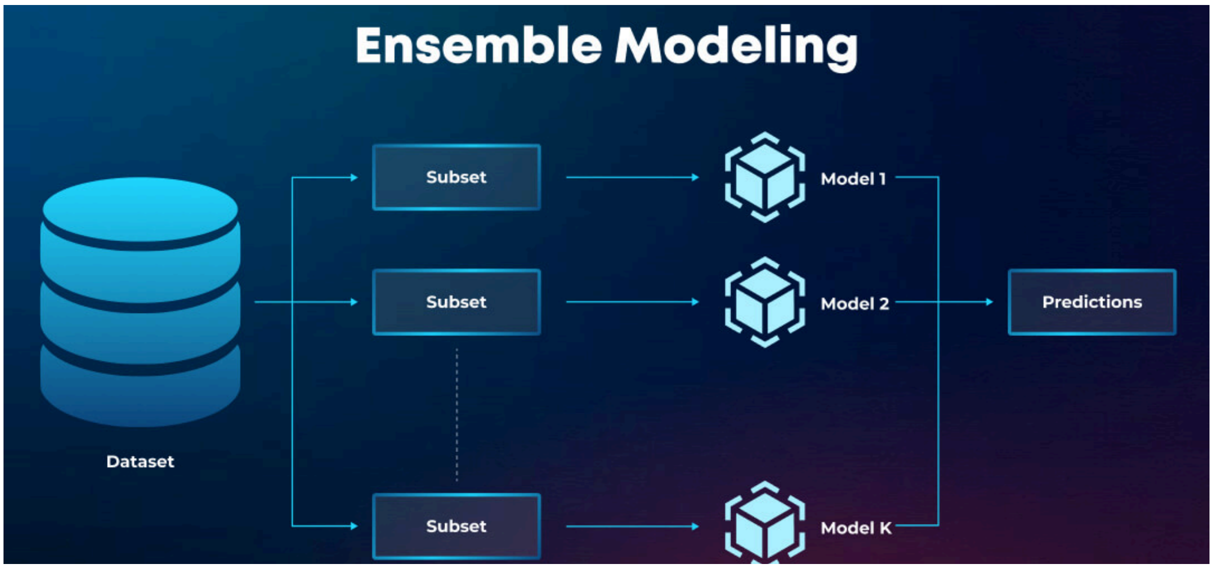
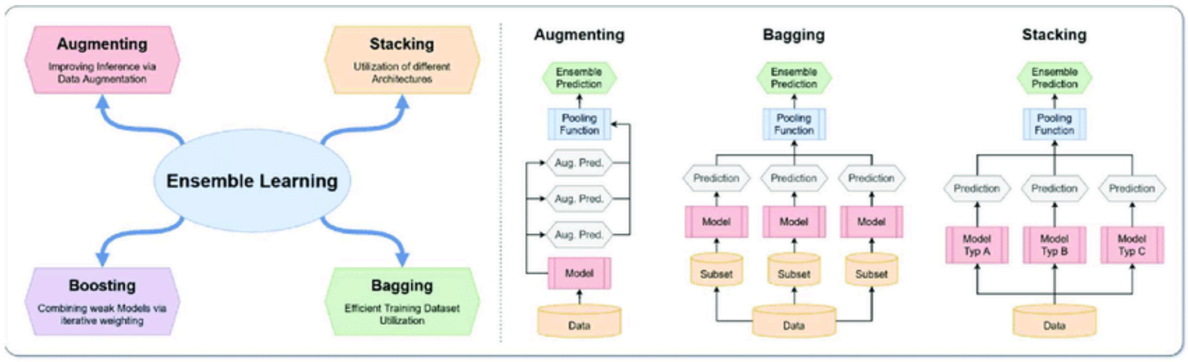


Modules 5: Ensemble Learning

> **Ensemble learning** is a method that combines multiple machine learning models to increase prediction accuracy and stability. Core idea: many weak learners combine to form a stronger learner.

🎯 I. Ensemble Learning Overview



Basic Concepts

Ensemble learning typically uses **Decision Trees** as base models to combine multiple models, thereby increasing accuracy and reducing overfitting.

