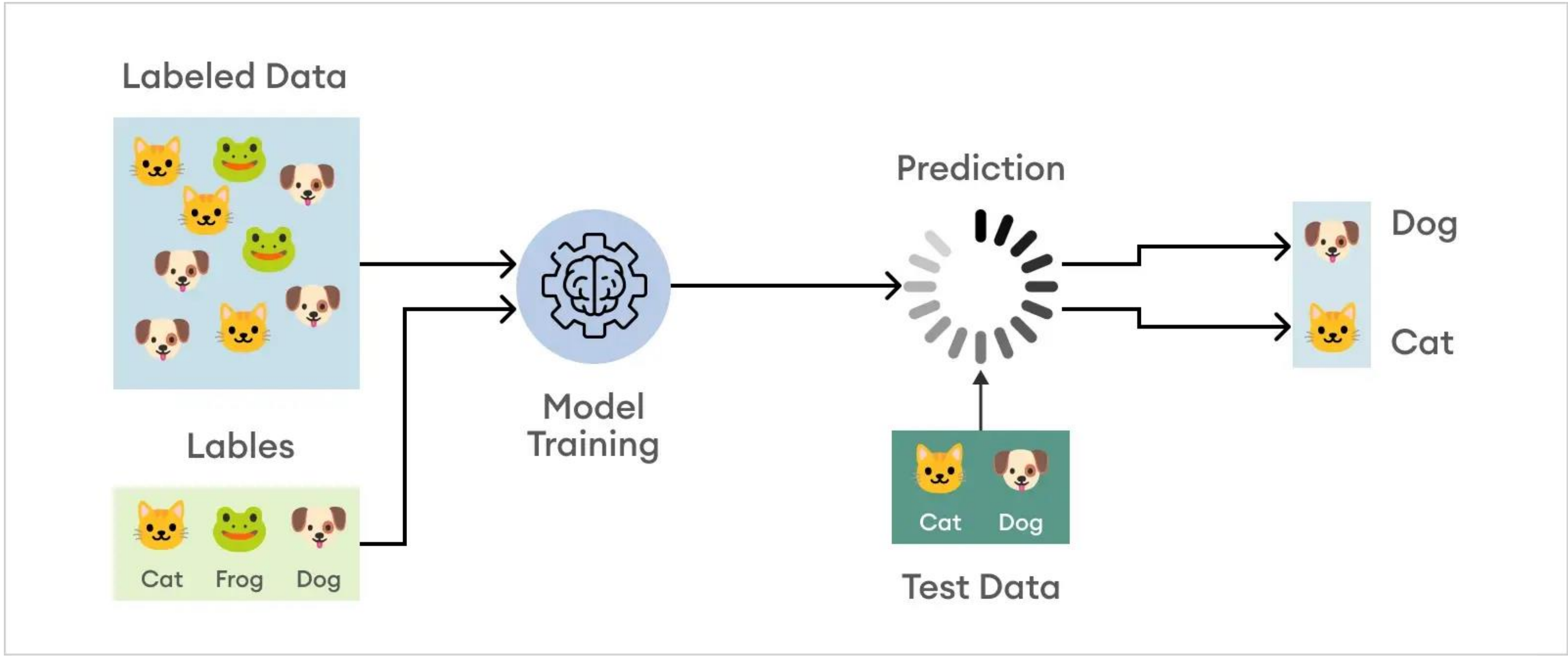
Embedded ML

ML Classification Algorithms

Prof. Venki Muthukumar

Classification

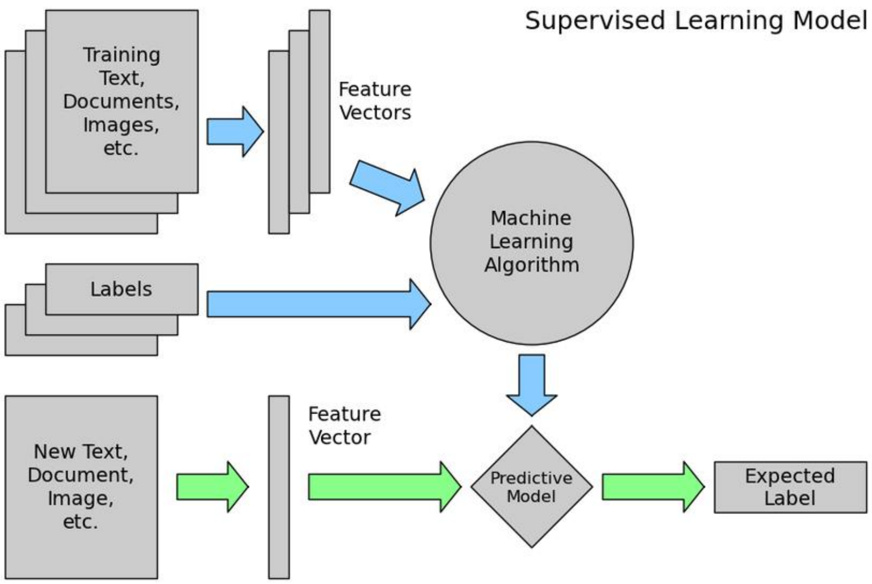
- **Classification** is a **supervised machine learning task** where a model learns to assign an input to **one of several discrete categories (classes)** based on labeled examples.



