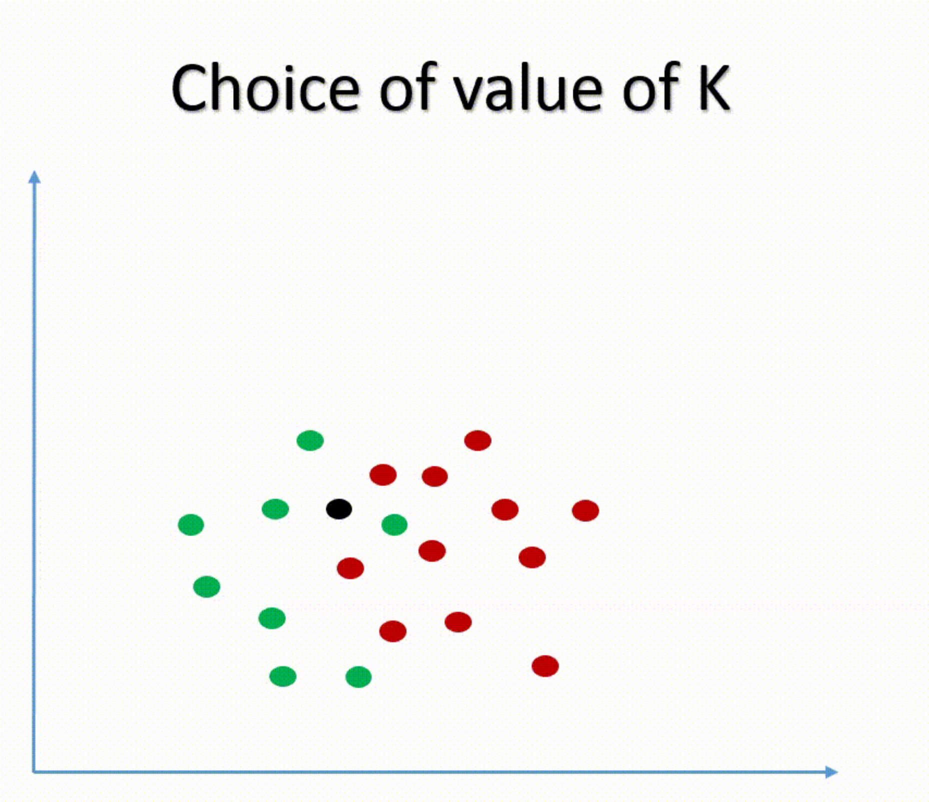
Model:

- finding the k closest training examples to a new input, and making a prediction based on the labels or values of those k neighbors.
- Good for dynamic datasets, as there is no model training phase. Memorizes the entire dataset, not suitable for large datasets.
- Depends on distance metrics (closeness) and k (closest neighbors).

k-Nearest Neighbors (k-NN)



K-Nearest Neighbors Classification



Choice of value of K

Support Vector Machines (SVMs)

SVMs are supervised learning models used for classification and regression tasks.

find a hyperplane that maximizes the margin between classes. Points closest to the hyperplane are support vectors.

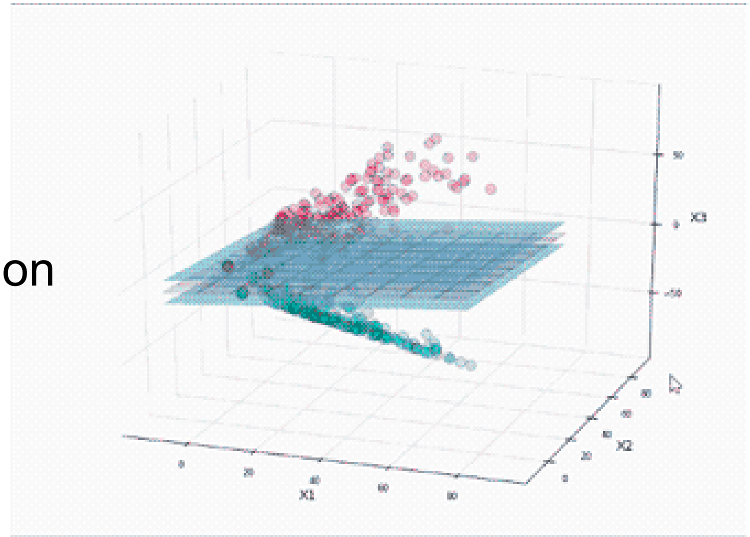
SVR – regression; finds the best-fitting line through data points while ignoring errors within a specified tolerance zone.

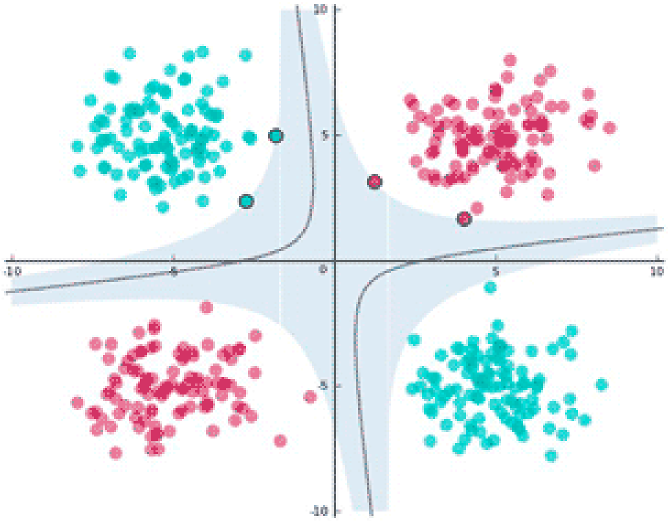
Model:

- Use SVM kernel, such as linear, polynomial or RBF to transform data points and create an optimal decision boundary.
- Hyperparameter Optimization
  - C parameter, how strictly the SVM optimization tries to find the maximum margin hyperplane between classes.
  - gamma controls the flexibility of non-linear SVM models. It determines how far the influence of individual training examples reaches when making predictions.