Two Basic Ensemble Types

📊 **Averaging (Regression)**

- Applied to regression problems
- Averages predictions from multiple models
- **Goal**: Reduce variance

🗳️ **Voting (Classification)**

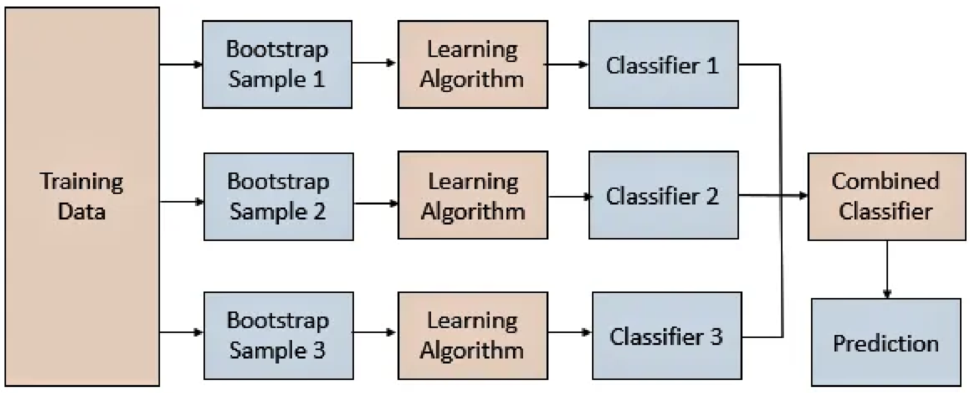
- Applied to classification problems
- Each model votes → most votes wins
- **Goal**: Reduce variance & increase stability

📋 Voting Example

Sample	Model 1	Model 2	Model 3	Voting Result
Email E1	Spam	Spam	Not spam	**Spam** ✅
Email E2	Not spam	Not spam	Not spam	**Not spam** ✅

→ Combining multiple models helps reduce individual model errors.

🌳 II. Bagging (Bootstrap Aggregation)



Theory

💡 **Bagging = Bootstrap Aggregation**
Randomly sample **with replacement** (bootstrap) to create multiple subsets from original data, train independent models, then combine results.

Process Flow

1. **Bootstrap Sampling**: Take bootstrap samples from original data (n samples, randomly selected with replacement)

2. **Independent Training**: Train separate models (usually Decision Trees) on each subset
3. **Aggregation**: Combine results (average or majority voting)

📐 Key Formulas

Regression (Averaging):

```
ŷ = (1/M) × Σ fᵢ(x)
```

Where:

- **ŷ**: final predicted value
- **M**: total number of models
- **$f_i(x)$**: prediction from model i

Classification (Voting):

```
ŷ = mode{f₁(x), f₂(x), ..., fₘ(x)}
```

Select the class with the most "votes" from all models.

Variance Reduction Formula:

```
Var(f̂) = s² × [(1-ρ)/M + ρ]
```

Where:

- **s^2**: variance of a single model
- **ρ**: correlation coefficient between models (0 to 1)
- **M**: number of models

🧮 Example 1: Averaging (Regression)

3 models predict for the same point x:

Model	Prediction
Model 1	$f_1(x) = 2.5$
Model 2	$f_2(x) = 3.1$
Model 3	$f_3(x) = 2.8$

Calculate average:

```
ŷ = (2.5 + 3.1 + 2.8) / 3 = 8.4 / 3 = 2.8
```

✅ **Result**: Bagging prediction = **2.8**

🧮 Example 2: Variance Reduction

Given:

- Single model variance: $s^2 = 4$
- Correlation coefficient: $\rho = 0.2$
- Number of models: $M = 10$

Calculation:

```
Var = 4 × [(1-0.2)/10 + 0.2]
    = 4 × [0.8/10 + 0.2]
    = 4 × [0.08 + 0.2]
    = 4 × 0.28
    = 1.12
```

📉 Variance reduced from **4** (1 tree) to **1.12** (10 trees) = **72% reduction!**

🧮 Example 3: Out-of-Bag Probability

Probability that a sample is NOT selected in bootstrap:

$$P = (1 - 1/n)^n \approx e^{-1} \approx 0.368$$

→ About **36.8%** of samples are not used in 1 tree → used to estimate **OOB error**.

