Lecture 20:

Model Explainability - SHAP and Deep Learning XAI

Ho-min Park

homin.park@ghent.ac.kr

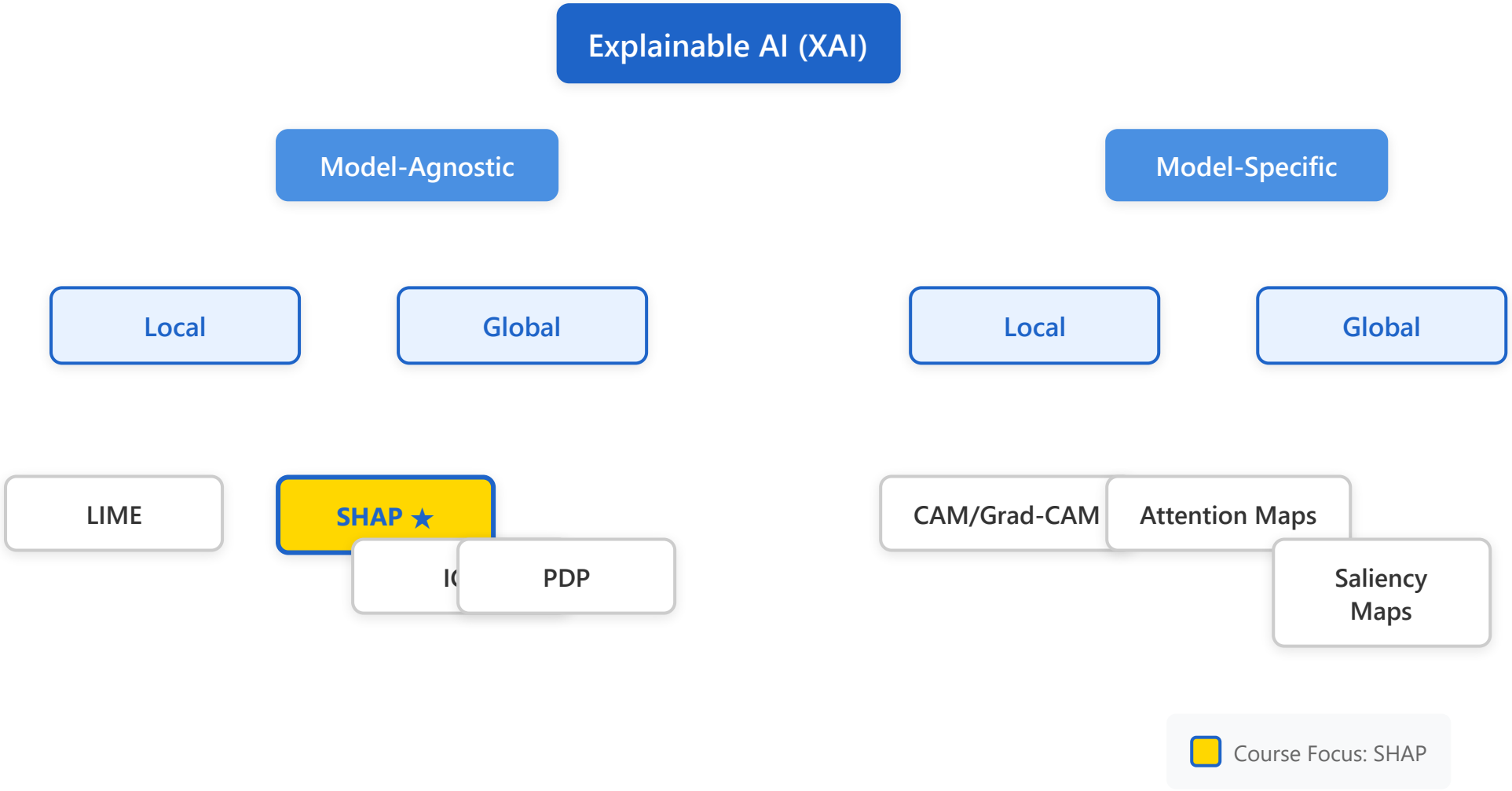
powersimmani@gmail.com

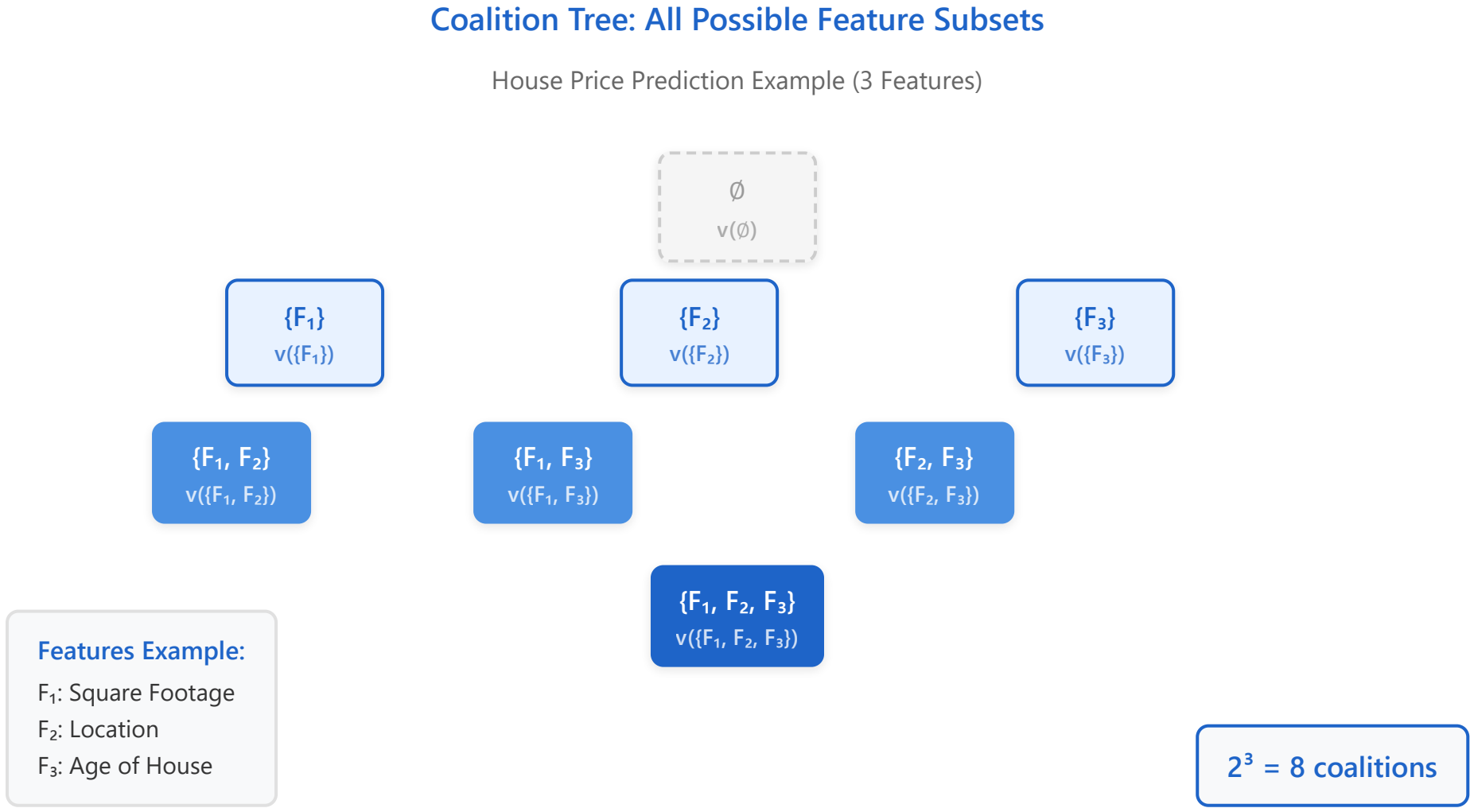
Lecture Contents

Part 1/4:

SHAP Theory and Fundamentals

1. Course Introduction - Game Theory-Based Explainability
2. Cooperative Game Theory Basics
3. Shapley Values Mathematical Definition
4. Core Ideas of SHAP
5. SHAP vs LIME Comparison
6. Interpreting SHAP Values
7. Mathematical Properties of SHAP
8. Hands-on: First SHAP Analysis


Explainable AI (XAI)
Model-Agnostic
Model-Specific
Local
Global
Local
Global
LIME
SHAP ★
I
PDP
CAM/Grad-CAM
Attention Maps
Saliency Maps
Course Focus: SHAP


Coalition Tree: All Possible Feature Subsets
House Price Prediction Example (3 Features)
∅
v(∅)
{F1}
v({F1})
{F2}
v({F2})
{F3}
v({F3})
{F1, F2}
v({F1, F2})
{F1, F3}
v({F1, F3})
{F2, F3}
v({F2, F3})
{F1, F2, F3}
v({F1, F2, F3})
Features Example:
F1: Square Footage
F2: Location
F3: Age of House
2^3 = 8 coalitions

Shapley Values: Mathematical Definition

