MAE (Mean Absolute Error)

📘 What is it?

The **average of the absolute differences** between predicted and actual values.
It tells you **how wrong your predictions are**, on average.

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

📌 Example:

Actual: [3, 5, 7]
Predicted: [2, 5, 8]
Errors = [1, 0, 1] → MAE = (1 + 0 + 1) / 3 = **0.67**

✅ Pros:

- Simple to interpret
- Not sensitive to outliers

❌ Cons:

- Does not penalize large errors more than small ones

2. RMSE (Root Mean Squared Error)

📘 What is it?

The square root of the average of **squared** differences.
Gives **more weight to larger errors**.

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

📌 Example:

Actual: [3, 5, 7]
Predicted: [2, 5, 8]
Squared Errors = [1, 0, 1] → RMSE = √(2 / 3) ≈ **0.82**

✅ Pros:

- Penalizes large errors more than MAE
- Popular in many applications (e.g., forecasting)

❌ Cons:

- Sensitive to outliers

3. R^2 Score (Coefficient of Determination)

📘 What is it?

Measures the **proportion of variance explained** by the model.
Ranges from $-\infty$ **to 1**. Closer to **1** means better fit.

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

Where $\bar{y}$ is the mean of actual values.

📌 Example:

If $R^2 = 0.85$ → the model explains **85%** of the variability in the data.

✅ Pros:

- Indicates how well the model explains data
- Interpretable (like 85% of variation is explained)

❌ Cons:

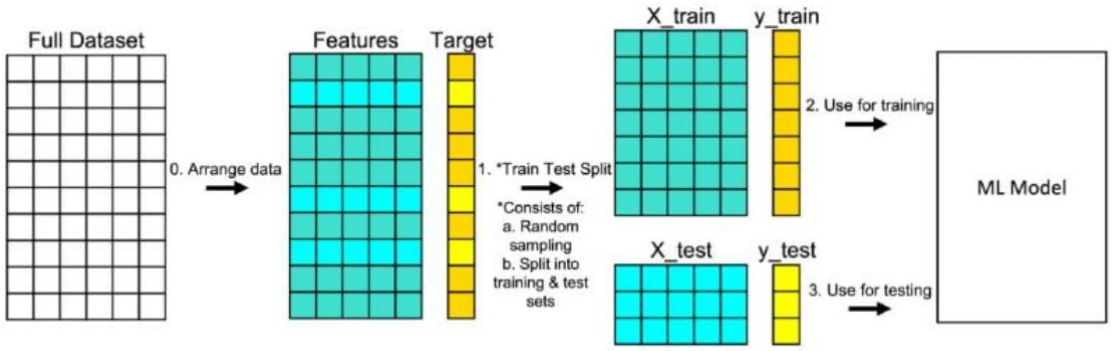
- Can be negative (for poor models)
- Doesn't show **how much** the predictions deviate

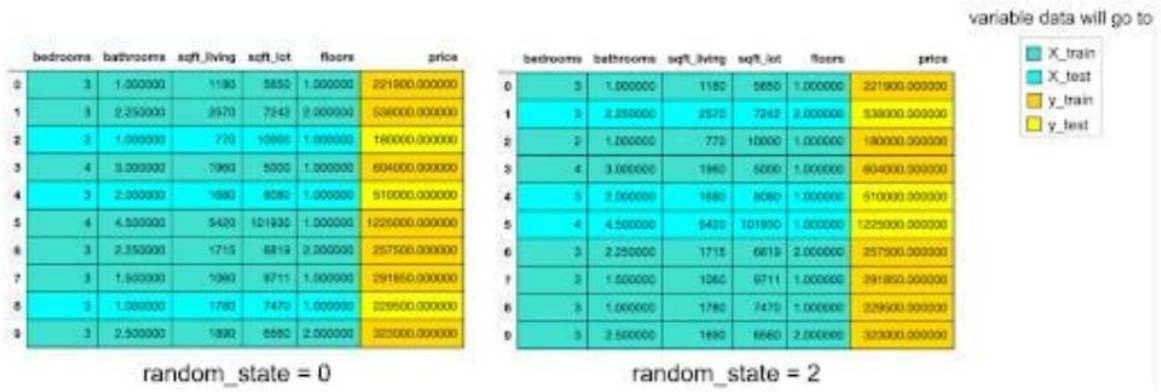
When to Use What?

