Supervised Learning

Essential Guide to Prediction & Classification

Machine Learning for Smarter Innovation

BSc Innovation & Design Thinking

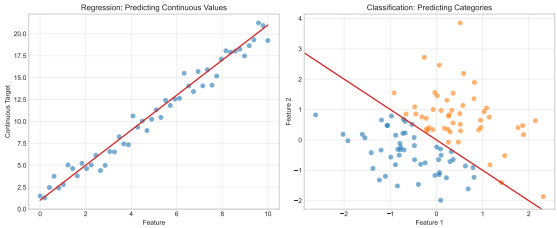
What is Supervised Learning?

Definition

- Learning from **labeled examples**
- Input features $X \rightarrow$ Output labels y
- Algorithm learns mapping: $f(X) \approx y$

Two Main Tasks

1. **Regression:** Predict continuous values
   - House prices, sales forecasts
2. **Classification:** Predict categories
   - Spam detection, medical diagnosis



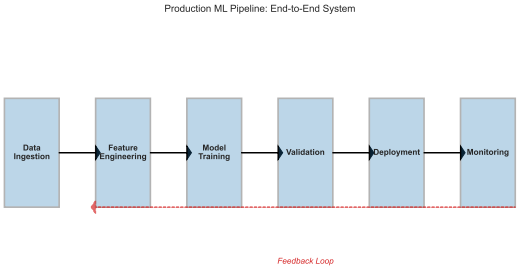
Real-World Examples

- Real estate pricing
- Email spam filtering
- Customer churn prediction
- Sales forecasting

Supervised learning transforms labeled historical data into predictive models - algorithms discover patterns from examples

The Supervised Learning Pipeline

1. **Training Phase**
   - Collect labeled data: $(X_1, y_1), ..., (X_n, y_n)$
   - Split: 70-80% training, 20-30% testing
   - Algorithm learns pattern: $\hat{f}(X)$
2. **Prediction Phase**
   - New input: X_{new}
   - Apply model: $\hat{y} = \hat{f}(X_{new})$
3. **Evaluation Phase**
   - Test on held-out data
   - Measure: Accuracy, RMSE, F1



Critical Rule

NEVER test on training data!

Why?

- Cannot measure generalization
- Overestimates performance

Train-test split prevents overfitting evaluation - testing on unseen data reveals true generalization

Linear Methods: The Foundation

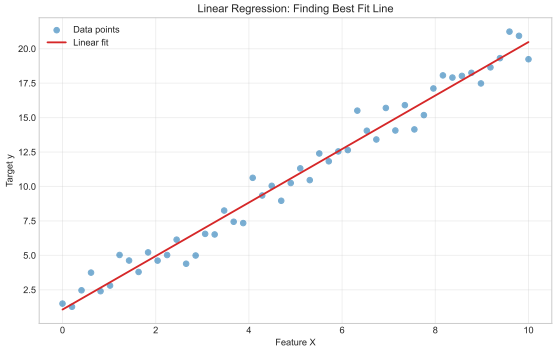
Ordinary Least Squares (OLS)
Model: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \epsilon$
Goal: Minimize squared errors

$$\min_{\beta} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

Solution: $\hat{\beta} = (X^T X)^{-1} X^T y$

Key Properties

- Fast to compute
- Interpretable coefficients
- Works for linear relationships



For Classification: Logistic Regression
Maps linear to probability:

$$p(y = 1|x) = \frac{1}{1 + e^{-(\beta_0 + \beta^T x)}}$$

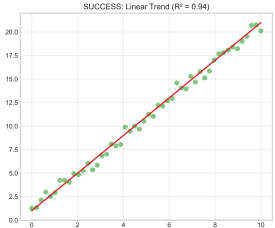
Decision boundary: $\beta_0 + \beta^T x = 0$

