Supervised Learning: Ensemble Methods

Random Forests, Bagging, Boosting (AdaBoost, Gradient Boosting)

Sarwan Ali

Department of Computer Science
Georgia State University

Combining Learners for Better Performance

Today's Learning Journey

What are Ensemble Methods?

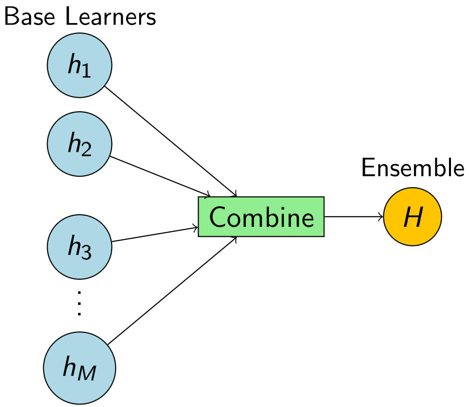
Definition: Combine multiple learning algorithms to create a stronger predictor than any individual learner alone.

Key Principle: *"Wisdom of the crowd"*

- Multiple weak learners $\rightarrow$ Strong learner
- Reduce overfitting and variance
- Improve generalization

Mathematical Foundation:

$$\hat{y} = f_{ensemble}(x) = \text{Combine}(f_1(x), f_2(x), \ldots, f_M(x))$$



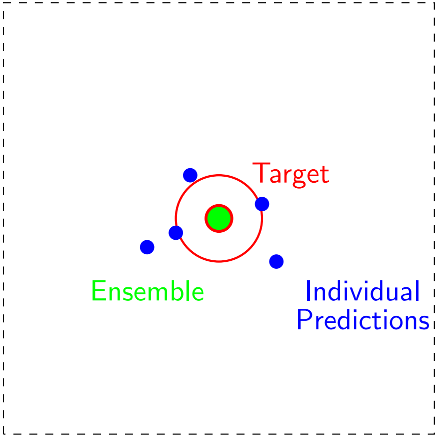
Why Do Ensembles Work?

Bias-Variance Decomposition:

$$E[(y - \hat{f}(x))^2] = \text{Bias}^2 + \text{Variance} + \text{Noise}$$

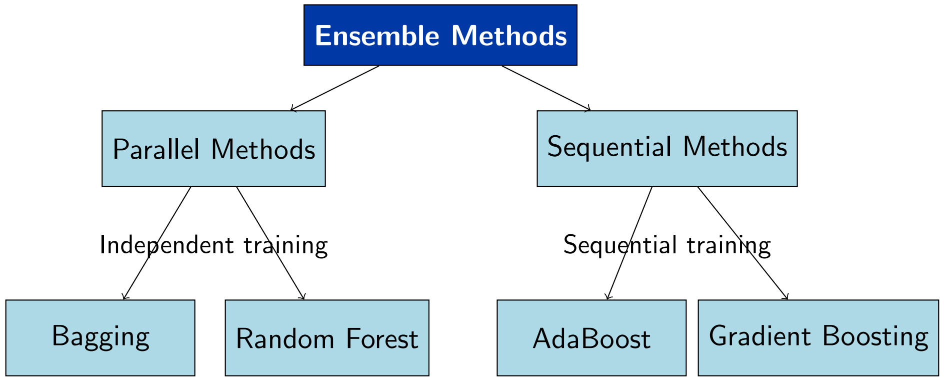
Benefits:

- **Reduce Variance**: Averaging reduces fluctuations
- **Reduce Bias**: Sequential correction of errors
- **Improve Robustness**: Less sensitive to outliers



Feature Space

Types of Ensemble Methods



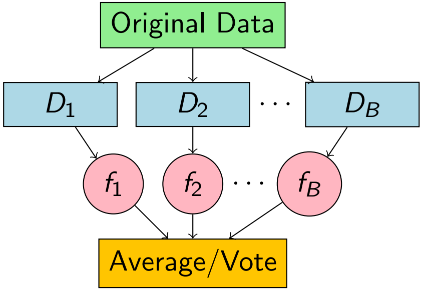
Bootstrap Aggregating (Bagging)

Key Idea: Train multiple models on different bootstrap samples

Algorithm:

