■ **Course:** CS109A: Introduction to Data Science

■ **Lecture:** Lecture 21: Bagging and Out-of-Bag Error

■ **Instructors:** Pavlos Protopapas, Kevin Rader, Chris Gumb

■ **Objective:** Understand ensemble learning through Bagging and learn how to use Out-of-Bag error for model validation

Contents

1 The Big Picture: Why We Need Bagging

Key Summary

This lecture introduces **Bagging** (**B**ootstrap **Agg**regat**ing**), a powerful technique that combines multiple decision trees to create a stronger, more reliable model. We'll also learn about **Out-of-Bag (OOB) error**, a clever way to evaluate our model without needing a separate validation set.

1.1 Where We Are in the Course

Let's recap what we've learned so far about decision trees:

1. **Classification trees**: Split data to maximize purity (using Gini index or entropy)
2. **Regression trees**: Split data to minimize MSE within each region
3. **Controlling complexity**: We use max depth, minimum samples per leaf, or pruning
4. **Cross-validation**: We use it to choose the best hyperparameters

But here's the problem: **single decision trees have limitations**.

1.2 The Fundamental Problem with Single Trees

Decision trees make **axis-aligned splits**—they can only draw vertical and horizontal lines to separate data. This means:

- If the true decision boundary is complex (like a circle), you need **many splits** to approximate it
- Many splits means a **deep tree**
- Deep trees tend to **overfit**

Example: Approximating a Circular Boundary

Imagine your data has two classes separated by a circular boundary. With decision trees:

- **Depth 1-2**: Makes a few horizontal/vertical cuts. Result: **severe underfitting**—the splits can't capture the circle at all
- **Depth 4-5**: Starts to form a rough rectangular approximation of the circle
- **Depth 20+**: Creates many small rectangles that start to look like a circle, but now you're **overfitting** to noise in your training data

The dilemma: go shallow and underfit, or go deep and overfit. There's no perfect depth!

Warning

The Bias-Variance Tradeoff in Trees:

- **Shallow trees**: High bias, low variance (underfit)
- **Deep trees**: Low bias, high variance (overfit)

Single trees struggle to achieve **both** low bias AND low variance.

1.3 The Key Insight: Multiple Opinions Are Better Than One

Think about how you make important decisions in real life:

Example: The Medical Second Opinion Analogy

Suppose you get an MRI scan and one doctor tells you that you need surgery. What do you do? Most people get a **second opinion**. Maybe even a third. Why?

- Each doctor might make mistakes (high variance)
- Different doctors were trained at different hospitals with different experiences
- By combining multiple opinions, individual errors tend to **cancel out**

If 3 doctors say "surgery needed" and 1 says "no surgery," you'd probably go with the majority. **This is exactly the idea behind ensemble learning!**

1.4 The Mathematical Insight: Averaging Reduces Variance

Here's a beautiful statistical fact that underlies all of ensemble learning:

Definition: The Power of Averaging

If you have n independent random variables, each with variance σ^2, then the variance of their average is:

$$\text{Var}\left(\frac{1}{n}\sum_{i=1}^{n} X_i\right) = \frac{\sigma^2}{n}$$

In plain English: **averaging reduces variance by a factor of** n.

This is a profound insight! It means:

- One tree might overfit and give a wrong prediction
- But if we train many trees and average their predictions, the errors wash out
- The key requirement: the trees need to be **somewhat independent** (make different errors)

Key Information

Historical Note: This idea traces back to Carl Friedrich Gauss in the late 1700s. When astronomers were trying to calculate planetary orbits, they initially thought they needed exactly 6 measurements for 6 unknowns. Gauss showed that **more measurements actually reduce error**—what seemed like "too much data" actually improved accuracy because random errors cancel out. This was one of the founding insights of statistics!

2 What is Bagging?

Definition: Bagging

Bagging stands for **B**ootstrap **Agg**regat**ing**. It's an ensemble method that:

1. Creates multiple different training datasets using **bootstrap sampling**
2. Trains a separate model (typically a deep decision tree) on each dataset
3. **Aggregates** their predictions (majority vote for classification, average for regression)

The name "Bagging" was coined by Leo Breiman in 1996 when he introduced this technique. It's a portmanteau of "Bootstrap" and "Aggregating."

2.1 Step 1: Bootstrap Sampling

The first challenge: we want to train multiple trees on **different** training data, but we only have **one** dataset. How do we create multiple datasets?