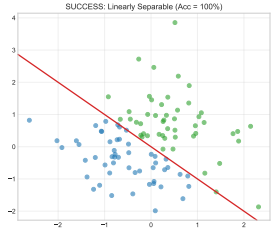
Linear methods form foundation - OLS for regression, logistic for classification

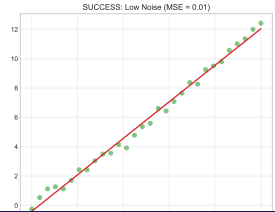
When Linear Methods Work (and When They Don't)

Success Cases

- Simple patterns: One feature predicts well
- Monotonic: More X $\rightarrow$ more Y
- Need interpretability
- Small datasets

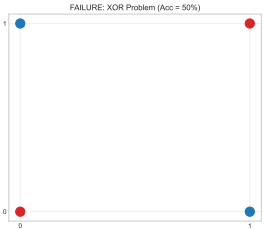


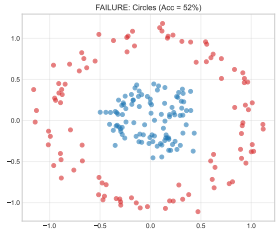


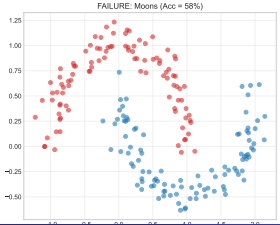


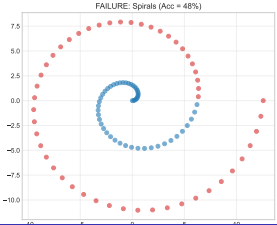
Failure Cases

- Curved patterns
- Feature interactions
- Multiple regions









Decision Trees: Human-Like Reasoning

Intuition: 20 Questions Game
Example: House Pricing
If sqft $>$ 2000?

- YES: If bedrooms $>$ 3? YES $\rightarrow$ \$520k, NO $\rightarrow$ \$450k
- NO: Predict \$280k

How It Works

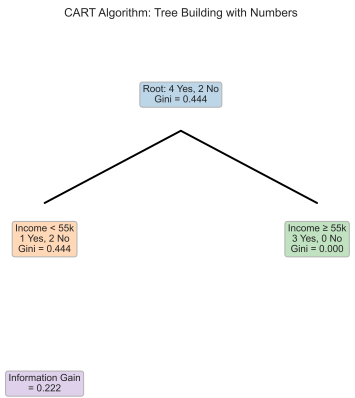
1. Find best split (minimize error)
2. Recursively split each group
3. Stop at leaves (predictions)

Advantages

- Interpretable (IF-THEN rules)
- Handles nonlinear patterns
- No scaling needed

Disadvantages

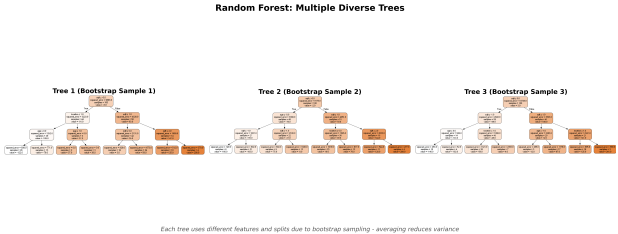
- Unstable, overfits easily

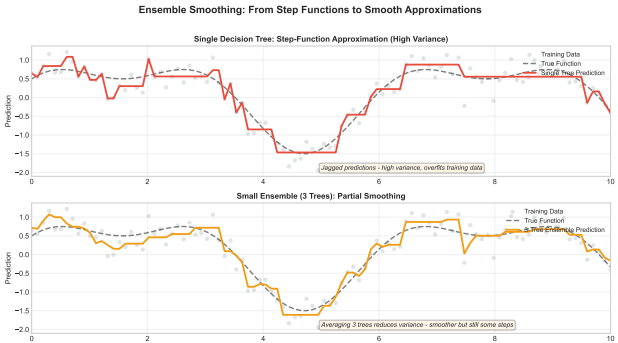


Solution: Use Ensembles
Combine many trees to stabilize predictions

Decision trees mimic human step-by-step reasoning - interpretable but unstable alone, requiring ensemble methods for robustness