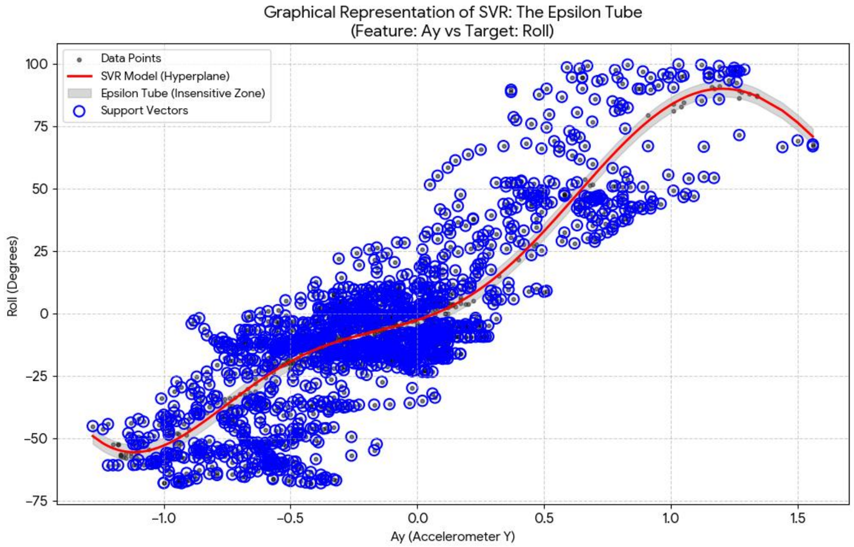
Support Vector Machines (SVMs)