Definition: Bootstrap Sampling

Bootstrap sampling creates new datasets by sampling **with replacement** from the original data:

1. Start with original dataset of N samples
2. Randomly select N samples WITH REPLACEMENT (same sample can be picked multiple times)
3. This creates one "bootstrap sample"
4. Repeat to create as many bootstrap samples as you want

Example: Bootstrap in Action

Suppose your original dataset has 5 animals: {cow, cat, horse, dog, rat}
Three bootstrap samples might look like:

Sample	Contents
Bootstrap 1	dog, cat, dog, horse, horse
Bootstrap 2	rat, rat, cow, dog, cow
Bootstrap 3	dog, rat, horse, cat, rat

Notice:

- Bootstrap 1 has dog 3 times, horse 2 times, but NO rat or cow
- Bootstrap 2 has rat and cow twice each
- Each sample is the same SIZE as the original (5 animals), but with different compositions

Important: The 36.8% Rule

On average, each bootstrap sample will be missing about **36.8%** of the original data points. Why? The probability that a specific point is NOT chosen in any of the N draws is:

$$\left(1 - \frac{1}{N}\right)^N \approx e^{-1} \approx 0.368$$

This means roughly one-third of the data is "out of bag" (OOB) for each tree. We'll use this fact later!

2.2 Step 2: Train Multiple Trees

Once we have B bootstrap samples, we train one decision tree on each sample:

- **Tree 1** is trained on Bootstrap Sample 1
- **Tree 2** is trained on Bootstrap Sample 2
- ... and so on
- **Tree B** is trained on Bootstrap Sample B

Key Information

In bagging, we typically let each tree grow **deep** (high complexity). Why? Because even though deep trees overfit, the averaging process will reduce this overfitting. We're intentionally using low-bias, high-variance models and then reducing variance through aggregation.

2.3 Step 3: Aggregate Predictions

When a new data point comes in, we get predictions from ALL trees and combine them:

For Classification:

$$\hat{y} = \text{majority vote of } \{\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_B\}$$

Each tree "votes" for a class, and we pick the class with the most votes.

For Regression:

$$\hat{y} = \frac{1}{B}\sum_{b=1}^{B} \hat{y}_b$$

We simply average all the predictions.

Example: Complete Bagging Example

Task: Predict whether a patient has diabetes (classification)

Setup:

- Original training data: 1000 patients
- Number of trees: $B = 100$
- Tree depth: 20 (deep trees)

Training:

1. Create 100 bootstrap samples, each with 1000 patients (with replacement)
2. Train a depth-20 decision tree on each sample

Prediction for a new patient:

1. Send patient through all 100 trees
2. Suppose 73 trees predict "Diabetes" and 27 predict "No Diabetes"
3. Final prediction: "Diabetes" (majority vote)

The confidence might even be interpreted as 73%, giving us a probability-like output!

3 Why Bagging Works

3.1 Visualizing the Magic

Let's see how bagging reduces variance visually:

Example: Decision Boundaries at Different Depths

Consider a classification problem with a complex decision boundary.

With Single Trees:

- **Depth 2**: Decision boundary is too simple (underfit)
- **Depth 5**: Starting to capture the shape
- **Depth 100**: Captures detail but is extremely "wiggly" (overfit)

With Bagging (100 trees, depth 100):

- Each individual tree has a wiggly, overfit boundary
- But when we average/vote, the individual wiggles "cancel out"
- The average boundary is much smoother while still capturing the true pattern

It's like taking a photo with a shaky camera: one shot is blurry, but averaging 100 shots gives you a clearer image.

3.2 The Key Advantages of Bagging

1. **High Expressiveness**: Each tree can be deep, capturing complex patterns
2. **Low Variance**: Averaging multiple predictions reduces overall variance
3. **Reduced Overfitting**: Individual trees overfit differently; averaging cancels out
4. **Works with Any Base Model**: Though trees are most common, bagging works with any algorithm

Key Information

Bagging as Regularization:

Regularization is any technique that prevents overfitting. Usually we think of it as penalizing model complexity (like L1/L2 penalties). But bagging achieves regularization through a different mechanism: **averaging out the variance**. This makes bagging a form of "implicit regularization."