💻 Code Example

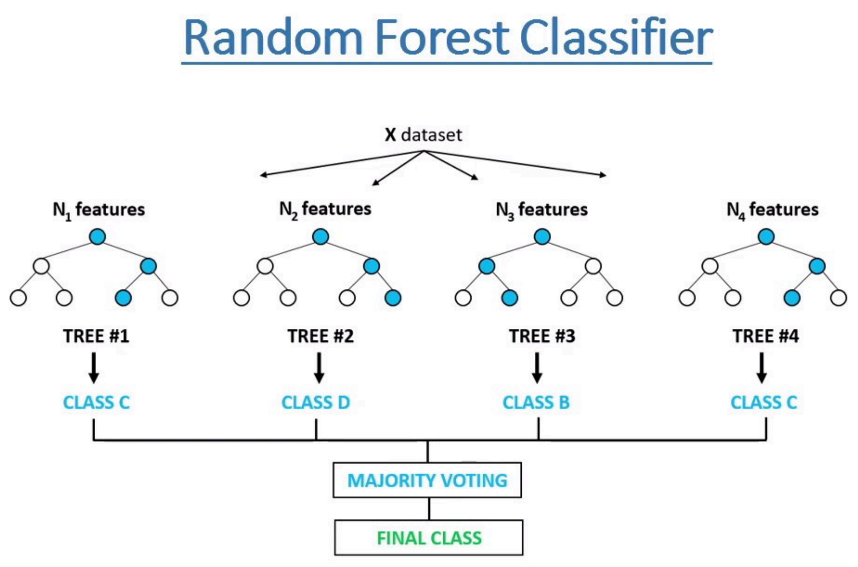
```
from sklearn.ensemble import BaggingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load data
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, random_state=42
)

# Create Bagging Model
bag = BaggingClassifier(
    base_estimator=DecisionTreeClassifier(),
    n_estimators=50,
    random_state=42
)

# Train and evaluate
bag.fit(X_train, y_train)
y_pred = bag.predict(X_test)
print("Bagging Accuracy:", accuracy_score(y_test, y_pred))
```

III. Random Forest



Theory

🎲 **Random Forest** is an improved version of Bagging that adds randomness when selecting features at each tree node.
Goal: reduce correlation between trees, increase accuracy

Process Flow

1. **Bootstrap Sampling**: Create multiple bootstrap samples
2. **Random Feature Selection**: At each tree node:
    - Randomly select **k features** from total d features
    - Find best split among those k features
3. **Aggregation**: Aggregate results (Voting or Averaging)

📐 Feature Selection Formula

Classification:

```
k = √d
```

(square root of total features)

Regression:

```
k = d/3
```

(1/3 of total features)

📊 Example

Assume data has **16 features** (d = 16):

```
k = √16 = 4
```

🎯 Each node only considers **4 random features** to find the best split.