Classification

Embedded ML

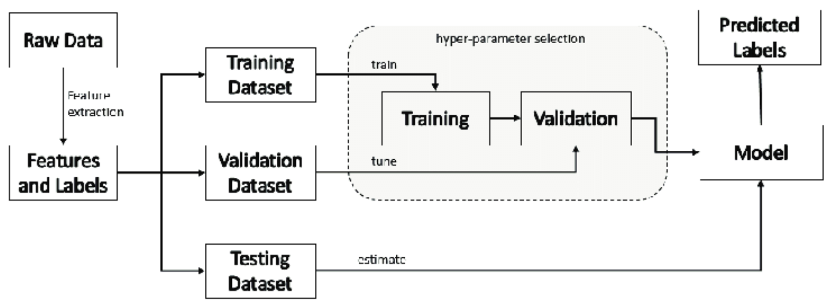
- Goal: Learn a mapping from **input features (X)** to **class labels (y)**
- Requires **labeled training data**
- Output: discrete class (e.g., walk / run / sit)
- Common applications:
  - Human Activity Recognition (IMU)
  - Gesture recognition
  - Fault detection
  - Audio / image classification
- Algorithms covered:
  - Decision Tree
  - Random Forest
  - Support Vector Machine (SVM)
  - Multi-Layer Perceptron (MLP)



Classification Pipeline

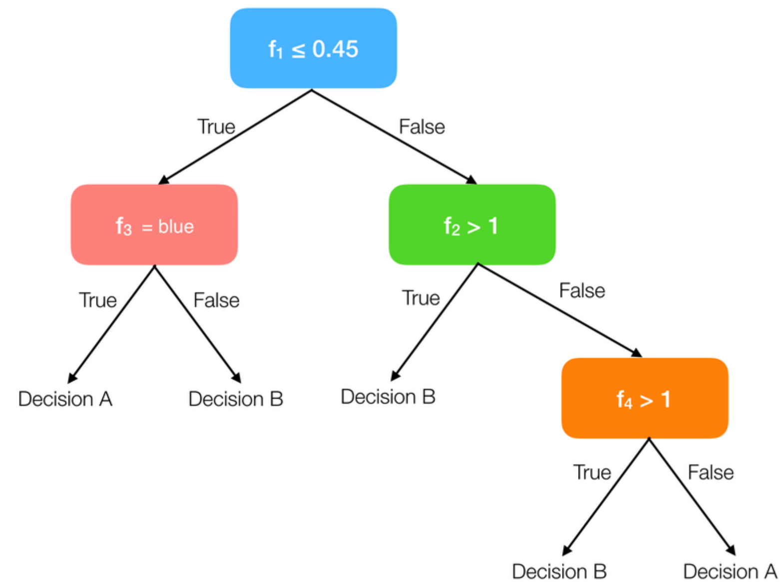
Embedded ML

- Data collection (sensors, logs, datasets)
- Feature extraction (statistical, spectral, embeddings)
- Train / validation / test split
- Model training
- Evaluation metrics:
  - Accuracy
  - Precision / Recall
  - Confusion Matrix
  - F1-score
- Deployment (Edge / Cloud)



Decision Tree (DT)

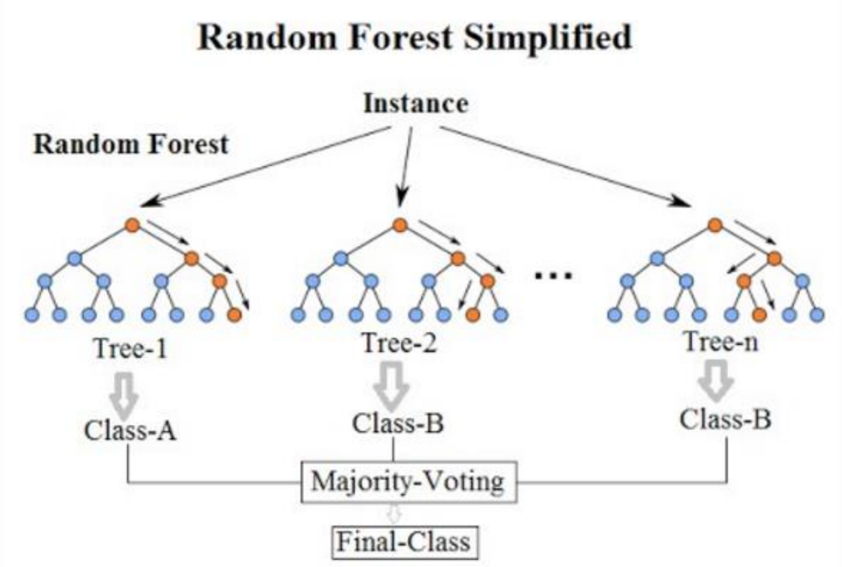
- Tree-structured model using **if–else rules**
- Splits data using feature thresholds
- Nodes: decision rules
- Leaves: class labels
- Popular splitting criteria:
  - Gini impurity (probability)
  - Entropy (Information Gain)
- **Advantages**
  - Easy to understand & visualize
  - No feature scaling required
  - Fast inference (good for embedded)
- **Limitations**
  - Overfitting on noisy data
  - Unstable to small data changes