- Use **MAE** when:
  - You want interpretability (e.g., "5 units off on average")
  - Outliers should not dominate
- Use **RMSE** when:
  - You want to **penalize large errors more**
  - More sensitive models (like deep learning, time series)
- Use **R^2** when:
  - You want to **evaluate overall model fit**
  - Used for comparing models on the same dataset

==========

Decision Trees into two main categories:





🌳 1. Classification Trees (CART for Classification)

- **Purpose**: Predict **categorical** outcomes (e.g., “Yes” or “No”, class 0 or 1)
- **Splitting Criteria**: Uses **Gini impurity** or **Entropy (Information Gain)** to split nodes.
- **Output**: A **class label** (like 0 or 1)

📈 2. Regression Trees (CART for Regression)

- **Purpose**: Predict **continuous** outcomes (e.g., prices, age, temperature)
- **Splitting Criteria**: Uses **Mean Squared Error (MSE)** or **Mean Absolute Error (MAE)**
- **Output**: A **numeric value** (like 45.2)

🔁 Comparison Table

Feature	Classification Tree	Regression Tree
Target Output	Categorical (Discrete Labels)	Continuous (Real-valued numbers)
Splitting Metric	Gini, Entropy	MSE, MAE
Use Cases	Spam detection, Loan approval	Price prediction, Age estimation
Output at Leaf	Most common class	Average of outputs in the leaf

Criterion	Used in	Description
`gini`	Classification	Gini impurity
`entropy`	Classification	Information gain
`squared_error`	Regression	Mean Squared Error (default)
`friedman_mse`	Regression	Faster variant of MSE for boosting
`absolute_error`	Regression	Mean Absolute Error (robust to outliers)
`poisson`	Regression	For count data (Poisson-distributed)

Bagging, also known as bootstrap aggregation, is the ensemble learning method that is commonly used to reduce variance within a noisy data set.

ensemble learning refers to a group (or ensemble) of base learners, or models, which work collectively to achieve a better final prediction.

In bagging, weak learners are trained in parallel,

but in boosting, they learn sequentially.

Redistribution of weights helps the algorithm identify the parameters that it needs to focus on to improve its performance.

AdaBoost, which stands for “adaptative boosting algorithm”.

Bagging algorithm, which has three basic steps:

Bootstrapping: Bagging leverages a bootstrapping sampling technique to create diverse samples. Select a data point from the training data set, able to select the same instance multiple times.

Parallel training: These bootstrap samples are then trained independently and in parallel with each other using weak or base learners.

In the case of regression, an average is taken of all the outputs predicted by the individual classifiers; this is known as **soft voting.**

For classification problems, the class with the highest majority of votes is accepted; this is known as **hard voting or majority voting.**

Random Forest

Random Forest is a machine learning algorithm. This algorithm brings together many decision trees and performs classification, regression and other prediction operations on complex data sets.

The Random Forest algorithm is based on the creation of more than one decision tree and the interaction of these trees with each other. Each tree is trained by randomly selecting features and using a random subsampling (bootstrap) part of the dataset. In this way, each tree creates a prediction model on its own.

What is gradient descent?

Gradient descent is an optimization algorithm which is commonly-used to train machine learning models and neural networks. It trains machine learning models by minimizing errors between predicted and actual results.

Loss function refers to the error of one training example,

Cost function calculates the average error across an entire training set.

How Gradient Descent Works

Gradient Descent **starts somewhere on the curve** and takes steps to reach the **lowest point**:

1. Start at a point on the curve.
2. Calculate the slope (gradient) of the loss function at that point.
3. Move in the opposite direction of the gradient (downhill).
4. Repeat until the slope becomes nearly zero (you reach the bottom).

The gradient is just the **derivative** of the loss function:

$$w = w - \alpha \cdot \frac{dJ(w)}{dw}$$

Where:

- w is the weight
- α is the learning rate (step size)
- $\frac{dJ(w)}{dw}$ is the slope (gradient)

Local Minimum

A **local minimum** is a point where the function value is **lower than all nearby points**, but **not necessarily the lowest overall**.

Global Minimum

A **global minimum** (also called **absolute minimum**) is the point where the function reaches its **lowest possible value over its entire domain**

Formal Definition:

A point `x = b` is a **global minimum** of a function `f(x)` if:

For **all** values of `x` in the domain of `f`,
f(b) ≤ f(x)

Gradient boosting is based on training weak learners independently of each other and **developing the next learner by learning from the mistakes of these learners**. With this technique, GBM aims to reduce the difference (i.e. error) between the predicted values of each learner and the actual values. While constructing each learner, the GBM algorithm finds the most appropriate model parameters by using the gradient descent method.