$$\varphi_i = \sum_{S \subseteq N \setminus \{i\}} \left[|S|!(n-|S|-1)! \,/\, n! \right] \times \left[v(S \cup \{i\}) - v(S) \right]$$

Shapley value for feature i

φ_i

Shapley Value

Contribution of feature i to the prediction

$\sum_{S \subseteq N \setminus \{i\}}$

Sum Over Coalitions

All possible subsets S without feature i

$|S|!(n-|S|-1)! \,/\, n!$

Weight Factor

Probability of each ordering (combinatorial weight)

$v(S \cup \{i\}) - v(S)$

Marginal Contribution

Value added when i joins coalition S

Fairness Axioms (Uniqueness Theorem)

- ✓ Efficiency
- ✓ Symmetry
- ✓ Dummy
- ✓ Additivity

⚠ Complexity: $O(2^n)$ for n features

🔍 Practical Calculation Example: House Price Prediction

📊 Scenario

A house has a predicted price of **$4,000,000**. Three features contributed to this prediction:

x_1: Area ($85m^2$) | x_2: Number of Rooms (3) | x_3: Near Subway (5 min walk)

Goal: Calculate how much each feature contributed to the prediction using Shapley Values

1 Enumerate All Coalitions for Feature x_1 (Area)

Consider all possible subsets S that exclude feature x_1. With n=3, there are $2^{3-1} = 4$ coalitions.