3.3 Can We Use Bagging with Other Models?

Yes! While bagging is most commonly used with decision trees, the concept applies to any model:

- **Bagged Linear Regression**: Train linear regression on bootstrap samples, average predictions
- **Bagged Neural Networks**: Train neural networks on bootstrap samples, average predictions
- **Bagged Any-Model**: Same principle applies

However, in practice:

- Bagging helps most when the base model has **high variance** (like deep trees)
- For already-stable models (like linear regression), the benefit is smaller

- For expensive models (like deep neural networks), the computational cost is prohibitive

4 Hyperparameters in Bagging

Bagging introduces new hyperparameters on top of the single-tree hyperparameters.

4.1 Single Tree Hyperparameters (Still Apply)

These control the complexity of each individual tree:

- **max_depth**: Maximum depth of each tree
- **min_samples_split**: Minimum samples needed to split a node
- **min_samples_leaf**: Minimum samples required in each leaf
- **criterion**: Gini, entropy (classification) or MSE (regression)

4.2 New Bagging Hyperparameter: Number of Estimators

The most important new hyperparameter is **n_estimators**—the number of trees in the ensemble.

Definition: Number of Estimators

The **number of estimators** (B or `n_estimators` in sklearn) is the number of trees in the bagging ensemble.
Key property: **More trees always helps (up to a point)** and **never hurts**!

Important: n_estimators Does NOT Control Complexity

This is a crucial distinction:

- **max_depth**: Controls bias-variance tradeoff (too much = overfit)
- **n_estimators**: Controls variance only (more = less variance)

Unlike max_depth, increasing n_estimators will NOT cause overfitting. It only reduces variance.

Example: How Error Changes with n_estimators

Training Error: As n_estimators increases, training error **goes up slightly**.
Why? With one deep tree, you can perfectly memorize the training data. But when you average many trees, some of that memorization washes out.
Validation Error: As n_estimators increases, validation error **goes down significantly**.
Why? The overfitting from individual trees cancels out, so the model generalizes better.
The validation error will decrease and then **flatten out**—it won't go back up!

4.3 The Hyperparameter Explosion Problem

With bagging, we need to tune:

- Splitting criterion (Gini vs. Entropy vs. MSE)
- Stopping condition (depth, min samples, etc.)
- Values for each stopping condition
- Number of estimators

If each has 5-10 options, we might have $3 \times 4 \times 10 \times 10 = 1,200$ combinations!

With 5-fold cross-validation, that's $1,200 \times 5 = 6,000$ models to train.

This is getting expensive! We need a better way to validate...

5 Out-of-Bag (OOB) Error

Key Summary

Out-of-Bag (OOB) error is a clever technique that uses the bootstrap's "waste" as a free validation set. It lets us evaluate our model without setting aside data for validation or performing expensive cross-validation.

5.1 The Key Insight

Remember: each bootstrap sample leaves out about 36.8% of the original data. This "left out" data is called the **Out-of-Bag (OOB)** data for that tree.

Definition: Out-of-Bag Data

For tree b, the **Out-of-Bag samples** are all data points from the original training set that were NOT included in bootstrap sample b.
Since the tree never saw these points during training, they act like a built-in validation set!

Example: OOB Samples Illustration

Original data: $\{x_1, x_2, x_3, x_4, x_5\}$

Tree	Bootstrap Sample	OOB Samples
Tree 1	$\{x_1, x_1, x_3, x_3, x_5\}$	$\{x_2, x_4\}$
Tree 2	$\{x_2, x_2, x_4, x_5, x_5\}$	$\{x_1, x_3\}$
Tree 3	$\{x_1, x_3, x_3, x_4, x_5\}$	$\{x_2\}$

Each tree has different OOB samples. We can use this to get a validation error for each data point!

5.2 Computing OOB Error

The OOB error calculation works as follows:

1. **For each data point x_i:**
   (a) Find all trees where x_i was OUT of the bootstrap sample
   (b) Get predictions from only those trees
   (c) Aggregate these predictions (majority vote or average)
   (d) Compare to the true label y_i
2. **Average the errors across all data points**

Definition: OOB Error Formula

For Classification:

Let $\hat{y}_i^{OOB}$ be the majority vote from trees that didn't see x_i:

$$\text{OOB Error} = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[\hat{y}_i^{OOB} \neq y_i]$$

For Regression:
Let $\hat{y}_i^{OOB}$ be the average prediction from trees that didn't see x_i:

$$\text{OOB Error} = \frac{1}{N}\sum_{i=1}^{N} (y_i - \hat{y}_i^{OOB})^2$$

Example: Step-by-Step OOB Calculation

Using our earlier example with 5 data points and 3 trees:

Computing OOB prediction for x_2:

- Tree 1: x_2 is OOB, so use Tree 1's prediction
- Tree 2: x_2 is IN the bootstrap, so skip
- Tree 3: x_2 is OOB, so use Tree 3's prediction

If Tree 1 predicts "Class A" and Tree 3 predicts "Class B":

- For classification: It's a tie! Typically we'd use more trees to break ties.
- For regression: Average the two predictions

Repeat for all data points, then compute overall error.