Random Forest (RF)

Embedded ML

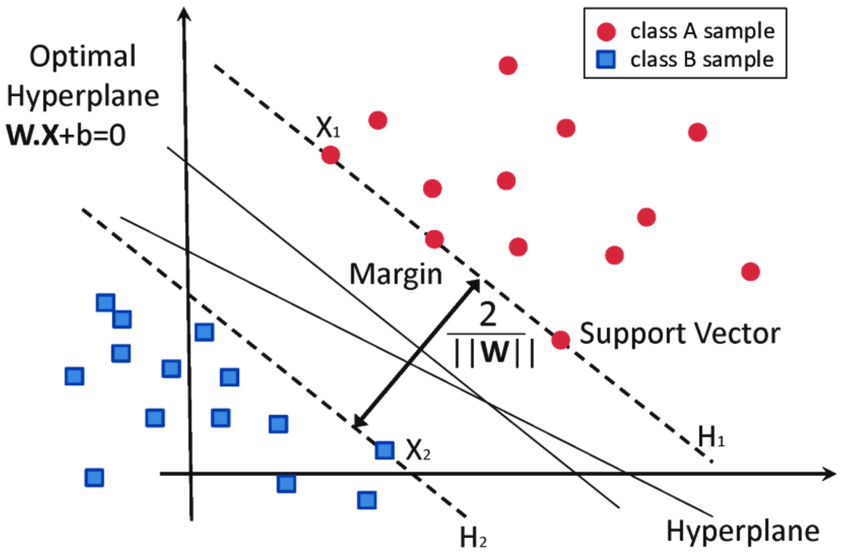
- Ensemble of **multiple decision trees**
- Each tree:
  - Trained on random subset of data
  - Uses random subset of features
- Final prediction = majority vote
- **Advantages**
  - High accuracy
  - Reduces overfitting
  - Works well with sensor data
- **Limitations**
  - Larger memory footprint
  - Less interpretable than single tree



Support Vector Machine (SVM)

Embedded ML

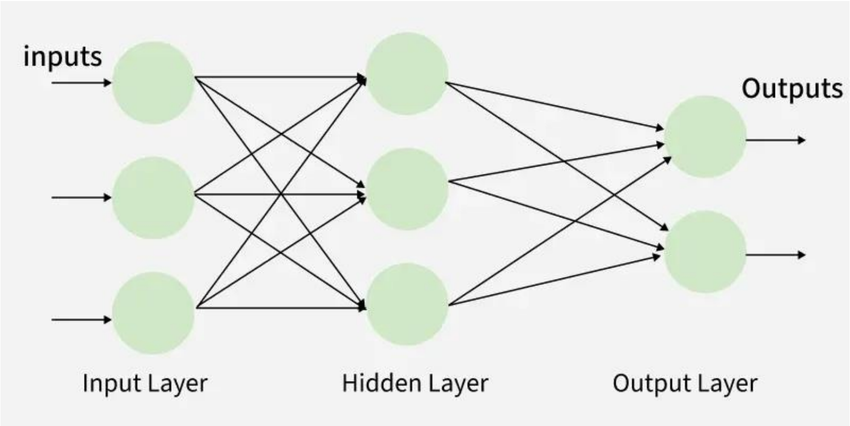
- Finds an **optimal separating hyperplane**
- Maximizes margin between classes
- Can use **kernel functions**:
  - Linear
  - Polynomial
  - RBF (Gaussian)
- **Advantages**
  - Strong performance on small datasets
  - Effective in high-dimensional spaces
- **Limitations**
  - Sensitive to parameter tuning
  - Computationally expensive for large datasets
  - Less suitable for real-time embedded inference



Multi-Layer Perceptron (MLP NN)

Embedded ML

- Fully connected **feed-forward neural network**
- Consists of:
  - Input layer
  - One or more hidden layers
  - Output layer
- Learns nonlinear decision boundaries
- **Advantages**
  - Very flexible
  - Learns complex feature interactions
  - Foundation for deep learning
- **Limitations**
  - Requires more data
  - Needs careful tuning
  - Harder to interpret