Classification

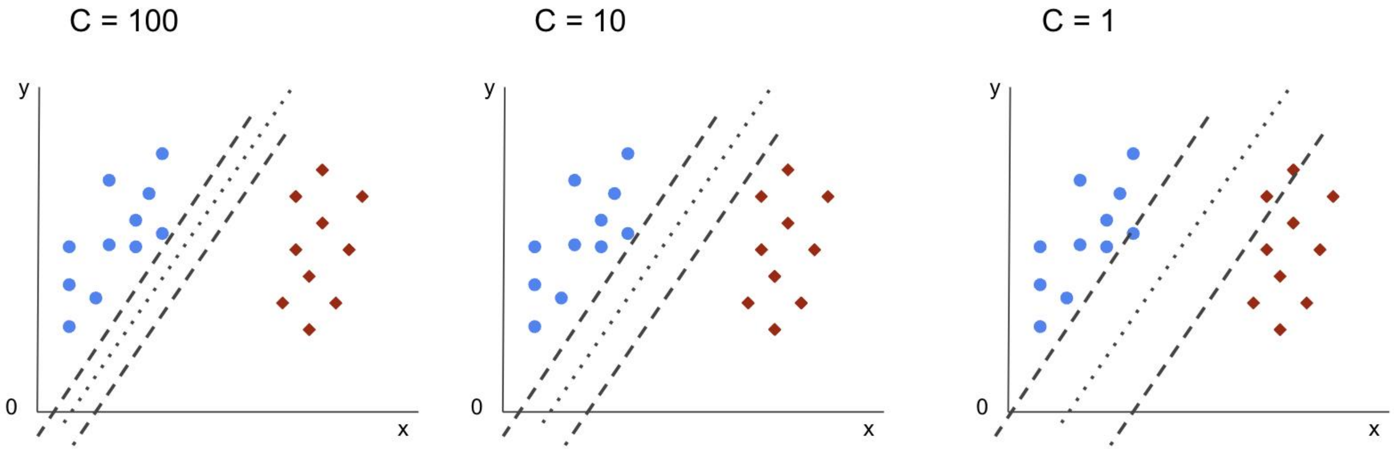


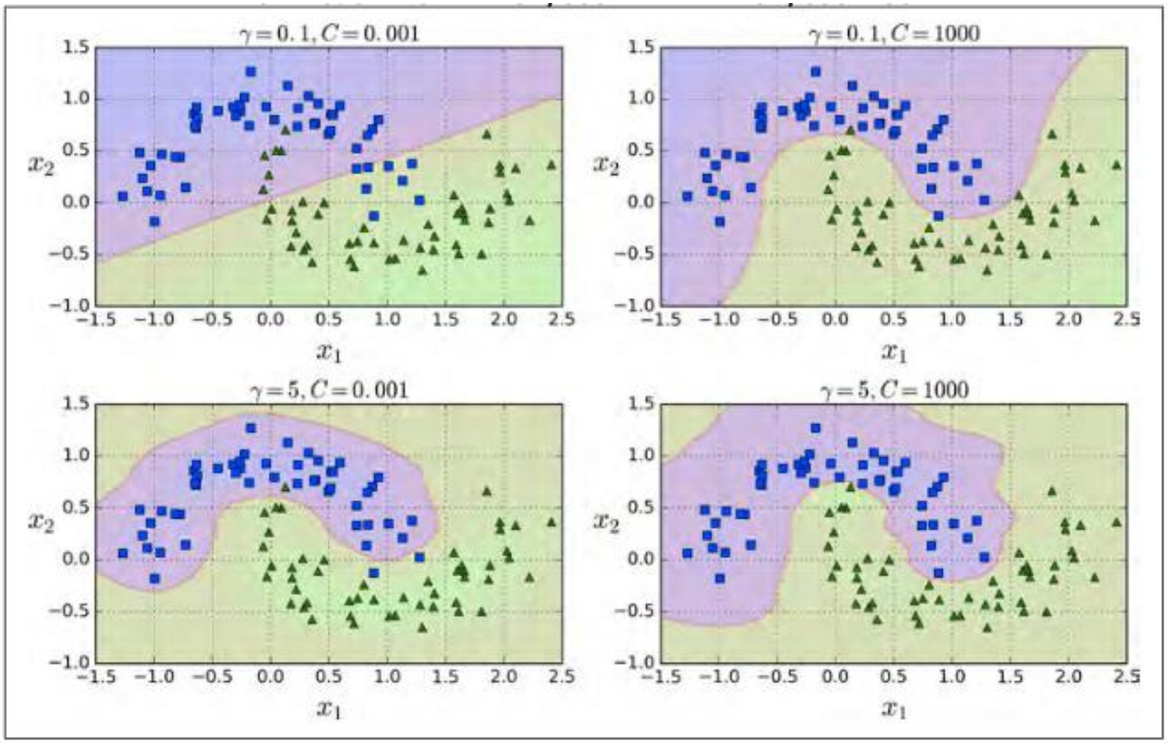


Regression



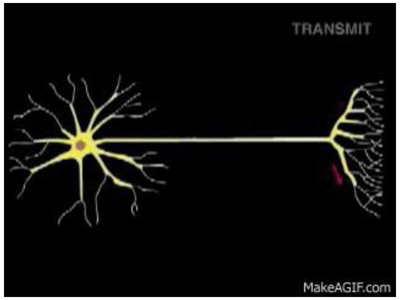
Support Vector Machines (SVMs)

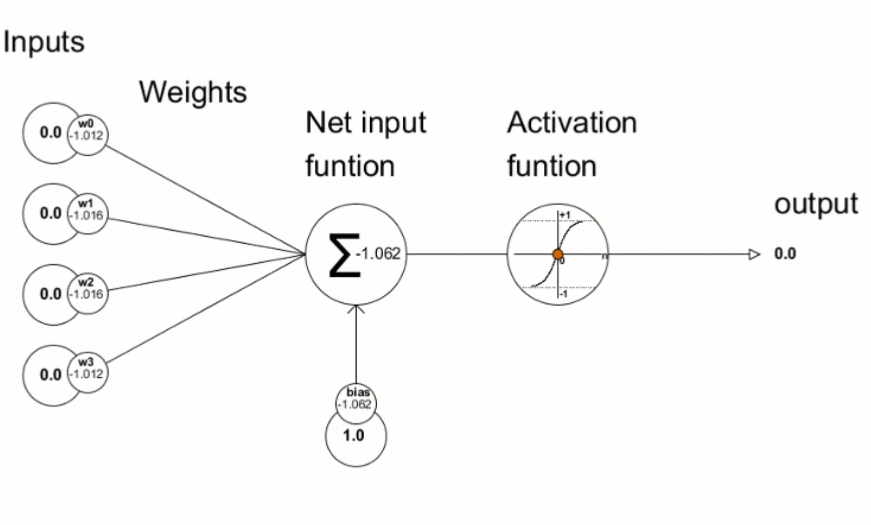




Neural Networks

- Neural networks models are inspired by biological neural networks.
- Model:
  - General: They consist of layers of interconnected nodes like neurons that transmit signals.
  - Hyperparameter Optimization
    - Number of layers
    - Number of nodes per layer
    - Activation functions
    - Regularization methods
    - Initial weights and biases
    - Learning rate
    - Batch size
    - Number of epochs





Neural Networks Hyperparameters

Number of layers: How many stacked transformations (input → hidden(s) → output) the network has; depth controls representational power.

Number of nodes per layer: The count of neurons in a layer; width sets the layer's capacity to capture features.

Activation functions: Nonlinear functions (e.g., ReLU, Tanh) applied to neuron outputs so the network can model nonlinear relationships.

Regularization methods: Techniques (e.g., L2, dropout, early stopping) that reduce overfitting by constraining or randomizing learning.