1. For $b = 1, 2, \dots, B$:
    - Draw bootstrap sample D_b from training set D
    - Train model f_b on D_b
2. Combine predictions:
    - **Regression:** $\hat{y} = \frac{1}{B}\sum_{b=1}^{B} f_b(x)$
    - **Classification:** Majority vote

Bootstrap Sample: Sample n observations with replacement from original dataset of size n



Bagging: Mathematical Analysis

Variance Reduction: For independent models with variance σ^2: $\text{Var}\left(\frac{1}{B}\sum_{b=1}^{B} f_b(x)\right) = \frac{\sigma^2}{B}$

With Correlation ρ:

$$\text{Var}\left(\frac{1}{B}\sum_{b=1}^{B} f_b(x)\right) = \rho\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

Key Insights

- As $B \to \infty$, variance approaches $\rho\sigma^2$
- Lower correlation ρ leads to better variance reduction
- **Goal:** Create diverse, uncorrelated models

Out-of-Bag (OOB) Error: Use samples not in bootstrap for validation

$$P(\text{observation not selected}) = \left(1 - \frac{1}{n}\right)^n \approx \frac{1}{e} \approx 0.37$$

Random Forests: Enhanced Bagging

Key Innovation: Add feature randomness to bagging

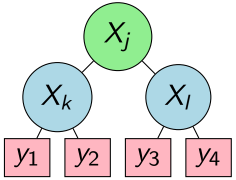
Algorithm:

1. For $b = 1, 2, \dots, B$:
    - Draw bootstrap sample D_b
    - At each split in tree:
        - Randomly select m features ($m < p$)
        - Find best split among these m features
    - Grow tree fully (no pruning)
2. Combine via averaging/voting

Feature Selection:

- **Classification:** $m = \sqrt{p}$
- **Regression:** $m = p/3$

Random subset of features



Features: $\{X_1, X_2, \dots, X_p\}$

Random m: $\{X_j, X_k, X_l\}$

Random Forests: Advantages and Properties

Advantages:

- ✓ Handles large datasets efficiently
- ✓ Robust to outliers and noise
- ✓ Provides feature importance
- ✓ No overfitting with more trees
- ✓ Handles missing values
- ✓ Works for both classification and regression

Hyperparameters:

- **n_estimators**: Number of trees (B)
- **max_features**: m (features per split)
- **max_depth**: Tree depth limit
- **min_samples_split**: Min samples to split
- **bootstrap**: Use bootstrap sampling

Feature Importance:

$$\text{Importance}(X_j) = \frac{1}{B} \sum_{b=1}^{B} \sum_{t \in T_b} I(v(t) = j) \cdot p(t) \cdot \Delta$$

Best Practices

- Start with default parameters
- Increase trees until OOB error stabilizes
- Tune max_features for your problem

Introduction to Boosting

Key Idea: Sequentially learn from mistakes of previous models

Philosophy:

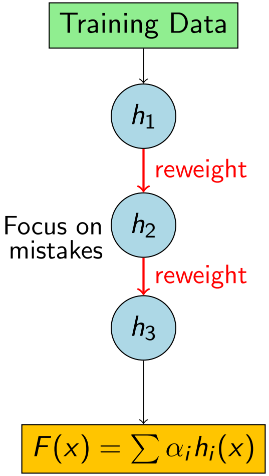
- Focus on *hard-to-classify* examples
- Each model corrects errors of previous ones
- Combine weak learners into strong learner

General Framework:

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

where:

- $F_m(x)$: Ensemble after m iterations
- $h_m(x)$: m-th weak learner
- α_m: Weight of m-th learner



AdaBoost (Adaptive Boosting)

Algorithm:

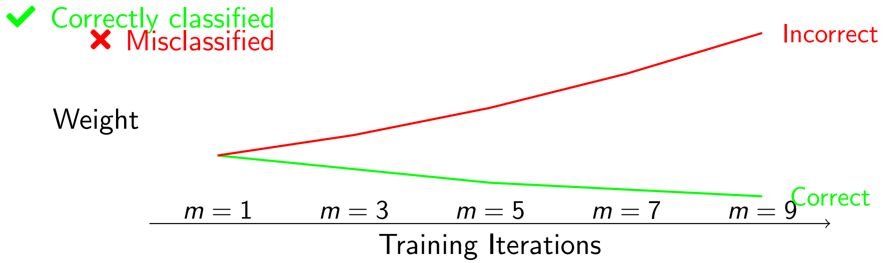
1. Initialize weights: $w_i^{(1)} = \frac{1}{n}$ for $i = 1, \ldots, n$
2. For $m = 1, 2, \ldots, M$:
    1. Train weak learner h_m on weighted dataset
    2. Calculate weighted error: $\epsilon_m = \sum_{i=1}^{n} w_i^{(m)} \mathbf{1}[y_i \neq h_m(x_i)]$
    3. Calculate learner weight: $\alpha_m = \frac{1}{2} \log \left(\frac{1-\epsilon_m}{\epsilon_m} \right)$
    4. Update sample weights:

    $$w_i^{(m+1)} = w_i^{(m)} \exp(-\alpha_m y_i h_m(x_i)) / Z_m$$

    where Z_m is normalization constant
3. Final classifier: $H(x) = \text{sign} \left(\sum_{m=1}^{M} \alpha_m h_m(x) \right)$

Key Insight: Misclassified examples get higher weights, correctly classified get lower weights

AdaBoost: Weight Updates Visualization



Learner Weight Relationship:

- If $\epsilon_m < 0.5$ (better than random): $\alpha_m > 0$
- If $\epsilon_m = 0.5$ (random): $\alpha_m = 0$
- If $\epsilon_m > 0.5$ (worse than random): $\alpha_m < 0$

Gradient Boosting

Key Idea: Fit new models to residuals of previous models

Algorithm:

1. Initialize: $F_0(x) = \arg\min_\gamma \sum_{i=1}^{n} L(y_i, \gamma)$
2. For $m = 1, 2, \dots, M$:
   1. Compute negative gradients (pseudo-residuals):
   $$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F=F_{m-1}}$$
   2. Train weak learner h_m to predict r_{im}
   3. Find optimal step size:
   $$\alpha_m = \arg\min_\alpha \sum_{i=1}^{n} L(y_i, F_{m-1}(x_i) + \alpha h_m(x_i))$$
   4. Update: $F_m(x) = F_{m-1}(x) + \alpha_m h_m(x)$

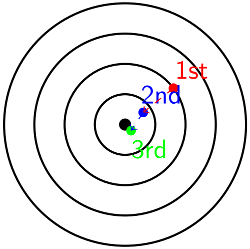
Common Loss Functions:

- **Regression:** Squared loss: $L(y, F) = \frac{1}{2}(y - F)^2$
- **Classification:** Logistic loss: $L(y, F) = \log(1 + \exp(-yF))$

Gradient Boosting: Intuitive Understanding

Analogy: Like learning to play darts

- **First throw:** Aim at center, miss
- **Second throw:** Aim at previous miss
- **Third throw:** Aim at new miss
- **Continue:** Each throw corrects previous errors



Each throw corrects previous error

Mathematical Perspective:

$$\text{New Model} = \text{Old Model} + \text{Learning Rate} \times \text{Error Correction}$$

Bagging vs. Boosting Comparison

Training	Parallel (independent)	Sequential (dependent)
Focus	Reduce variance	Reduce bias
Sampling	Bootstrap sampling	Weighted sampling
Base Learners	Strong learners (deep trees)	Weak learners (stumps)
Overfitting	Less prone	More prone
Noise Sensitivity	Robust	Sensitive
Scalability	Highly parallelizable	Sequential only
Examples	Random Forest	AdaBoost, XGBoost

Key Takeaway

Bagging: Good when you have high variance models (overfitting)
Boosting: Good when you have high bias models (underfitting)

When to Use Each Method

Use Random Forest when:

- ✓ Need interpretable feature importance
- ✓ Have noisy data with outliers
- ✓ Want robust, stable performance
- ✓ Need fast training/prediction
- ✓ Have mixed data types

Typical Performance:

- Good baseline performance
- Consistent across datasets
- Minimal hyperparameter tuning

Use Gradient Boosting when:

- ✓ Need highest possible accuracy
- ✓ Have clean, well-preprocessed data
- ✓ Can afford longer training time
- ✓ Have expertise for tuning
- ✓ Competition/production setting

Typical Performance:

- Often best single-model performance
- Requires careful tuning
- Prone to overfitting

Hyperparameter Tuning Guidelines

Random Forest:

- **n_estimators**: 100-500 (more is better)
- **max_features**: $\sqrt{p}$ or $\log_2(p)$
- **max_depth**: 10-20 or None
- **min_samples_split**: 2-10