5.3 Why OOB Error is Great

1. **It's FREE:** No extra computation—we're using data that was "wasted" anyway
2. **No data splitting needed:** We don't have to set aside a validation set, so we can use ALL data for training
3. **Lower leakage than CV:** In cross-validation, each data point appears in training sets for $(K-1)$ folds. There's subtle information leakage. OOB has **zero leakage** because each prediction uses only trees that never saw that point.
4. **Automatically computed:** Sklearn computes it for you with `oob_score=True`

Important: OOB is for Validation

OOB error replaces your **validation set**. Use it for:

- Hyperparameter tuning
- Model selection
- Comparing different configurations

But you still need a **separate test set** for final, unbiased evaluation!

The workflow:

1. Split data: 80% train, 20% test (test goes in a vault)
2. Train bagging model with `oob_score=True`

3. Use OOB error to tune hyperparameters
4. Once happy, evaluate ONCE on the test set

6 Bagging in Python with sklearn

6.1 Basic Implementation

```python
from sklearn.ensemble import BaggingClassifier, BaggingRegressor
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor
from sklearn.model_selection import train_test_split

# Load your data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

# Create bagging classifier
bagging_clf = BaggingClassifier(
    estimator=DecisionTreeClassifier(max_depth=20),  # Base model
    n_estimators=100,      # Number of trees
    oob_score=True,        # Enable OOB scoring
    random_state=42,
    n_jobs=-1              # Use all CPU cores
)

# Train
bagging_clf.fit(X_train, y_train)

# Get OOB score (for validation)
print(f"OOB Accuracy: {bagging_clf.oob_score_:.4f}")

# Final test score
print(f"Test Accuracy: {bagging_clf.score(X_test, y_test):.4f}")
```

6.2 Key Parameters

Table 1: *Important BaggingClassifier/BaggingRegressor Parameters*

Parameter	Default	Description
`estimator`	None	Base estimator (uses DecisionTree if None)
`n_estimators`	10	Number of trees in ensemble
`max_samples`	1.0	Fraction of samples for each bootstrap
`max_features`	1.0	Fraction of features for each tree
`bootstrap`	True	Whether to use bootstrap sampling
`oob_score`	False	Whether to compute OOB error
`n_jobs`	None	Number of parallel jobs (-1 for all cores)

6.3 Tuning with OOB Error

```python
import numpy as np
import matplotlib.pyplot as plt

```

```python
# Test different numbers of estimators
n_estimators_range = [10, 25, 50, 100, 200, 500]
oob_scores = []
test_scores = []

for n in n_estimators_range:
    clf = BaggingClassifier(
        estimator=DecisionTreeClassifier(max_depth=15),
        n_estimators=n,
        oob_score=True,
        random_state=42
    )
    clf.fit(X_train, y_train)
    oob_scores.append(clf.oob_score_)
    test_scores.append(clf.score(X_test, y_test))

# Plot results
plt.figure(figsize=(10, 6))
plt.plot(n_estimators_range, oob_scores, 'b-o', label='OOB Score')
plt.plot(n_estimators_range, test_scores, 'r-s', label='Test Score')
plt.xlabel('Number of Estimators')
plt.ylabel('Accuracy')
plt.title('Effect of Number of Trees')
plt.legend()
plt.grid(True)
plt.show()
```

6.4 Comparing with Single Decision Tree

```python
from sklearn.tree import DecisionTreeClassifier

# Single deep tree
single_tree = DecisionTreeClassifier(max_depth=20)
single_tree.fit(X_train, y_train)
print(f"Single Tree - Train: {single_tree.score(X_train, y_train):.4f}")
print(f"Single Tree - Test:  {single_tree.score(X_test, y_test):.4f}")

# Bagged trees
bagging = BaggingClassifier(
    estimator=DecisionTreeClassifier(max_depth=20),
    n_estimators=100,
    oob_score=True,
    random_state=42
)
bagging.fit(X_train, y_train)
print(f"Bagging - Train: {bagging.score(X_train, y_train):.4f}")
print(f"Bagging - OOB:   {bagging.oob_score_:.4f}")
print(f"Bagging - Test:  {bagging.score(X_test, y_test):.4f}")