Ensemble Methods: Many Trees Better Than One





Random Forest

- Build 100-500 trees
- Random data + features
- Average predictions

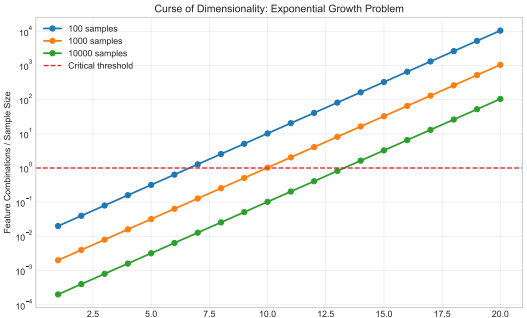
Gradient Boosting

- Sequential trees
- Correct previous errors
- Weighted sum

Insight: Single tree is jagged, many trees smooth
Tools: XGBoost, LightGBM

Common Pitfalls to Avoid

1. **Overfitting**
   - Memorizes training data
   - Fix: More data, cross-validation
2. **Data Leakage**
   - Test info in training
   - Fix: Strict separation
3. **Wrong Metrics**
   - Accuracy for imbalanced data
   - Fix: Use F1, AUC



4. **Too Many Features**
   - Curse of dimensionality
   - Fix: Feature selection
5. **Ignoring Domain**
   - Blind algorithm application
   - Fix: Feature engineering

Best Practices

- Split train/test first
- Cross-validate
- Sanity check predictions

Best Practices & Summary

Essential Workflow

1. **Start Simple**
   - Begin with OLS baseline
   - Add complexity only if needed
2. **Proper Validation**
   - 70-80% train, 20-30% test
   - Never touch test set until final
   - Use cross-validation for tuning
3. **Feature Engineering**
   - Handle missing values
   - Encode categorical variables
   - Scale features if needed
4. **Choose Algorithm**
   - Need interpretability? $\rightarrow$ Linear or tree
   - Need accuracy? $\rightarrow$ Random Forest or boosting
   - Small data? $\rightarrow$ Keep it simple

Algorithm Comparison

Method	Pros	Cons
Linear	Fast, interpretable	Assumes linearity
Decision Tree	Interpretable, non-linear	Unstable
Random Forest	Accurate, robust	Less interpretable
Boosting	Highest accuracy	Slow, complex

Key Takeaways

1. Supervised learning needs **labeled data**
2. Two tasks: **regression** vs **classification**
3. Always **train-test split**
4. Start **simple**, add complexity if needed
5. **Ensembles** achieve best accuracy

Supervised learning success requires proper methodology - split data, start simple, validate rigorously

Appendix A: Python Implementation - Trees & Random Forest

Decision Tree

```
from sklearn.tree import DecisionTreeRegressor

model = DecisionTreeRegressor(max_depth=5)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

Random Forest

```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_estimators=100, max_depth=10)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

For Classification
Just replace `Regressor` with `Classifier`

Scikit-learn provides simple, consistent API - fit, predict pattern works for all models

Appendix B: Gradient Boosting & Evaluation

Gradient Boosting

```
from sklearn.ensemble import GradientBoostingRegressor

model = GradientBoostingRegressor(n_estimators=100, learning_rate=0.1)
model.fit(X_train, y_train)
predictions = model.predict(X_test)
```

Evaluation

```
from sklearn.metrics import mean_squared_error, r2_score

mse = mean_squared_error(y_test, predictions)
rmse = mse ** 0.5
r2 = r2_score(y_test, predictions)

print(f"RMSE: {rmse:.2f}")
print(f"R-squared: {r2:.3f}")
```

Always evaluate on held-out test set - RMSE measures prediction error, R-squared measures explained variance