XGBoost

XGBoost (eXtreme Gradient Boosting) is an expansion or enhancement of the Gradient Boosting Machine (GBM) algorithm. Unlike the GBM algorithm, **XGBoost has a faster solution that can perform parallel computation on large data sets.**

Ensemble Learning?

Ensemble Learning is a technique where **multiple models (often called "weak learners") are combined to produce a stronger overall model**.

The main idea is:

A group of weak learners working together can be stronger than a single strong learner.

✅ Types of Ensemble Methods:

1. **Bagging (Bootstrap Aggregating)**
2. **Boosting**
3. **Stacking**

Bagging stands for **Bootstrap Aggregating**. It's an ensemble technique that:

1. **Trains multiple models** on **different random subsets** of the training data (with replacement).
2. Each model (usually a **decision tree**) is trained independently.
3. The predictions from each model are then **combined**:
    - For **classification**: majority voting.
    - For **regression**: average prediction.

🎯 **Goal of Bagging:** Reduce variance (overfitting) by averaging multiple models.

📌 Popular Example:

- **Random Forest** = Bagging + Decision Trees.

1. Random Forest

☑ Key Features:

- Type: Bagging ensemble
- Base Learner: Decision Trees
- Technique: Combines multiple decision trees trained on bootstrapped samples and averages their predictions (for regression) or uses majority voting (for classification).
- Key Parameters:
    - n_estimators: Number of trees
    - max_depth: Maximum depth of each tree
    - criterion: Gini or Entropy (for classification)

🧠 Concept:

Each tree is trained on a different random subset of data, with a random subset of features. This reduces overfitting and increases generalization.

2. Gradient Boosting Machines (GBM)

☑ Key Features:

- Type: Boosting ensemble
- Base Learner: Typically shallow decision trees
- Technique: Sequentially builds trees where each new tree corrects errors from the previous one.
- Key Parameters:
    - n_estimators: Number of boosting rounds
    - learning_rate: Shrinks the contribution of each tree
    - max_depth: Maximum depth of individual trees

🧠 Concept:

Instead of training all trees independently, GBM builds them sequentially, minimizing a loss function.

3. Model Stacking

☑ Key Features:

- Type: Meta-ensemble method
- Base Learners: Can use any models (e.g., logistic regression, SVM, decision trees)
- Technique: Combines multiple model predictions using a meta-model (e.g., another logistic regression)

🧠 Concept:

Train several base models and then use their outputs as features for a second-level model (meta-learner) to make the final prediction.

🔍 Comparison:

Technique	Goal	How It Works
Bagging	Reduce variance	Train models on different data samples, average or vote predictions
Boosting	Reduce bias & variance	Train models sequentially, each correcting errors of the previous model
Stacking	Combine diverse models	Train multiple models and use another model (meta-learner) to combine them

Method	Handles Overfitting	Learns from Errors	Works in Parallel	Complexity	Customization
Random Forest	✅	❌ (Independent trees)	✅	Medium	Moderate
Gradient Boosting	⚠️ (Needs tuning)	✅ (Sequential trees)	❌	High	High
Model Stacking	✅	✅ (via meta-model)	⚠️	High	High

Boosting

Boosting is an **ensemble technique** that builds models **sequentially**, where each new model focuses on correcting the errors made by the previous ones.

Boosting = Train weak learners one after the other → combine them → strong learner.

Popular Boosting Algorithms:

- ✅ **XGBoost**
- ✅ **LightGBM**

- ✅ **CatBoost**

1️⃣ XGBoost (Extreme Gradient Boosting)

🔹 Key Features:

- Highly optimized and fast.
- Handles missing values automatically.
- Regularization (L1 & L2) to avoid overfitting.
- Can work on large datasets efficiently.
- Parallelized tree construction.

🔧 Use Cases:

- Kaggle competitions 🏆
- Financial models
- Classification/Regression tasks

LightGBM (Light Gradient Boosting Machine)

🔹 Key Features:

- **Very fast**: Uses histogram-based algorithms.
- **Leaf-wise** tree growth instead of level-wise (leads to better accuracy).
- Supports categorical features directly.
- **Efficient memory usage**.

