**Model Selection & Evaluation**

## **Train/Validation/Test Splits**

In machine learning, data is divided into separate sets to train, tune, and evaluate models effectively.  
This helps measure how well the model can generalize to unseen data.

## **Data Splitting**

Typically, data is divided as:

* Training Set: 60–70%
* Validation Set: 10–20%
* Test Set: 10–20%

**Training Set**

* Used to train the model — the model learns the relationship between inputs and outputs.
* Model parameters (weights) are updated here.

model.fit(X\_train, y\_train)

**Validation Set**

* Used to monitor model performance during training.
* Helps detect overfitting and tune hyperparameters.
* The model is not trained on this data; it’s only used for checking progress.

model.fit(X\_train, y\_train, validation\_split=0.2)

**Test Set**

* Used **after training is complete** to check the model’s **final performance**.
* It represents *completely unseen data* and gives an unbiased estimate of real-world accuracy.
* Used only once, no tuning is done after this step.

model.evaluate(X\_test, y\_test)

**Why Validation in Important**

* Training accuracy alone can be misleading — the model might be memorizing the data.
* Validation accuracy helps identify overfitting early.
* If training accuracy ↑ but validation accuracy ↓ → overfitting.

**K-Fold Cross-Validation**

### **Introduction**

K-Fold Cross-Validation is an improved method for model evaluation.  
 Instead of using one fixed validation set, it splits the dataset into *K equal parts (folds)* and performs training/testing multiple times — ensuring every data point is used for both training and validation.

### **Working Process**

1. The dataset is divided into K folds (e.g., 5 or 10).
2. The model is trained on K–1 folds and validated on the remaining fold.
3. This process repeats K times, each time with a different fold as validation.
4. The average performance across all folds gives a more reliable estimate of the model’s true accuracy.

### **Why Use K-Fold Cross-Validation**

* Reduces bias from a single random split.
* Gives a better estimate of model performance.
* Useful when the dataset is small.
* Helps in model selection and hyperparameter tuning.

Each fold acts once as validation — the final average shows overall model performance.

### **Choosing K**



**Bias Variance Tradeoff**

The bias–variance tradeoff explains how well a machine learning model can generalize to unseen data.

It’s about finding the right balance between two types of errors:

* Bias error
* Variance error

### **Bias**

Bias means error due to simplifying the problem too much.

The model makes strong assumptions, ignores patterns, and performs poorly on both train and test data.

High bias = underfitting

Example:  
 A linear regression model trying to fit a complex nonlinear curve.

### **Variance**

Variance means error due to too much sensitivity to training data.

The model learns noise or random fluctuations.

Performs very well on training data but poorly on unseen data.

High variance = overfitting

Example:  
 A deep decision tree memorizing the training set exactly.

**The Tradeoff**

Increasing model complexity → ↓ bias but ↑ variance

Decreasing model complexity → ↑ bias but ↓ variance

You need a sweet spot where both are balanced → best generalization

**The perfect Balance**

With high bias and low variance – The model will underfit

With low bias and high variance – The model will overfit

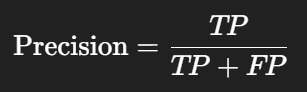
So, we need to find the perfect and ideal balance between bias and variance where both are low, so the model generalizes better.

**Performance Evaluation Metrics**

### **Precision**

Out of all predicted positives, how many were actually correct?

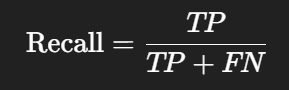
If a model is making a certain claim (positives), how correct is he in those claims?



High precision → fewer false alarms.

**Recall**

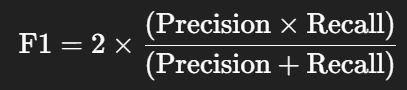
Out of all actual positives, how many did the model catch?



High recall → fewer misses.

**F1-Score**

The harmonic mean of Precision and Recall.



* Balances both Precision and Recall
* Useful when classes are imbalanced
* F1 close to 1 → excellent, close to 0 → poor

**ROC Curves & AUC**

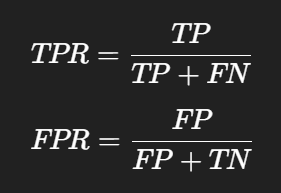
### **What is ROC**

ROC (Receiver Operating Characteristic) curve shows the performance of a classification model at all possible threshold values.

It plots:

* X-axis → False Positive Rate (FPR)
* Y-axis → True Positive Rate (TPR) (same as Recall)

### **Formula**



### **How does it work**

When your model predicts probabilities (not just 0/1), you can set different thresholds.

Example:

* At threshold **0.9**, only very confident predictions are positive → high precision, low recall
* At threshold **0.3**, more positives → high recall, but more false positives

The **ROC curve** shows how TPR and FPR change as the threshold moves.

### **AUC (Area Under Curve)**

* AUC measures the entire area under the ROC curve.
* Ranges from 0 to 1:
  + 1.0 → Perfect model
  + 0.5 → Random guess
  + < 0.5 → Worse than random

So, higher AUC = better performance.

### **When to Use ROC & AUC**

Use it when:

* You’re comparing multiple classifiers
* Classes are imbalanced
* You care about how well the model distinguishes between classes