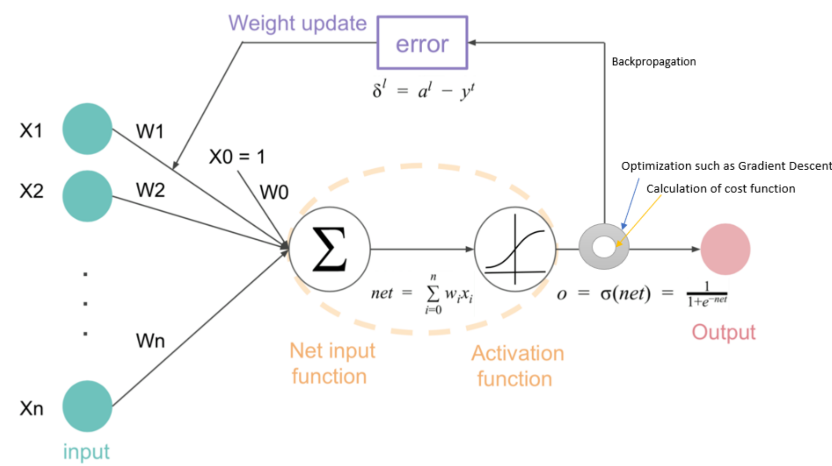


Backpropagation in Neural Networks

Embedded ML

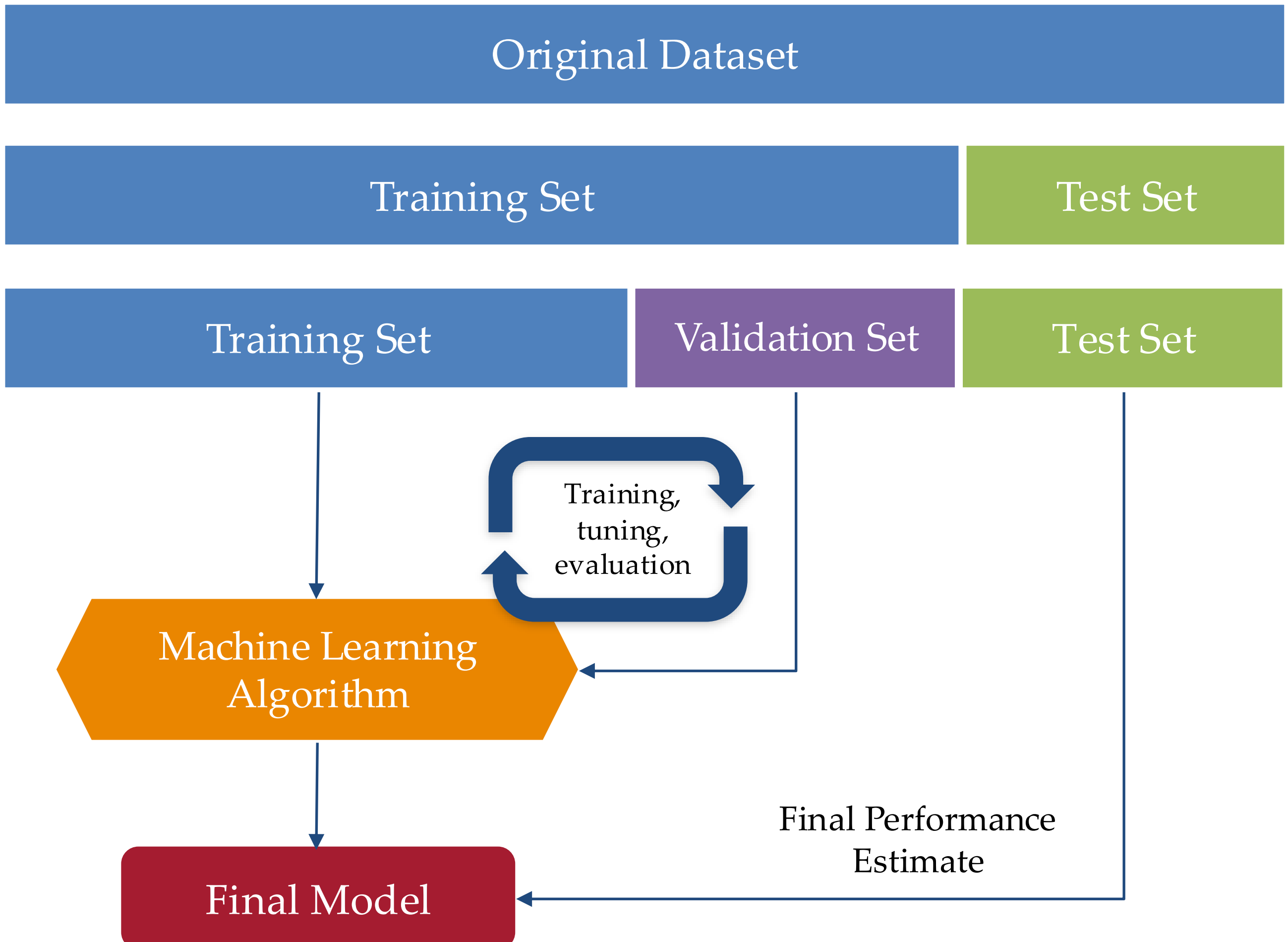
- **Backpropagation** is the algorithm used to **train neural networks**
- It computes how much each **weight contributes to the prediction error**
- Uses **gradient descent** to update weights and minimize loss
-