Gradient Boosting:

- **n_estimators**: 100-1000
- **learning_rate**: 0.01-0.3
- **max_depth**: 3-8 (shallow trees)
- **subsample**: 0.8-1.0

Tuning Strategy:

1. Start with default parameters
2. Use cross-validation
3. Grid search or random search
4. Monitor validation curves

Important Trade-offs

- **Learning rate vs. n_estimators**: Lower rate needs more estimators
- **Tree depth vs. regularization**: Deeper trees need more regularization
- **Training time vs. accuracy**: More complex models take longer

Modern Ensemble Extensions

XGBoost (Extreme Gradient Boosting):

- Optimized gradient boosting
- Built-in regularization
- Parallel processing
- Missing value handling
- Feature importance

LightGBM:

- Leaf-wise tree growth
- Faster training
- Lower memory usage
- Categorical feature support

CatBoost:

- Handles categorical features natively
- Reduces overfitting
- No hyperparameter tuning needed
- Robust to outliers

Stacking:

- Meta-learning approach
- Combines different algorithm types
- Uses cross-validation
- Higher complexity but better performance

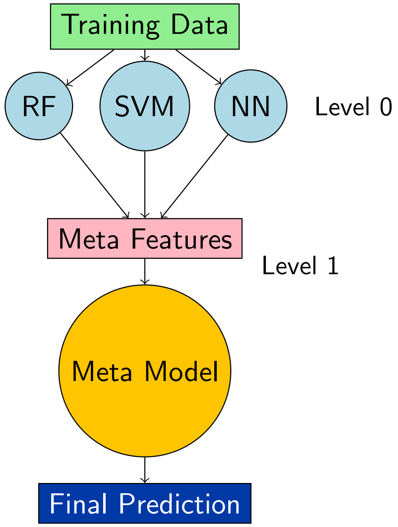
Stacking (Stacked Generalization)

Two-Level Architecture:

1. **Level 0 (Base Models):**
    - Train diverse algorithms (RF, SVM, NN)
    - Generate predictions using CV
2. **Level 1 (Meta-Model):**
    - Learn from base model predictions
    - Often simple: Linear/Logistic Regression

Algorithm:

1. Split data into K folds
2. For each base model:
    - Train on K-1 folds
    - Predict on held-out fold
3. Train meta-model on base predictions
4. Final prediction: Meta-model output



Ensemble Diversity and Model Selection

Importance of Diversity:

$$\text{Ensemble Error} = \bar{E} - \bar{A}$$

where $\bar{E}$ is average individual error and $\bar{A}$ is ambiguity (diversity measure)

Sources of Diversity:

- **Data:** Bootstrap, subsampling
- **Features:** Random subsets, PCA
- **Algorithms:** Different model types
- **Parameters:** Different hyperparameters
- **Objectives:** Different loss functions

Measuring Diversity:

- **Correlation:** Lower is better
- **Disagreement:** Higher is better
- **Q-statistic:** Measures pairwise diversity

Selection Strategies:

- Forward selection
- Backward elimination, Genetic algorithms

Key Principle

Accuracy-Diversity Trade-off: Need balance between individual model accuracy and ensemble diversity

Python Implementation: Random Forest

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
import numpy as np
# Load and prepare data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
# Create and train Random Forest
rf = RandomForestClassifier(n_estimators=100, max_features='sqrt',
    max_depth=10, min_samples_split=5, random_state=42,
    n_jobs=-1  # Use all processors)
# Train the model
rf.fit(X_train, y_train)
y_pred = rf.predict(X_test) # Make predictions
accuracy = accuracy_score(y_test, y_pred)
# Feature importance
feature_importance = rf.feature_importances_
```

Python Implementation: Gradient Boosting

```
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import GridSearchCV
# Define parameter grid
param_grid = {
    'n_estimators': [100, 200, 300], 'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 4, 5], 'subsample': [0.8, 0.9, 1.0]}
# Create gradient boosting classifier
gb = GradientBoostingClassifier(random_state=42)
# Grid search with cross-validation
grid_search = GridSearchCV(
    gb, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
# Fit and find best parameters
grid_search.fit(X_train, y_train)
best_gb = grid_search.best_estimator_
# Predictions
y_pred_gb = best_gb.predict(X_test)
```

Python Implementation: Advanced Ensembles