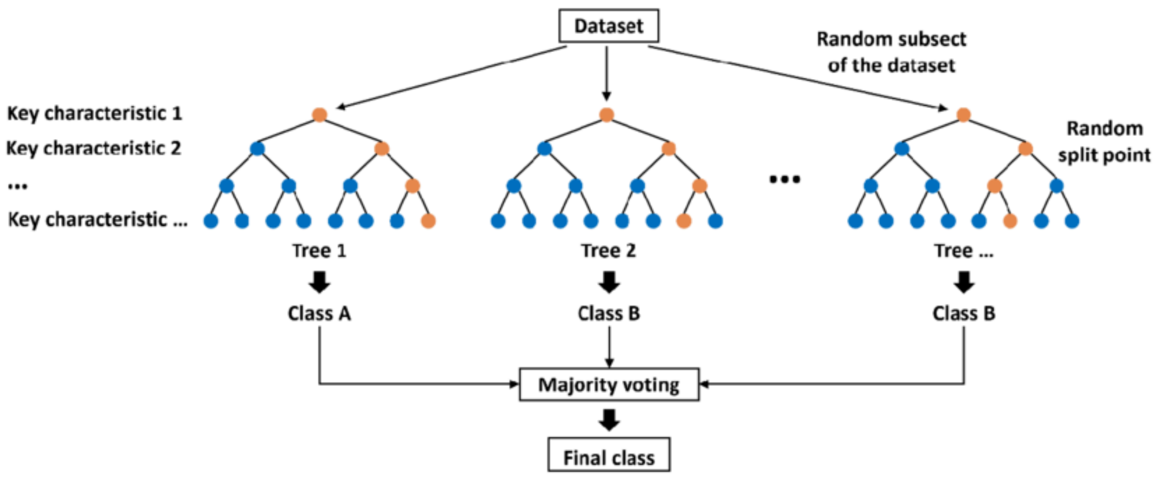
💻 Code Example

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(
    n_estimators=100,
    max_features='sqrt',  # Use square root
    random_state=42
)

rf.fit(X_train, y_train)
y_pred = rf.predict(X_test)
print("Random Forest Accuracy:", accuracy_score(y_test, y_pred))
```

🌴 IV. Extra Trees (Extremely Randomized Trees)



Theory

⚡ Extra Trees is similar to Random Forest but with **two key differences**:

1. **No bootstrap** – trains on entire original dataset
2. **Random split points** – no optimization needed

→ **Result**: Faster, more random, usually reduces overfitting

🔍 Comparison: Random Forest vs Extra Trees

Criteria	Random Forest	Extra Trees
Data per tree	Bootstrap (with replacement)	Entire original dataset
Split selection	Optimal (best split)	Random
Speed	Slower ⏱️	Faster ⚡
Bias	Higher	Lower
Variance	Low	Very low

📊 Random Split Selection Example

Feature X has values: **{2, 5, 7, 12, 18}**

Random Forest 🌲

→ Calculates and selects best split

→ Example: **9.5**

Extra Trees 🌴

→ Randomly selects a value

→ Example: **8.1**

→ **Extra Trees** is faster and more random.

💻 Code Example

```
from sklearn.ensemble import ExtraTreesClassifier

et = ExtraTreesClassifier(
    n_estimators=100,
    random_state=42
)

et.fit(X_train, y_train)
y_pred = et.predict(X_test)
print("Extra Trees Accuracy:", accuracy_score(y_test, y_pred))
```

📊 V. Comprehensive Variance Reduction

Comparing 3 Methods

Assumptions:

- Single model variance: **$s^2 = 5$**
- Number of trees: **$M = 50$**
- Correlation coefficients:
  - Bagging: **$\rho = 0.5$**
  - Random Forest: **$\rho = 0.2$**
  - Extra Trees: **$\rho = 0.1$**

Formula: $\mathrm{Var}(\hat{f}) = s^2 \times [(1-\rho)/M + \rho]$

Method	Calculation	Result	% Reduction
Bagging 🌳	5 × [(1-0.5)/50 + 0.5] = 5 × 0.51	**2.55**	49%
Random Forest 🌲	5 × [(1-0.2)/50 + 0.2] = 5 × 0.216	**1.08**	78%
Extra Trees 🌴	5 × [(1-0.1)/50 + 0.1] = 5 × 0.118	**0.59**	**88%** 🏆

🏆 **Winner**: Extra Trees achieves the **strongest variance reduction** (88% vs single model)

🧮 VI. Gini Impurity – Detailed Example

Formula

$$G(t) = 1 - \Sigma p_k^2$$

Where p_k is the proportion of class k in the node.

📊 Calculation Example

Setup: Node has **10 samples** → 6 class A, 4 class B

Step 1: Gini before split

```
G_root = 1 - (0.6² + 0.4²)
       = 1 - (0.36 + 0.16)
       = 1 - 0.52
       = 0.48
```

Step 2: Gini after split

- **Left** (5 samples, 4A–1B):

  ```
  G_L = 1 - (0.8² + 0.2²) = 1 - 0.68 = 0.32
  ```

- **Right** (5 samples, 2A–3B):