- **Two Phases of Training**
- $\boxed{1}$ **Forward Propagation**
- Input features → hidden layers → output
- Network produces prediction $\hat{y}$
- $\boxed{2}$ **Backward Propagation**
- Compute loss:
- $L(y, \hat{y})$
- Propagate error **backwards**
- Update weights using gradients



ML Dataset Modeling

Embedded ML

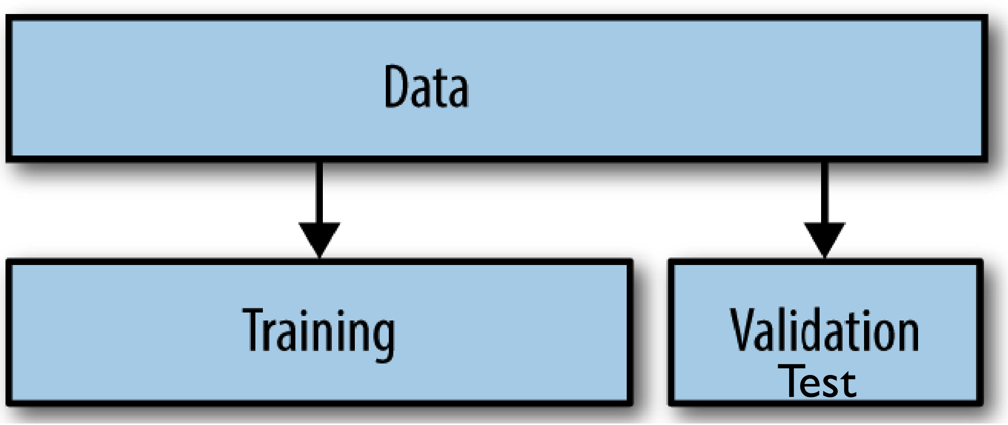


- Training data: labeled (ground truth - expected algorithm result; expensive!)
- Training: Algorithm uses labels to evaluate its accuracy on training data.

Holdout Validation (Train/Test Split)

- **Concept:**
  The dataset is divided into two disjoint subsets:
  - **Training set**: Used to fit the model (e.g., 70–80%).
  - **Test set**: Used to evaluate performance (e.g., 20–30%).
- **Advantages:**
  - Simple and fast to implement.
  - Useful for **large dataset.**
- **Disadvantages:**
  - high variance depends on how the data is split.

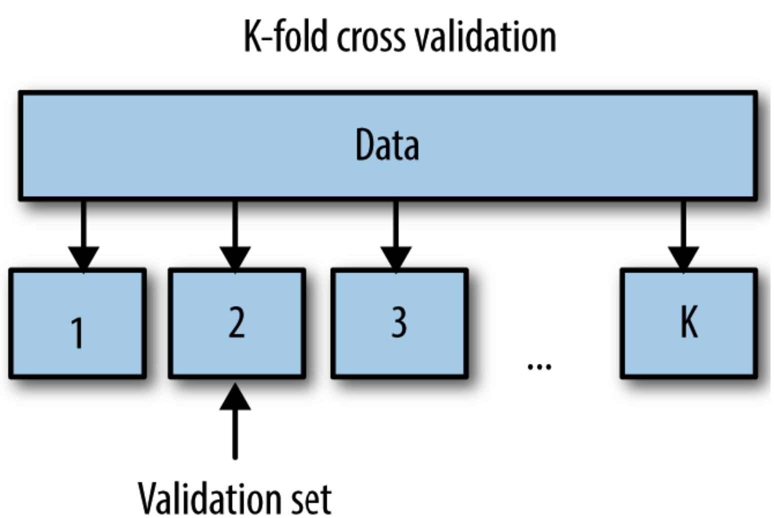
Hold-out validation



K-Fold Cross Validation

Embedded ML

- **Concept:**
  - The dataset is divided into **K equal-sized folds** (commonly K=5 or 10).
  - For each iteration, one fold is used as the **validation set**, and the remaining K-1 folds are used for **training**.
  - The process repeats K times, each time with a different validation fold.
  - Final performance = **average of all K validation scores**.
- **Advantages:**
  - Reduces variance.
  - more reliable estimate of performance.
- **Disadvantages:**
  - Computationally more expensive (train model K times).
  - slow for large datasets/models.
- Variations:
  - **pure K-Fold CV**, there is **no separate test set**.
  - Best practice - **final holdout test set** (never touched during K-Fold CV) is used to evaluate the model



What to consider for time series data?

