Supervised Learning

Essential Guide to Prediction & Classification

Machine Learning for Smarter Innovation

BSc Innovation & Design Thinking

What is Supervised Learning?

Definition

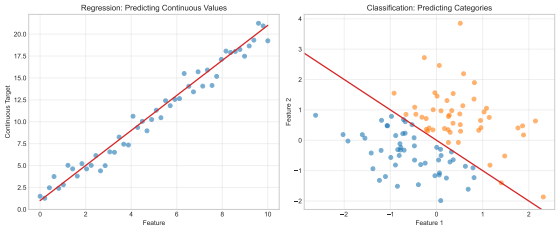
- Learning from **labeled examples**
- Input features $X \rightarrow$ Output labels y
- Algorithm learns mapping: $f(X) \approx y$

Two Main Tasks

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

Key Insight

Model learns patterns from historical data to predict future outcomes



Real-World Examples

- Real estate pricing (regression)
- Email spam filtering (classification)
- Customer churn prediction (classification)
- Sales forecasting (regression)

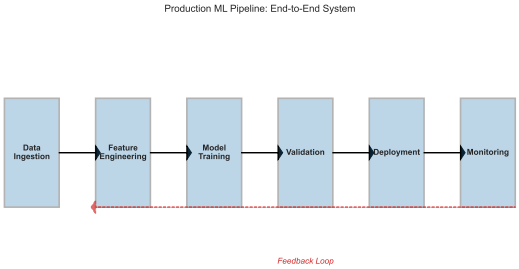
Supervised learning transforms labeled historical data into predictive models - the algorithm discovers patterns from examples rather than explicit programming

The Supervised Learning Pipeline

1. **Training Phase**
   - Collect labeled data: $(X_1, y_1), (X_2, y_2), ..., (X_n, y_n)$
   - Split: 70-80% training, 20-30% testing
   - Algorithm learns pattern: $\hat{f}(X)$
   - Minimize error on training set

2. **Prediction Phase**
   - Receive new unlabeled input: X_{new}
   - Apply learned model: $\hat{y} = \hat{f}(X_{new})$
   - Generate prediction

3. **Evaluation Phase**
   - Test on held-out data
   - Measure: Accuracy, RMSE, F1-score
   - Validate generalization



Critical Rule

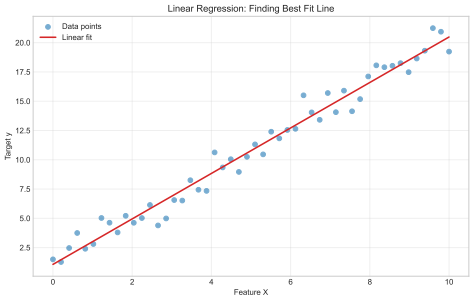
NEVER test on training data!

Why?

- Model has already seen training data
- Cannot measure true generalization
- Leads to overestimation of performance

Train-test split prevents overfitting evaluation - testing on unseen data reveals true generalization performance rather than mere memorization

Linear Methods: The Foundation



Ordinary Least Squares (OLS)

- Model: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \epsilon$
- Goal: Minimize squared errors
- Solution: $\hat{\beta} = (X^T X)^{-1} X^T y$
- Fast, interpretable

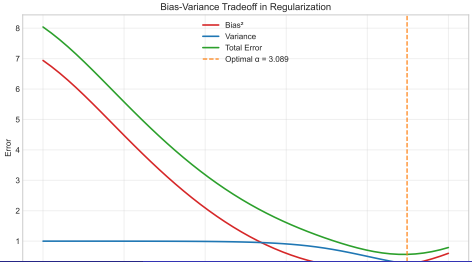
Regularization Methods

Ridge (L2):

- Penalizes large coefficients: $\lambda ||\beta||_2^2$
- Shrinks all coefficients smoothly
- Prevents overfitting

Lasso (L1):