```
# XGBoost implementation
import xgboost as xgb
xgb_model = xgb.XGBClassifier(
    n_estimators=100,learning_rate=0.1,
    max_depth=6,
    subsample=0.8,
    colsample_bytree=0.8,
    random_state=42)
xgb_model.fit(X_train, y_train)
# Voting Classifier (Simple Ensemble)
from sklearn.ensemble import VotingClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
voting_clf = VotingClassifier(
    estimators=[('rf', RandomForestClassifier()),
        ('svm', SVC(probability=True)),('nb', GaussianNB()) ],
    voting='soft'  # Use predicted probabilities)
voting_clf.fit(X_train, y_train)
```

Ensemble Methods in Practice

Industry Applications:

- **Finance:** Credit scoring, fraud detection
- **Healthcare:** Disease diagnosis, drug discovery
- **E-commerce:** Recommendation systems
- **Transportation:** Route optimization
- **Environment:** Climate modeling

Kaggle Competition Winners:

- Most winners use ensemble methods
- Combination of XGBoost, LightGBM, Neural Networks
- Stacking is very common

Success Stories:

- **Netflix Prize:** Ensemble of 107 algorithms
- **KDD Cup:** Random Forests for customer churn
- **ImageNet:** Deep ensemble networks

Production Considerations:

- **Latency:** Single models faster than ensembles
- **Memory:** Ensembles require more storage
- **Interpretability:** Individual trees more interpretable
- **Maintenance:** More complex deployment pipeline

Case Study: Credit Risk Assessment

Problem: Predict loan default probability

Dataset Characteristics:

- 50,000 loan applications
- 20 features (income, age, credit history)
- Imbalanced: 5% default rate
- Mixed data types

Model Performance:

Model	**AUC**	**F1**
Logistic Regression	0.72	0.31
Random Forest	0.85	0.47
XGBoost	0.88	0.52
Ensemble (Stacking)	**0.91**	**0.58**

Implementation Strategy:

1. Data preprocessing and feature engineering
2. Train diverse base models:
   - Random Forest (handles mixed types)
   - XGBoost (high performance)
   - Logistic Regression (linear patterns)
3. Meta-model: Logistic Regression
4. Cross-validation for robust evaluation

Business Impact:

- 15% reduction in default losses
- Better risk-adjusted pricing
- Improved customer experience

Key Takeaways

Ensemble Benefits:

- ↑Improved accuracy over single models
- ↑Increased robustness and stability
- ⚖Better bias-variance trade-off
- ⚙Reduced overfitting risk

Method Selection Guide:

- **Quick baseline:** Random Forest
- **Maximum accuracy:** Gradient Boosting
- **Noisy data:** Bagging methods
- **Complex problems:** Stacking

Best Practices:

- Start simple, add complexity gradually
- Ensure diversity in base models
- Use proper cross-validation
- Monitor for overfitting
- Consider computational constraints

Common Pitfalls:

- Using identical base models
- Ignoring computational costs
- Over-tuning hyperparameters
- Not validating properly

Future Directions and Advanced Topics

Emerging Trends:

- **Deep Ensembles:** Neural network combinations
- **AutoML:** Automated ensemble selection
- **Online Learning:** Streaming ensemble updates
- **Federated Ensembles:** Distributed learning

Research Directions:

- Theoretical understanding of ensemble diversity
- Efficient ensemble pruning
- Uncertainty quantification
- Fairness in ensemble decisions

Tools and Libraries:

- **Scikit-learn:** Basic ensemble methods
- **XGBoost/LightGBM:** Gradient boosting
- **MLxtend:** Stacking implementations
- **H2O.ai:** AutoML ensembles

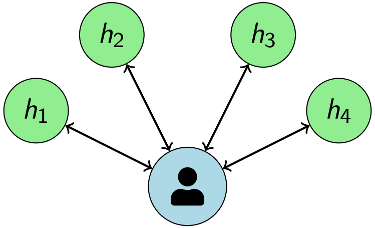
Next Steps:

- Practice with real datasets
- Experiment with different combinations
- Study competition solutions
- Understand your domain constraints

Thank You

Questions?

Remember: The wisdom of crowds often beats individual experts!



You + Ensemble Methods

Contact: sali85@student.gsu.edu