Scheme	Preserves Order	Handles Drift	Computation	Best For
Holdout Split	✓	✗	Low	Large, stable data
Expanding Window	✓	Partial	Medium	Periodic retraining
Sliding/Rolling Window	✓	✓	Medium	Non-stationary data
Walk-Forward Validation	✓	✓	High	Short-term forecasts
Time Series CV (Blocked)	✓	✗	Medium	Hyperparameter tuning
Nested CV	✓	Partial	Very High	Fair evaluation + tuning

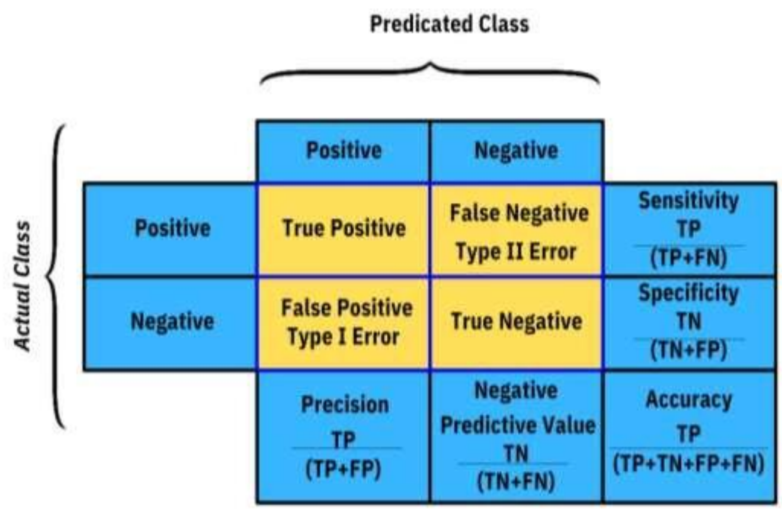
Sampling

- If dataset has 70% Class A and 30% Class B, How do you split the dataset? Avoid imbalance in split.

Method	When to Use	API
Random Sampling	Baseline, assumes data is independent and identically distributed.	train_test_split(X, y, test_size=0.2, shuffle=True)
Stratified Sampling	Classification, imbalanced data	train_test_split(..., stratify=y)
Cluster Sampling	Data grouped by clusters (e.g., patients, machines)	GroupKFold / GroupShuffleSplit
Group-wise CV	Ensure all samples from a group stay together	GroupKFold, LeaveOneGroupOut
Time Series Splits	Ordered data, no shuffling allowed	TimeSeriesSplit, or custom expanding/rolling splits
Nested CV	Model selection + final evaluation	Outer KFold + inner CV inside outer loop
Bootstrap Sampling	Estimate confidence intervals, robust metrics	sklearn.utils.resample (sampling with replacement)

Evaluation Metrics (Classification)

- **Accuracy**: overall correctness
- **Precision**: correctness of positive predictions
- **Recall**: ability to find all positives
- **F1-Score**: balance of precision & recall
- **Confusion Matrix**:
  - Shows per-class performance
  - Reveals misclassification patterns



Precision, Recall, and Accuracy

Embedded ML

- Precision (predictive accuracy)
  - Percentage of positive labels that are correct
  - Precision = (# true positives) / (# true positives + # false positives)
- Recall (sensitivity)
  - Percentage of positive examples that are correctly labeled
  - Recall = (# true positives) / (# true positives + # false negatives)
- Accuracy
  - Percentage of correct labels
  - Accuracy = (# true positives + # true negatives) / (# of samples)
- F1-score
  - Harmonic mean of precision and recall. Provides a balance between the two.

$$\text{F1 Score} = \frac{2 \times (\text{Precision} \times \text{Recall})}{\text{Precision} + \text{Recall}}$$

Improving Metrics

Metric	Condition for Good Classifier	If Low → How to Optimize
Precision	High → Few False Positives	• Increase decision threshold • Better feature selection • Regularization to reduce noise
Recall	High → Few False Negatives	• Lower decision threshold • Oversample minority class (SMOTE) • Use class weighting
Accuracy	High → Overall correctness (balanced classes)	• Check class imbalance • Use resampling techniques • Improve model generalization (features, tuning)
F1-Score	High → Balanced Precision & Recall	• Identify weak component (Precision or Recall) • Adjust threshold • Tune with F1 as scoring metric