```

```
# You'll typically see:
# - Single tree: High training, low test (overfit)
# - Bagging: Similar or lower training, much higher test
```

7 Limitations of Bagging

Bagging is powerful, but it has some important limitations:

7.1 Loss of Interpretability

Warning

One of the biggest advantages of decision trees is **interpretability**—you can explain exactly why a prediction was made by following the tree.
With bagging, we have 100+ trees, each making slightly different decisions. There's no single "path" to explain. The interpretability is lost.
We'll learn about techniques like **feature importance** in the Random Forest lecture to partially address this.

7.2 Still Need to Control Depth

While bagging reduces variance, you can't just set `max_depth=1000` and expect magic:

- **Too shallow**: Even averaging many underfit trees gives an underfit result
- **Too deep**: Computational cost increases, and correlation between trees increases

You still need to tune the depth, just less aggressively than with single trees.

7.3 The Correlation Problem (MAJOR)

This is the most important limitation and motivates the next topic (Random Forests).

Important: Trees Are Often Correlated!

The mathematical formula for variance reduction assumes trees are **independent**. But in practice, they're often highly **correlated**.
Why? If one feature (e.g., "Glucose" for diabetes prediction) is extremely predictive:

- Tree 1 will split on Glucose at the root
- Tree 2 will split on Glucose at the root
- Tree 3 will split on Glucose at the root
- ... and so on

All trees end up looking very similar! They make the same errors, so averaging doesn't help as much.

Example: Visualizing Tree Correlation

Look at bagged trees for diabetes prediction:

	Tree 1	**Tree 2**	**Tree 3**
Root Node	Glucose	Glucose	Glucose
Level 2	BMI	Blood Pressure	BMI
Level 3	Age	Age	Blood Pressure

All three trees start with Glucose! This correlation limits how much variance we can reduce.

This limitation leads us directly to **Random Forests**, which we'll cover in the next lecture. Random Forests add a clever twist: when building each tree, they only consider a **random subset of features** at each split. This "decorrelates" the trees and makes bagging much more effective.

8 Key Takeaways

Key Summary

What We Learned Today:

1. **The Problem:** Single decision trees struggle with bias-variance tradeoff
2. **The Solution:** Ensemble learning—combine multiple models
3. **Bagging = Bootstrap + Aggregating:**
   - Create diverse training sets via bootstrap sampling
   - Train a model on each (typically deep decision trees)
   - Aggregate predictions (vote or average)
4. **Why It Works:** Averaging independent predictions reduces variance
5. **Key Hyperparameters:**
   - Tree depth (controls complexity/bias-variance)
   - Number of estimators (reduces variance, never causes overfit)
6. **OOB Error:** Free validation using bootstrap "leftovers"
   - About 36.8% of data is OOB for each tree
   - Replaces need for validation set
   - More robust than cross-validation
7. **Limitations:**
   - Loss of interpretability
   - Tree correlation limits effectiveness
8. **Coming Up:** Random Forests solve the correlation problem!

Table 2: *Comparison: Single Tree vs. Bagging*

Aspect	Single Tree	Bagging
Bias	Can be low (deep tree)	Can be low
Variance	High	Low (averaging)
Interpretability	High	Low
Computation	Fast	Slower (many trees)
Overfitting Risk	High	Lower

9 Practice Questions

1. **Why does averaging reduce variance?** Explain the mathematical principle behind why combining multiple predictions gives a more stable result.
2. **Bootstrap Sampling:** If you have 1000 data points and create a bootstrap sample of 1000 points, approximately how many unique points will be in the sample? How many will be missing (OOB)?
3. **Hyperparameter Behavior:** Explain why increasing `n_estimators` reduces variance but doesn't cause overfitting, while increasing `max_depth` can cause overfitting.
4. **OOB vs. Cross-Validation:** What advantage does OOB error have over 5-fold cross-validation?
5. **The Correlation Problem:** Why might all trees in a bagging ensemble start with the same root node? How does this affect the variance reduction?
6. **Practical Question:** You train a bagging classifier with 100 trees. The training accuracy is 99%, but the OOB score is 85%. What does this tell you? What might you try to improve?
7. **Code Question:** Write sklearn code to:
   - Create a BaggingRegressor with 200 trees
   - Use trees of depth 10
   - Enable OOB scoring
   - Print both OOB score and test score