- Sparse penalty: $\lambda ||\beta||_1$



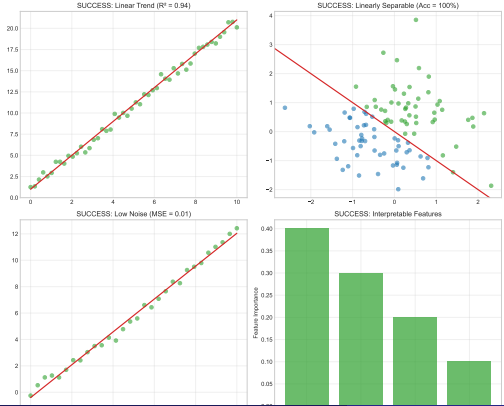
When Linear Methods Work (and When They Don't)

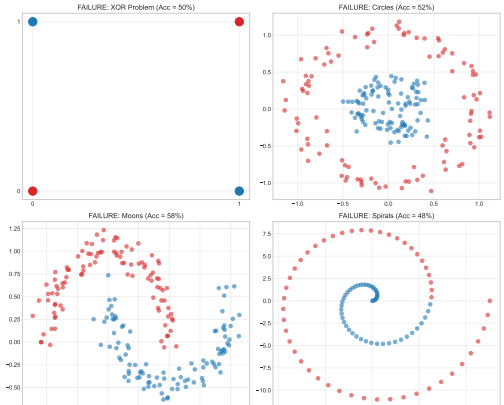
Success Cases

- **Simple relationships:** Monotonic, few features
- **Interpretability critical:** Regulatory requirements
- **Fast prediction needed:** Real-time systems
- **Limited data:** Few training examples

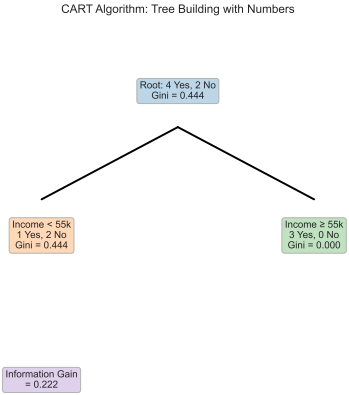
Failure Cases

- **Complex patterns:** Nonlinear relationships
- **Feature interactions:** XOR-like problems
- **High-dimensional:** Many correlated features
- **Mixed data types:** Categorical + continuous





Decision Trees: Interpretable Nonlinear Models



How It Works

- Recursive binary splits
- Like playing "20 questions"
- At each node: Choose best feature + threshold
- Leaves contain predictions

Splitting Criterion

Regression (minimize variance):

$$\text{Gain} = \text{Var}(\text{parent}) - \sum \text{Var}(\text{children})$$

Classification (Gini or Entropy):

$$\text{Gini} = 1 - \sum p_i^2$$

Advantages

- Highly interpretable (extract rules)
- Handles mixed data types

Disadvantages

- Prone to overfitting (memorizes noise)
- High variance (unstable to data changes)

Ensemble Methods: Power Through Combination

Random Forest

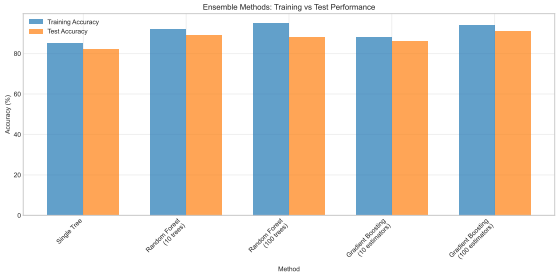
- Build T trees (e.g., 100-500)
- Each tree sees:
  - Random subset of data (bootstrap)
  - Random subset of features
- Prediction: Average (regression) or vote (classification)
- **Effect:** Reduces variance

Gradient Boosting

- Build trees sequentially
- Each tree corrects errors of previous
- Final: Weighted sum of all trees
- **Effect:** Reduces bias

Modern Implementations

- XGBoost, LightGBM, CatBoost
- State-of-art for tabular data



Trade-offs

Advantages:

- Best accuracy for tabular data
- Robust to overfitting
- Handles mixed data types
- Minimal feature engineering