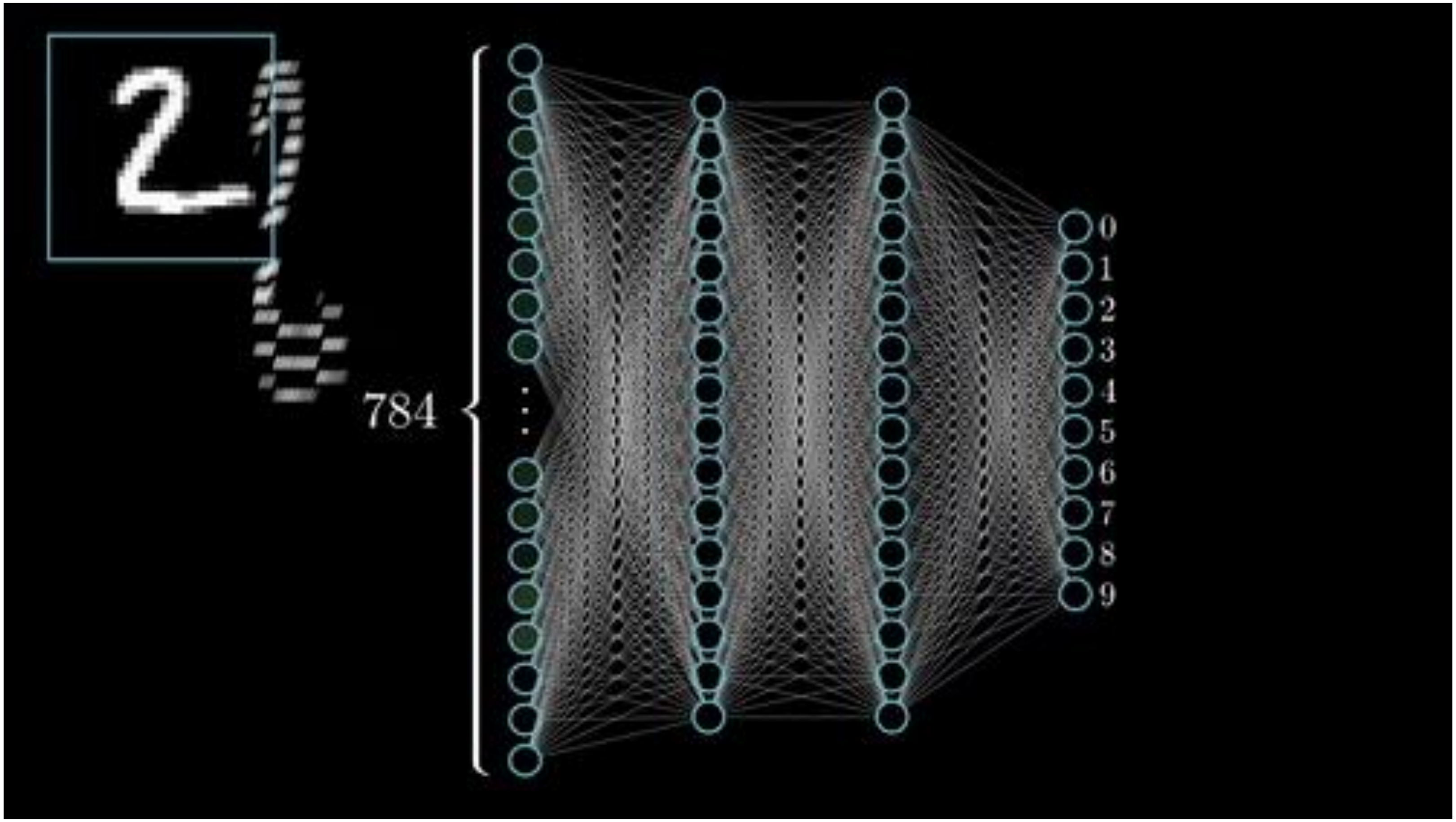
Initial weights and biases: The starting parameter values; good initialization (e.g., He/Xavier) speeds and stabilizes training.

Learning rate: The step size used by the optimizer when updating parameters each iteration.

Batch size: How many training samples are processed together to compute one gradient update.

Number of epochs: How many full passes the training loop makes over the entire dataset.

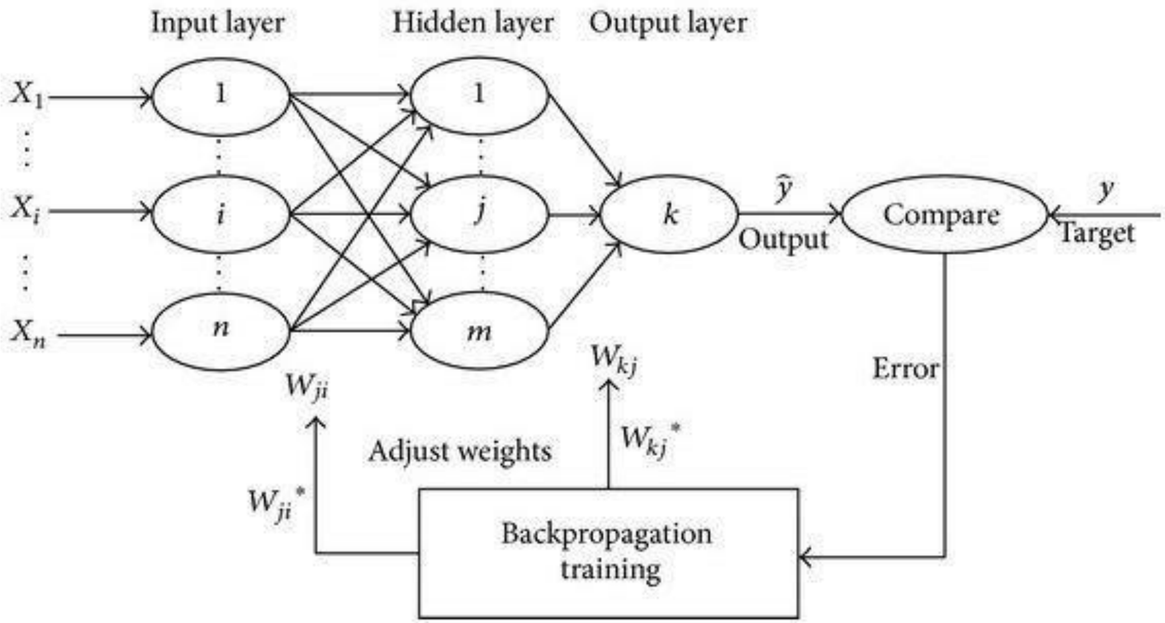
Neural Networks



What is Learning in Neural Network?