  ```
  G_R = 1 - (0.4² + 0.6²) = 1 - 0.52 = 0.48
  ```

Step 3: Weighted average

```
G_split = (5/10) × 0.32 + (5/10) × 0.48 = 0.40
```

Step 4: Gini Gain

```
ΔG = 0.48 - 0.40 = 0.08
```

✅ This split **reduces impurity by 0.08** ⇒ selected if ΔG is highest among all possible splits

📋 VII. Summary Comparison

Feature	Bagging 🌳	Random Forest 🌲	Extra Trees 🌴
Data per tree	Bootstrap	Bootstrap	Full sample
Added randomness	Data only	Data + feature	Data + feature + split
Bias reduction	Negligible	Moderate	Significant ✅
Variance reduction	✅	✅✅	✅✅✅
Speed	Medium	Medium	Fastest ⚡
When to use	Model overfits	Many features	Need speed & high stability

💻 VIII. Combined Code for 3 Models

```
from sklearn.ensemble import (
    BaggingClassifier,
    RandomForestClassifier,
    ExtraTreesClassifier
)
from sklearn.tree import DecisionTreeClassifier
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

# Load data
X, y = load_iris(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, random_state=42
)

# Define 3 models
models = {
    "Bagging": BaggingClassifier(
        DecisionTreeClassifier(),
        n_estimators=50,
        random_state=42
    ),
    "Random Forest": RandomForestClassifier(
        n_estimators=100,
        random_state=42
    ),
    "Extra Trees": ExtraTreesClassifier(
        n_estimators=100,
        random_state=42
    )
}

# Train and evaluate
for name, model in models.items():
    model.fit(X_train, y_train)
    acc = accuracy_score(y_test, model.predict(X_test))
    print(f"{name}: {acc:.3f}")
```

🚀 IX. Boosting Methods

1. Boosting Overview

🔄 **Sequential Learning**: Train models sequentially, where each new model focuses on samples misclassified by previous ones.
Goal: Reduce **bias** (systematic error)

General formula:

$$F_m(x) = F_{m-1}(x) + \alpha_m \times h_m(x)$$

Where:

- **$F_m(x)$**: ensemble model after round m
- **α_m**: weight of model m
- **$h_m(x)$**: weak learner m

---

2. AdaBoost (Adaptive Boosting)

⚖️ **Base learner**: Decision stump (1-level decision tree)
Key idea: Increase weights for misclassified samples

Formulas

Model weight:

$$\alpha_t = 0.5 \times \ln[(1 - \varepsilon_t) / \varepsilon_t]$$

Where **ε_t** is the error rate of model t.

Sample weight update:

$$w_i(\text{new}) = w_i(\text{old}) \times e\text{^}(-\alpha_t \times y_i \times h_t(x_i))$$

💻 Code Example

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier

base = DecisionTreeClassifier(max_depth=1)
ada = AdaBoostClassifier(
    base_estimator=base,
    n_estimators=10,
    learning_rate=1.0
)

ada.fit(X_train, y_train)
print("AdaBoost Accuracy:", accuracy_score(y_test, ada.predict(X_test)))
```

📊 Weight Update Example

Sample	Correct/Wrong	Initial Weight	Updated Weight	Change
1	✓	0.25	0.25	-
2	✓	0.25	0.25	-
3	×	0.25	**0.40**	⬆️ **+60%**
4	✓	0.25	0.10	⬇️

🎯 Misclassified samples get **increased weights** so next model focuses more on them

3. Gradient Boosting

📉 **Key concept**: Builds models using **gradient descent** on the loss function
Each new model learns to correct the **residual errors** of previous models

Formula

$$F_m(x) = F_{m-1}(x) + \alpha \times h_m(x)$$

where $h_m(x)$ learns according to gradient:

$$h_m(x) \approx -\partial L(y, F(x)) / \partial F(x)$$

Common loss functions:

- **Regression**: $L = 0.5 \times (y - F(x))^2$
- **Classification**: Log-loss (binomial deviance)