Bias & Variance

Embedded ML

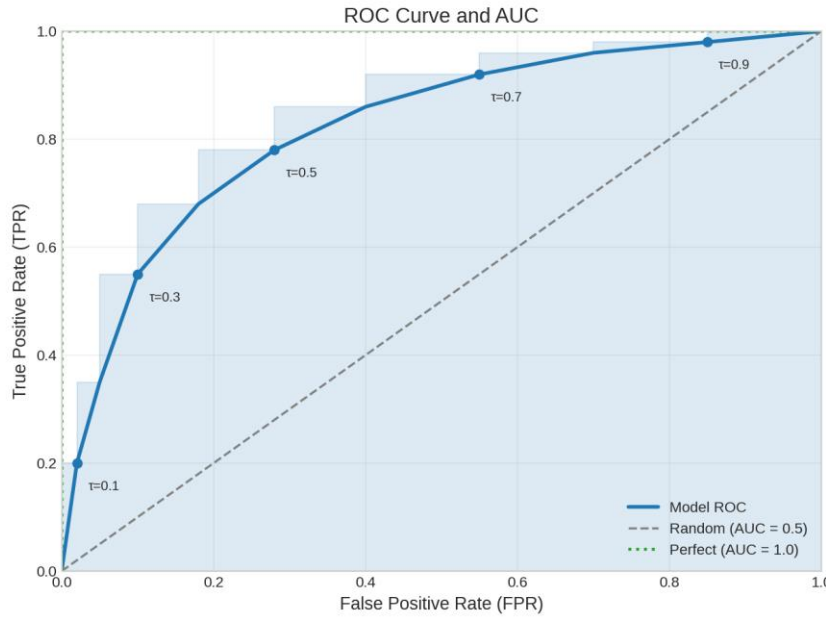
- Bias & Variance
  - bias and variance both indicate how well a model is likely to perform on new unseen data.
- Bias
  - expected difference between model's prediction and truth
- Variance
  - how much the model differs among training sets

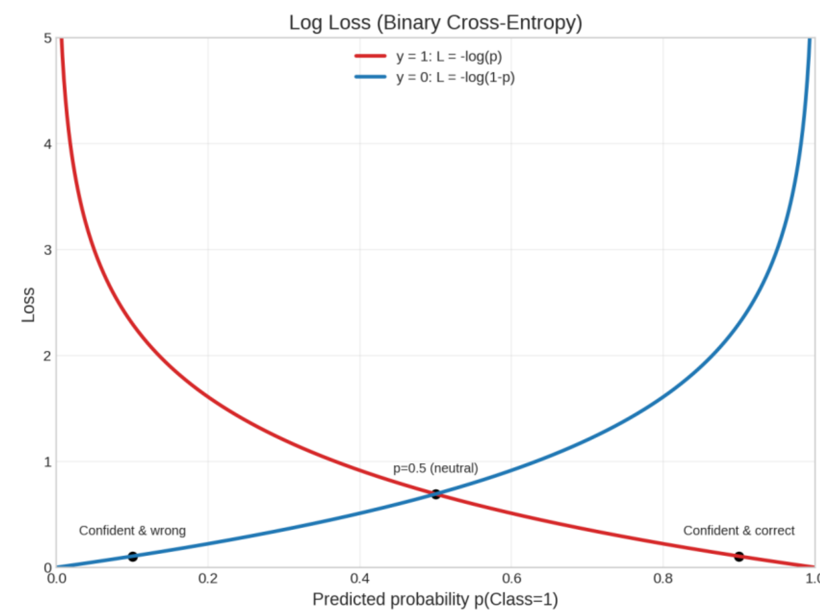
- Model Scenarios
  - High Bias (underfitting): Model makes inaccurate predictions on training data
  - High Variance (overfitting): Model does not generalize to new datasets
  - Low Bias: Model makes accurate predictions on training data
  - Low Variance: Model generalizes to new datasets