Learning (training):

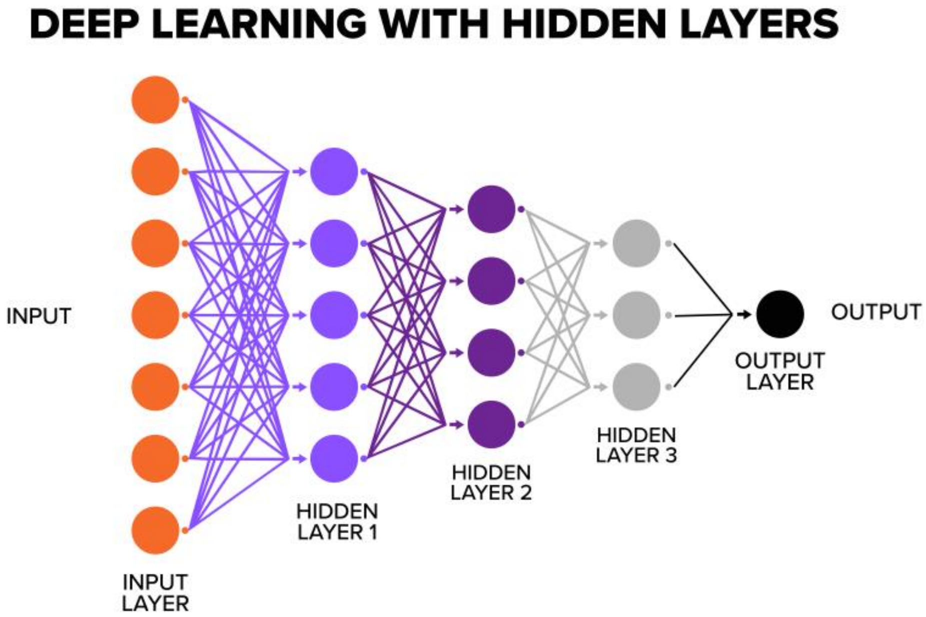
- Auto-adjusting model parameters with each new input sample so the output (**prediction**) gets closer to expected values (**ground truth**)
- forward-propagating activations (start input features and some random weights) for labeled input and back-propagating errors to adjust parameters in each node in order to minimize loss function



Deep Neural Networks

A **multi-layer artificial neural network** with multiple hidden layers between input and output.

Each layer learns hierarchical features, enabling complex pattern recognition.



Variations of DNN

CNN (Convolutional Neural Network):

- Designed for **spatial data** (images, grids).
- Learns local patterns through convolution filters.
- Strength: computer vision tasks (image classification, object detection).

RNN (Recurrent Neural Network):

- Designed for **sequential data** (time series, text, audio).
- Maintains a **hidden state** that carries information across time steps.
- Strength: short-term temporal dependencies.

LSTM (Long Short-Term Memory):

- An advanced type of **RNN**.
- Adds **memory cells + gating mechanisms*** to overcome vanishing gradients.
- Strength: captures **long-term dependencies** in sequences.

* **gated recurrent unit (GRU**)

EXTRA NOTES

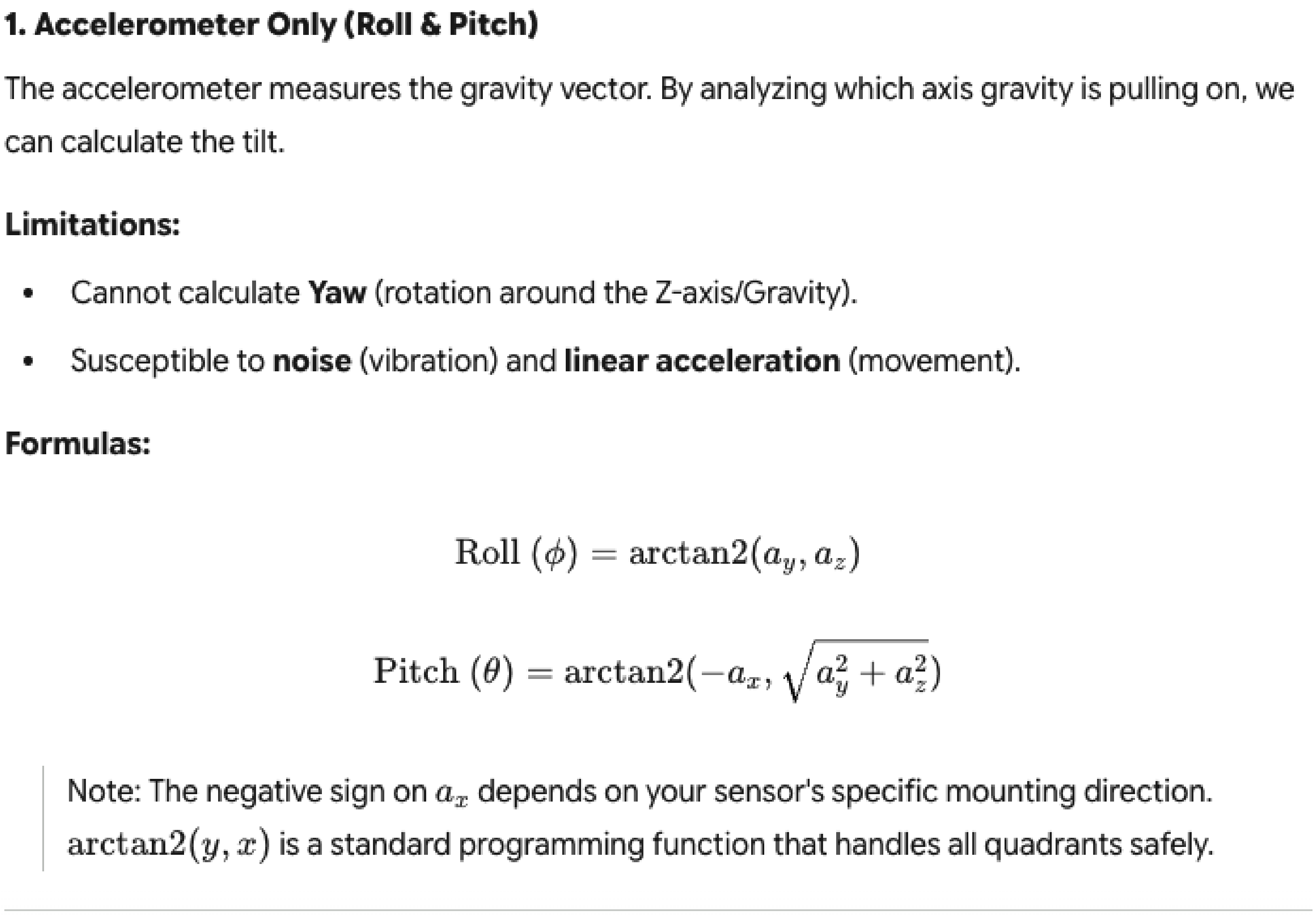
Assumptions & Axis Convention:

a_x, a_y, a_z: Accelerometer readings (in G-force).

g_x, g_y, g_z: Gyroscope readings (in radians/sec or degrees/sec).

m_x, m_y, m_z: Magnetometer readings (normalized).

We assume a standard **Right-Hand Coordinate System** (X=Forward, Y=Right, Z=Down).



1. Accelerometer Only (Roll & Pitch)

The accelerometer measures the gravity vector. By analyzing which axis gravity is pulling on, we can calculate the tilt.

Limitations:

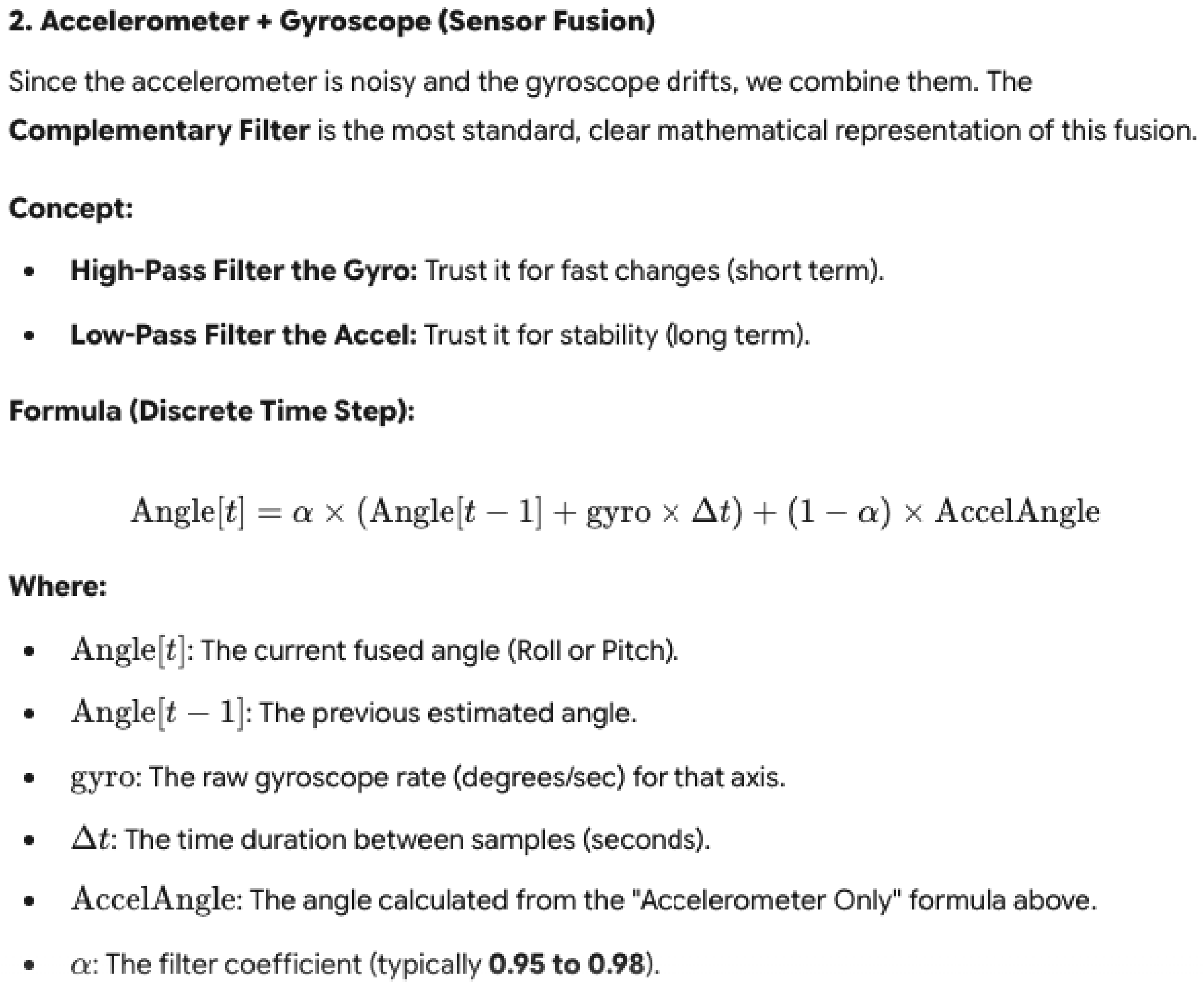
- Cannot calculate **Yaw** (rotation around the Z-axis/Gravity).
- Susceptible to **noise** (vibration) and **linear acceleration** (movement).