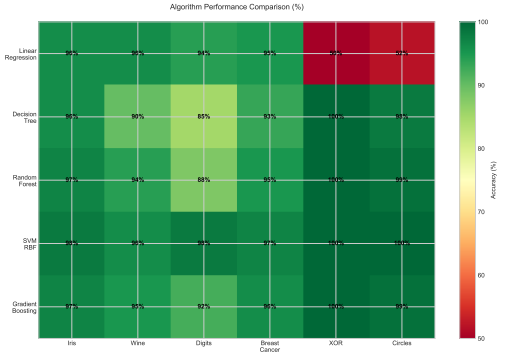
Disadvantages:

- Less interpretable (black box)
- Slower training and prediction

Algorithm Selection Guide

Decision Flowchart

1. **Is interpretability critical?**
   - YES → Linear methods or single tree
   - NO → Consider accuracy needs
2. **Is the relationship linear?**
   - YES → OLS, Ridge, Lasso, Logistic
   - NO → Nonlinear methods needed
3. **Do you need highest accuracy?**
   - YES → Random Forest, XGBoost
   - NO → Single tree or linear
4. **How much data do you have?**
   - Small (<1000) → Linear, regularization
   - Medium (1000-100k) → Trees, forests
   - Large (>100k) → Gradient boosting, deep learning
5. **What are your feature types?**
   - All numeric → Any method
   - Mixed (numeric + categorical) → Trees preferred



General Strategy

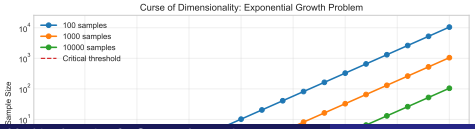
1. **Start simple:** Linear baseline
2. **Add complexity:** If performance insufficient
3. **Monitor trade-off:** Interpretability vs accuracy
4. **Cross-validate:** Always test generalization

Common Pitfalls to Avoid

1. **Overfitting**
   - Model memorizes training data
   - High training accuracy, poor test accuracy
   - **Fix:** Regularization, cross-validation, more data

2. **Underfitting**
   - Model too simple for patterns
   - Poor performance on both train and test
   - **Fix:** Add complexity, more features, nonlinear methods

3. **Curse of Dimensionality**
   - Too many features vs samples
   - Distance metrics become meaningless
   - **Fix:** Feature selection, dimensionality reduction, regularization



4. **Data Leakage**
   - Test information leaks into training
   - Examples:
     - Using future data to predict past
     - Including target in features
     - Normalizing before splitting
   - **Fix:** Strict train-test separation, careful preprocessing

5. **Imbalanced Classes**
   - Rare positive class (e.g., 1% fraud)
   - Model predicts majority class always
   - **Fix:** Resampling, cost-sensitive learning, different metrics (F1, AUC)

6. **Not Testing Generalization**
   - Evaluating only on training data
   - **Fix:** Always use held-out test set or cross-validation

Best Practices & Summary

Essential Best Practices

1. Data Splitting

- Always split train (70-80%) vs test (20-30%)
- Use cross-validation for hyperparameter tuning
- Never touch test set until final evaluation

2. Start Simple

- Begin with linear baseline
- Understand performance ceiling
- Add complexity incrementally

3. Feature Engineering

- Domain knowledge matters
- Handle missing values
- Encode categorical variables
- Scale/normalize features

4. Model Validation

- Monitor train vs test performance
- Use appropriate metrics (RMSE, accuracy, F1, AUC)
- Check predictions make sense

Algorithm Summary

Method	Strengths	Weaknesses
Linear (OLS, Ridge, Lasso)	Fast, interpretable, stable	Assumes linearity
Logistic Regression	Probabilistic, interpretable	Linear boundaries
Decision Tree	Interpretable, nonlinear	High variance
Random Forest	Accurate, robust	Less interpretable
Gradient Boosting	Highest accuracy	Slow, many parameters

Key Takeaways

1. Supervised learning requires **labeled data**
2. Two tasks: **regression** (continuous) vs **classification** (discrete)
3. Always use **train-test split**
4. Start **simple**, add complexity only if needed
5. Monitor **interpretability-accuracy** tradeoff
6. **Ensembles** achieve best accuracy for tabular data