Other KPI for classification

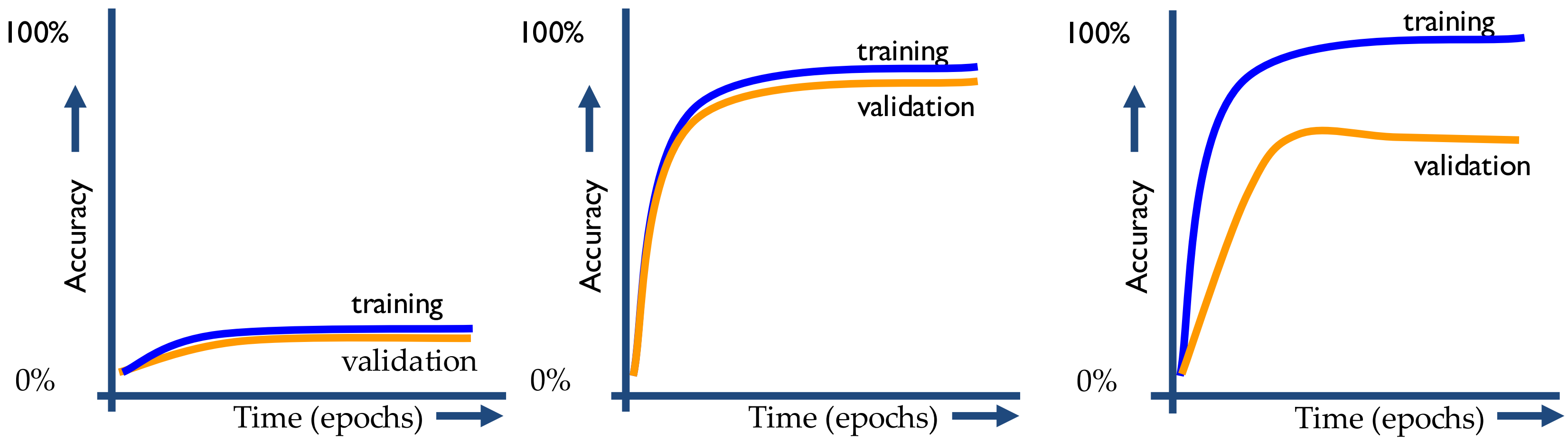
Embedded ML

- **ROC-AUC:** The probability that the model ranks a randomly chosen positive higher than a negative.
- **Log Loss:** A measure of classification error that penalizes confident but wrong predictions.





Spotting Underfitting and Overfitting



Underfit: Model performs poorly on training and validation data

Good fit: Model generalizes well from training to validation data

Overfit: Model predicts training data well but fails to generalize to validation data

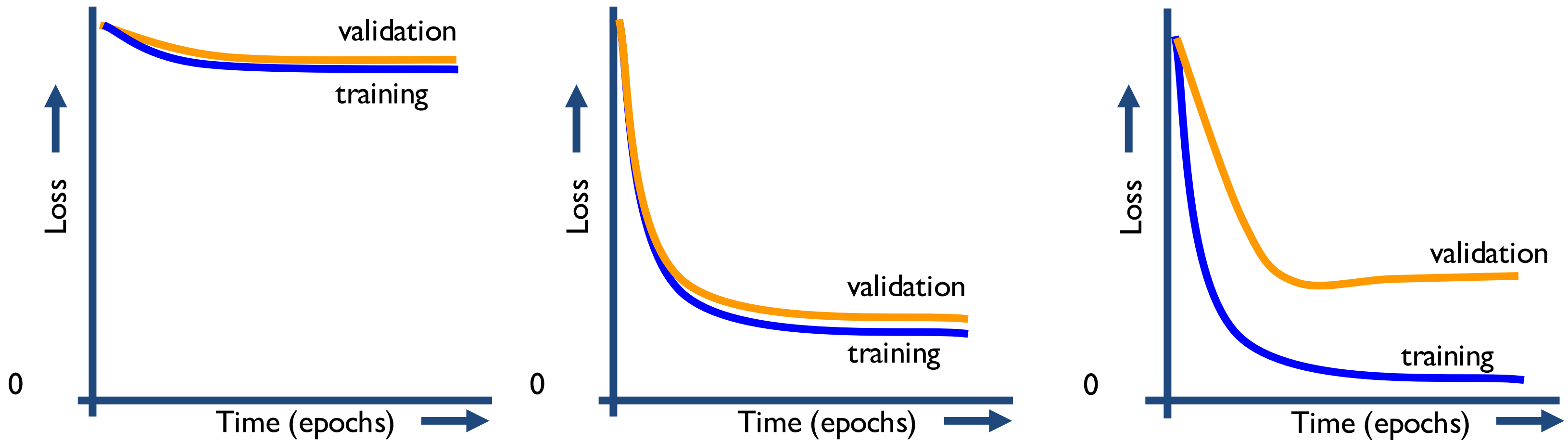
Accuracy = proportion of correctly classified samples.

$$\text{Accuracy} = \frac{\textit{Number of correct predictions}}{\textit{Total predictions}}$$

Only applies to **classification tasks** (not regression).

Embedded ML

Spotting Underfitting and Overfitting



Underfit: Model performs poorly on training and validation data

Good fit: Model generalizes well from training to validation data

Overfit: Model predicts training data well but fails to generalize to validation data

- **Loss** is a number computed from a **loss function** (e.g., MSE for regression, cross-entropy for classification).
- It measures the **error** between predicted outputs and true labels.
- The goal of training = **minimize loss**.

Fix Overfitting and Underfitting

Embedded ML

Issue	Symptoms	Fixes
Overfitting	High training accuracy, low test accuracy	• Simplify model (reduce layers, tree depth) • Apply regularization (L1/L2, dropout) • Early stopping • Data augmentation • Cross-validation • Reduce noisy features
Underfitting	Low accuracy on both training & test sets	• Increase model complexity (more layers, deeper trees) • Add informative features • Reduce regularization strength • Train longer (more epochs) • Hyperparameter tuning

HandsOn Session

- Codes @ http://github.com/venki666/embai
- Collect smooth data using the code IMU_SD_ST_AGEQ.ino for a series of actions (slow-moving action) of practical importance.
- Change the stored file name for each action. Place the recorded csv files in your python workspace and evaluate classification algorithms for your dataset.
- Estimate the classification accuracy for the below classifiers
  - Decision Tree
  - Random Forest
  - SVM
  - MLP NN
  - MLPBP NN
- Discuss/Comment on the performance of these algorithms for your dataset. Complete your wiki doc.