Formulas:

$$\text{Roll } (\phi) = \operatorname{arctan2}(a_y, a_z)$$

$$\text{Pitch } (\theta) = \operatorname{arctan2}(-a_x, \sqrt{a_y^2 + a_z^2})$$

> Note: The negative sign on a_x depends on your sensor's specific mounting direction. $\operatorname{arctan2}(y, x)$ is a standard programming function that handles all quadrants safely.



2. Accelerometer + Gyroscope (Sensor Fusion)

Since the accelerometer is noisy and the gyroscope drifts, we combine them. The **Complementary Filter** is the most standard, clear mathematical representation of this fusion.

Concept:

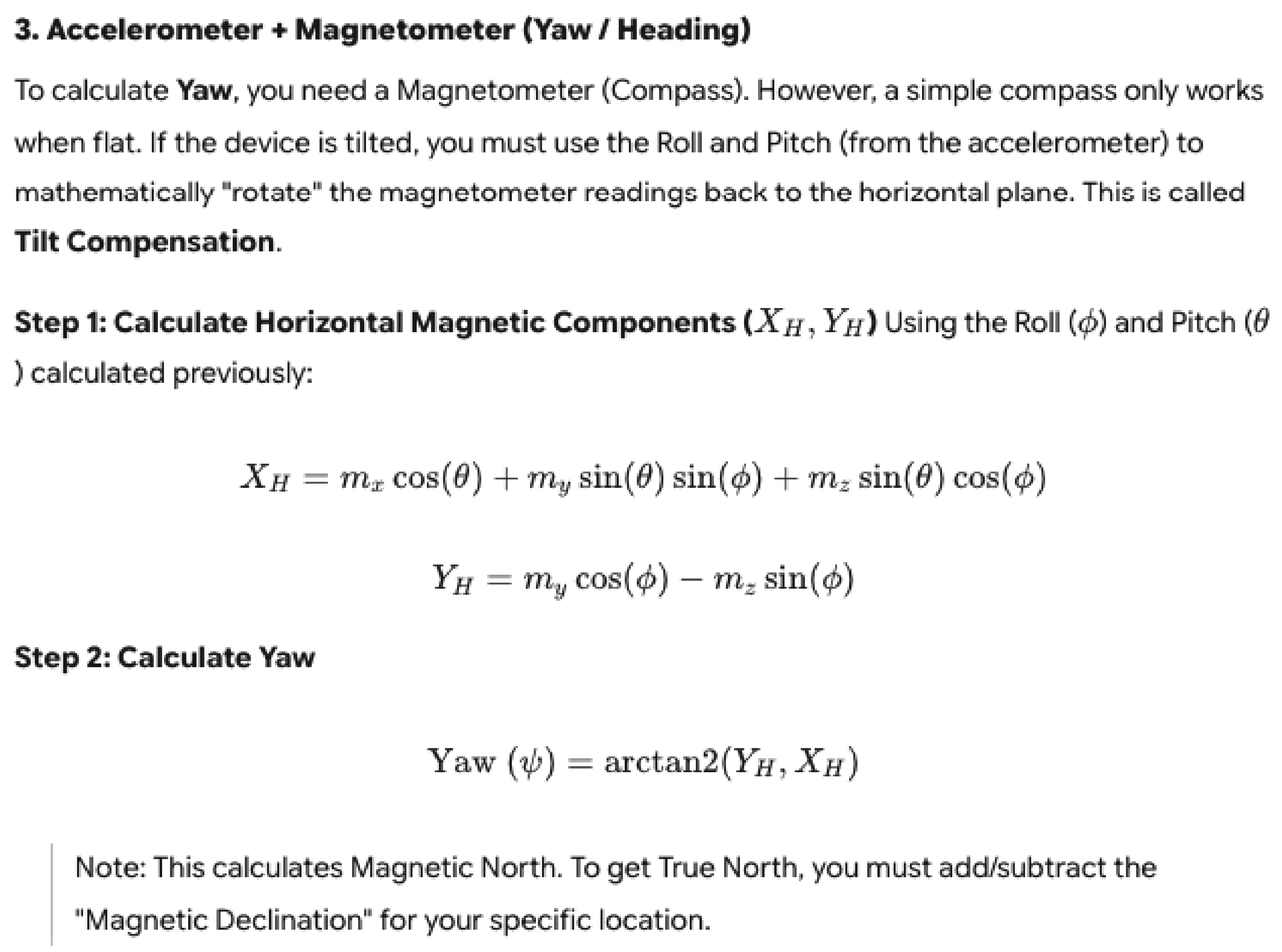
- **High-Pass Filter the Gyro:** Trust it for fast changes (short term).
- **Low-Pass Filter the Accel:** Trust it for stability (long term).

Formula (Discrete Time Step):

$$\text{Angle}[t] = \alpha \times (\text{Angle}[t-1] + \text{gyro} \times \Delta t) + (1-\alpha) \times \text{AccelAngle}$$

Where:

- $\text{Angle}[t]$: The current fused angle (Roll or Pitch).
- $\text{Angle}[t-1]$: The previous estimated angle.
- gyro: The raw gyroscope rate (degrees/sec) for that axis.
- Δt: The time duration between samples (seconds).
- AccelAngle: The angle calculated from the "Accelerometer Only" formula above.
- α: The filter coefficient (typically **0.95 to 0.98**).



3. Accelerometer + Magnetometer (Yaw / Heading)

To calculate **Yaw**, you need a Magnetometer (Compass). However, a simple compass only works when flat. If the device is tilted, you must use the Roll and Pitch (from the accelerometer) to mathematically "rotate" the magnetometer readings back to the horizontal plane. This is called **Tilt Compensation**.