🧠 Difference from XGBoost:

- XGBoost grows **level-wise** (balanced).
- LightGBM grows **leaf-wise** (deeper, may overfit on small data).

CatBoost (Categorical Boosting)

🔹 Key Features:

- Handles **categorical variables** automatically (no need for encoding).
- Efficient with **less hyperparameter tuning**.
    - **Hyperparameters** are external configurations of a model that are **set before training** (like `n_estimators`, `learning_rate`, `max_depth` in tree-based models). Tuning these helps find the best combination that leads to optimal performance.
    - Think of it like choosing the best recipe ingredients before you start cooking 🍲.

- Faster training and better performance with **default settings**.
- Works well with **imbalanced datasets**.

🎯 Ideal For:

- Datasets with many categorical features.
- Quick, accurate predictions with minimal tuning.

🔄 Comparison Table:

Feature	XGBoost	LightGBM	CatBoost
Speed	Fast	Fastest	Fast
Memory Efficiency	Moderate	High	Moderate
Categorical Handling	Manual	Basic	Automatic ✅
Tree Growth	Level-wise	Leaf-wise	Symmetric
Overfitting Risk	Lower	Higher	Low
Best For	Numeric Data	Large Data	Mixed Data

✅ Hyperparameters to Tune (for Tree-based Models)

Algorithm	Important Hyperparameters
Random Forest	`n_estimators`, `max_depth`, `min_samples_split`, `max_features`
XGBoost	`n_estimators`, `max_depth`, `learning_rate`, `subsample`, `colsample_bytree`
LightGBM	`num_leaves`, `learning_rate`, `max_depth`, `n_estimators`, `min_data_in_leaf`
CatBoost	`iterations`, `depth`, `learning_rate`, `l2_leaf_reg`

Hyperparameters

1. `n_estimators` (All Ensemble Models)

- **Meaning**: Number of trees (or boosting rounds).
- **Effect**: More trees = better learning (up to a point), but slower.

- **Typical Range**: 50 to 1000+

```
model = RandomForestClassifier(n_estimators=100)
```

---

2. `max_depth`

- **Meaning**: Maximum depth of each decision tree.
- **Effect**:
  - Small values prevent overfitting (shallow trees).
  - Large values capture more detail but may overfit.
- **Typical Range**: 3 to 30

```
model = XGBClassifier(max_depth=5)
```

---

3. `learning_rate` (Boosting models: XGBoost, LightGBM, CatBoost)

- **Meaning**: Step size to update weights after each boosting round.
- **Effect**:
  - Low value = slow learning but better accuracy.
  - High value = faster but may overfit.
- **Typical Range**: 0.01 to 0.3

```
model = XGBClassifier(learning_rate=0.1)
```

---

4. `subsample` (XGBoost, LightGBM, CatBoost)

- **Meaning**: Fraction of data used per boosting round.
- **Effect**:
  - Prevents overfitting by introducing randomness.
- **Typical Range**: 0.5 to 1.0

```
model = XGBClassifier(subsample=0.8)
```

---

5. `colsample_bytree` (XGBoost, LightGBM)

- **Meaning**: Fraction of features to consider per tree.
- **Effect**: Like feature bagging. Helps prevent overfitting.

```
model = XGBClassifier(colsample_bytree=0.7)
```

---

6. `min_child_weight` (XGBoost)

- **Meaning**: Minimum sum of instance weight (Hessian) needed in a child.
- **Effect**:
  - Higher → more conservative tree (prevents splits with few samples).

- Lower → allows more splits, might overfit.

---

7. `early_stopping_rounds` (XGBoost, LightGBM, CatBoost)

- **Meaning**: Stop training if validation score doesn't improve after N rounds.
- **Effect**: Prevents overfitting and unnecessary computation.

```
model.fit(X_train, y_train, eval_set=[(X_val, y_val)],
early_stopping_rounds=10)
```

---

8. `boosting_type` (LightGBM-specific)

- **Options**:
  - `'gbdt'`: Gradient Boosting Decision Tree (default)
  - `'dart'`: Dropouts meet Multiple Additive Regression Trees
  - `'goss'`: Gradient-based One-Side Sampling

---

9. `loss_function` (CatBoost-specific)

- **Meaning**: Objective function like `Logloss`, `RMSE`, etc.
- **Example**:

```
model = CatBoostClassifier(loss_function='Logloss')
```

Reference:

Image: Michael Galarnyk