💻 Code Example

```
from sklearn.ensemble import GradientBoostingClassifier

gb = GradientBoostingClassifier(
    n_estimators=100,
    learning_rate=0.1,
    max_depth=3,
    subsample=0.8,
    random_state=42
)

gb.fit(X_train, y_train)
print("Gradient Boosting Accuracy:",
      accuracy_score(y_test, gb.predict(X_test)))
```

🧮 Regression Example: Learning Process

Round	Model Prediction	True Value	Residual	Learning Rate	Update
1	$F_1(x) = 5$	$y = 8$	$r = 3$	-	-
2	$h_2(x) = 2.5$	-	-	$\alpha = 0.1$	$F_2(x) = 5 + 0.1 \times 2.5 =$ **5.25**
3	-	-	$r = 2.75$	-	Continue...

📈 Model gradually **converges** to true value through iterative residual learning

🏗️ X. Stacking (Stacked Generalization)

Theory

🎭 **Meta-learning approach**: Combines multiple different model types
Base model predictions become **input features** for a meta-model

Formula:

$$\hat{y} = f\_meta(h_1(x), h_2(x), ..., h_m(x))$$

Where:

- **$h_1, h_2, ..., h_m$**: base models (different types)
- **f_meta**: meta-learner (usually Logistic Regression)

💻 Code Example

```
from sklearn.ensemble import StackingClassifier, RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
```

```
# Define base models
estimators = [
    ('rf', RandomForestClassifier(n_estimators=50, random_state=42)),
    ('svm', SVC(probability=True))
]

# Stacking with meta-learner
clf = StackingClassifier(
    estimators=estimators,
    final_estimator=LogisticRegression()
)

clf.fit(X_train, y_train)
print("Stacking Accuracy:",
      accuracy_score(y_test, clf.predict(X_test)))
```

📊 How Stacking Works

Sample	Model 1 (RF)	Model 2 (SVM)	Meta Input	Meta Output	Final Result
1	0.6	0.7	[0.6, 0.7]	0.65	**Class 1** ✅
2	0.3	0.2	[0.3, 0.2]	0.25	**Class 0** ✅

🧠 Meta model learns **how to optimally combine** predictions from diverse base models

⚖️ XI. Comprehensive Method Comparison

Feature	Bagging 🌳	AdaBoost ⚖️	Gradient Boosting 📉	Stacking 🏗️
Mechanism	Parallel	Sequential (weights)	Sequential (residuals)	Meta-learning
Sample reweighting	❌	✅	✅ (via gradient)	❌
Main goal	↓ Variance	↓ Bias	↓ Bias	↑ Overall
Outlier sensitivity	Low 🟢	High 🔴	Medium 🟡	Depends on base
Overfitting risk	Low 🟢	Possible 🟡	Easy 🔴	Possible 🟡
Training speed	Fast ⚡	Medium	Slow ⏱️	Slow ⏱️
Typical examples	Random Forest	AdaBoost	XGBoost / LightGBM	AutoML systems

📚 XII. Key Formula Summary

📖 **Quick reference guide for all important formulas**

Content	Formula	Meaning
Averaging	$\hat{y} = (1/M) \times \Sigma f_i(x)$	Average predictions
Voting	$\hat{y} = \text{mode}\{f_i(x)\}$	Most frequent class
Variance reduction	$Var = s^2 \times [(1-\rho)/M + \rho]$	Variance reduction formula
OOB Probability	$(1 - 1/n)^n \approx 0.368$	Sample not selected probability
Gini Impurity	$1 - \Sigma p_k^2$	Impurity measure
Gini Gain	G_parent - avg(G_child)	Impurity reduction
Feature selection (RF)	$k = \sqrt{d}$ or $d/3$	Features per split

📝 Notation Guide

Symbol	Meaning	Range
s^2	Variance of single model	$[0, \infty)$
ρ	Correlation coefficient	[0, 1]
M	Number of models	Positive integer
p_k	Proportion of class k	[0, 1]
d	Total number of features	Positive integer
k	Selected features per split	[1, d]

✨ **End of Ensemble Learning Guide**
This comprehensive guide covers all major ensemble methods from basic concepts to advanced implementations. Practice with the code examples and experiment with different parameters to master these techniques!