Step 1: Calculate Horizontal Magnetic Components (X_H, Y_H) Using the Roll (ϕ) and Pitch (θ) calculated previously:

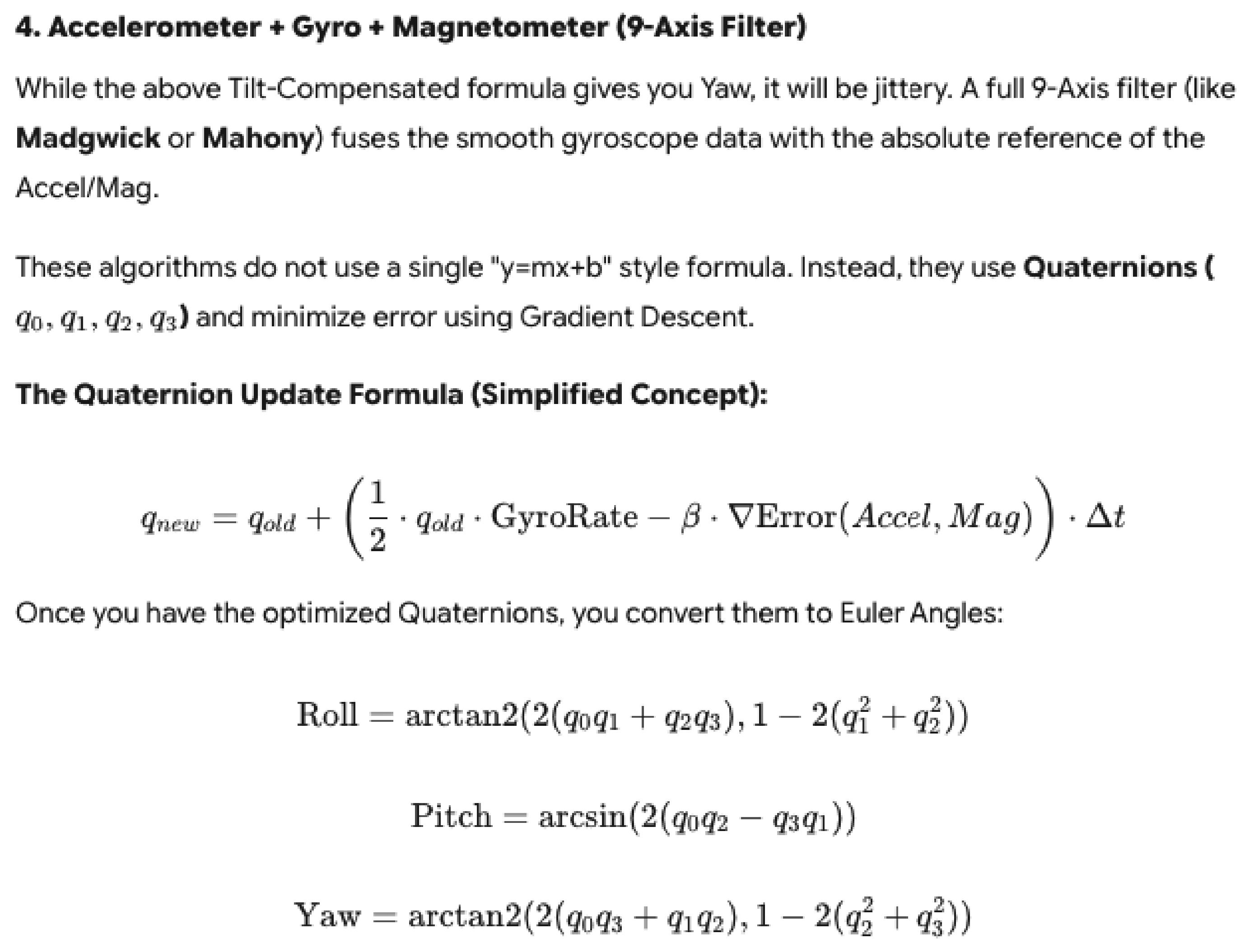
$$X_H = m_x \cos(\theta) + m_y \sin(\theta)\sin(\phi) + m_z \sin(\theta)\cos(\phi)$$

$$Y_H = m_y \cos(\phi) - m_z \sin(\phi)$$

Step 2: Calculate Yaw

$$\text{Yaw } (\psi) = \text{arctan2}(Y_H, X_H)$$

> Note: This calculates Magnetic North. To get True North, you must add/subtract the "Magnetic Declination" for your specific location.



4. Accelerometer + Gyro + Magnetometer (9-Axis Filter)

While the above Tilt-Compensated formula gives you Yaw, it will be jittery. A full 9-Axis filter (like **Madgwick** or **Mahony**) fuses the smooth gyroscope data with the absolute reference of the Accel/Mag.

These algorithms do not use a single "y=mx+b" style formula. Instead, they use **Quaternions (** q_0, q_1, q_2, q_3**)** and minimize error using Gradient Descent.

The Quaternion Update Formula (Simplified Concept):

$$q_{new} = q_{old} + \left(\frac{1}{2} \cdot q_{old} \cdot \text{GyroRate} - \beta \cdot \nabla\text{Error}(Accel, Mag)\right) \cdot \Delta t$$

Once you have the optimized Quaternions, you convert them to Euler Angles:

$$\text{Roll} = \text{arctan2}(2(q_0 q_1 + q_2 q_3), 1 - 2(q_1^2 + q_2^2))$$

$$\text{Pitch} = \arcsin(2(q_0 q_2 - q_3 q_1))$$

$$\text{Yaw} = \text{arctan2}(2(q_0 q_3 + q_1 q_2), 1 - 2(q_2^2 + q_3^2))$$