Coalition S	|S|	v(S)	v(S∪{x_1})	Marginal v(S∪{x_1}) - v(S)
∅ (empty set)	0	$3,000,000	$3,500,000	+$500,000
{x_2} (rooms only)	1	$3,200,000	$3,800,000	+$600,000
{x_3} (subway only)	1	$3,400,000	$3,900,000	+$500,000
{x_2, x_3} (both)	2	$3,700,000	$4,000,000	+$300,000

2 Calculate Weight for Each Coalition

Weight formula: **|S|!(n-|S|-1)! / n!** where n=3

S = ∅:

Weight = 0! × (3-0-1)! / 3! = 1 × 2! / 6 = 2/6 = **1/3**

S = {x_2}:

Weight = 1! × (3-1-1)! / 3! = 1 × 1! / 6 = 1/6 = **1/6**

S = {x_3}:

Weight = 1! × (3-1-1)! / 3! = 1 × 1! / 6 = 1/6 = **1/6**

S = {x_2, x_3}:

Weight = 2! × (3-2-1)! / 3! = 2 × 0! / 6 = 2/6 = **1/3**

3 Calculate Shapley Value for Feature x_1

$\varphi_1 = \Sigma$ [Weight × Marginal Contribution]

$\varphi_1 = (1/3 \times 500{,}000) + (1/6 \times 600{,}000) + (1/6 \times 500{,}000) + (1/3 \times 300{,}000)$

$\varphi_1 = 166{,}667 + 100{,}000 + 83{,}333 + 100{,}000$

$\varphi_1 = \$450{,}000$

4 Shapley Values for Remaining Features (Same Method)

Repeating the same process for x_2 and x_3:

> **Shapley Value for x_2 (Number of Rooms):** $\varphi_2 = \$350{,}000$
>
> **Shapley Value for x_3 (Near Subway):** $\varphi_3 = \$200{,}000$

✨ Final Results: Shapley Values

Area (x_1)	Rooms (x_2)	Subway (x_3)
+$450K	**+$350K**	**+$200K**

Verification: 450K + 350K + 200K = 1,000K (difference between baseline $3M and prediction $4M) ✓

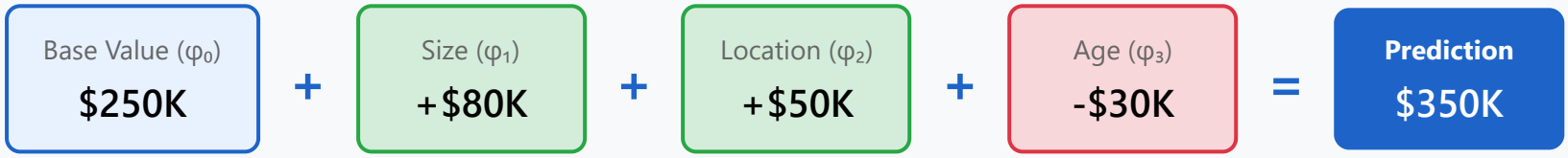
💡 Interpretation

- **Area ($450K)** made the largest contribution. This means that across all possible feature combinations, area added the most value on average.
- **Number of Rooms ($350K)** is the second most important feature.
- **Near Subway ($200K)** has the smallest contribution, but still makes a positive impact.
- The sum of all Shapley Values exactly equals the difference between prediction and baseline ($1M). This satisfies the **Efficiency axiom**.

- Each feature's contribution is fairly distributed by considering all possible combinations with other features.

SHAP: Additive Feature Attribution

$$f(x) = \varphi_0 + \Sigma\varphi_i$$



Characteristic	SHAP Values	Traditional Feature Importance
Model-Agnostic	✓ Works with any model	✗ Model-specific
Local Explanation	✓ Individual predictions	✗ Global only
Theoretical Foundation	✓ Game theory (Shapley)	✗ Heuristic-based
Feature Interactions	✓ Captures interactions	✗ Independent only
Consistency	✓ Fairness axioms satisfied	✗ No guarantees

SHAP vs LIME: Side-by-Side Comparison

LIME

🎯 **Approach:**
Local linear approximation

🔄 **Sampling:**
Perturbed samples around data point

⚡ **Speed:**
Fast computation

📊 **Consistency:**
No theoretical guarantees

🎲 **Results:**
May vary between runs

SHAP

🎯 **Approach:**
Game-theoretic framework

🔄 **Sampling:**
Coalitional sampling (exact solutions)

⚡ **Speed:**
More computationally intensive

📊 **Consistency:**
Fairness axioms satisfied

🎲 **Results:**
Globally consistent, deterministic

When to Choose?

Use LIME

→ Need quick explanations
→ Prototyping phase
→ Simple interpretability

Use SHAP

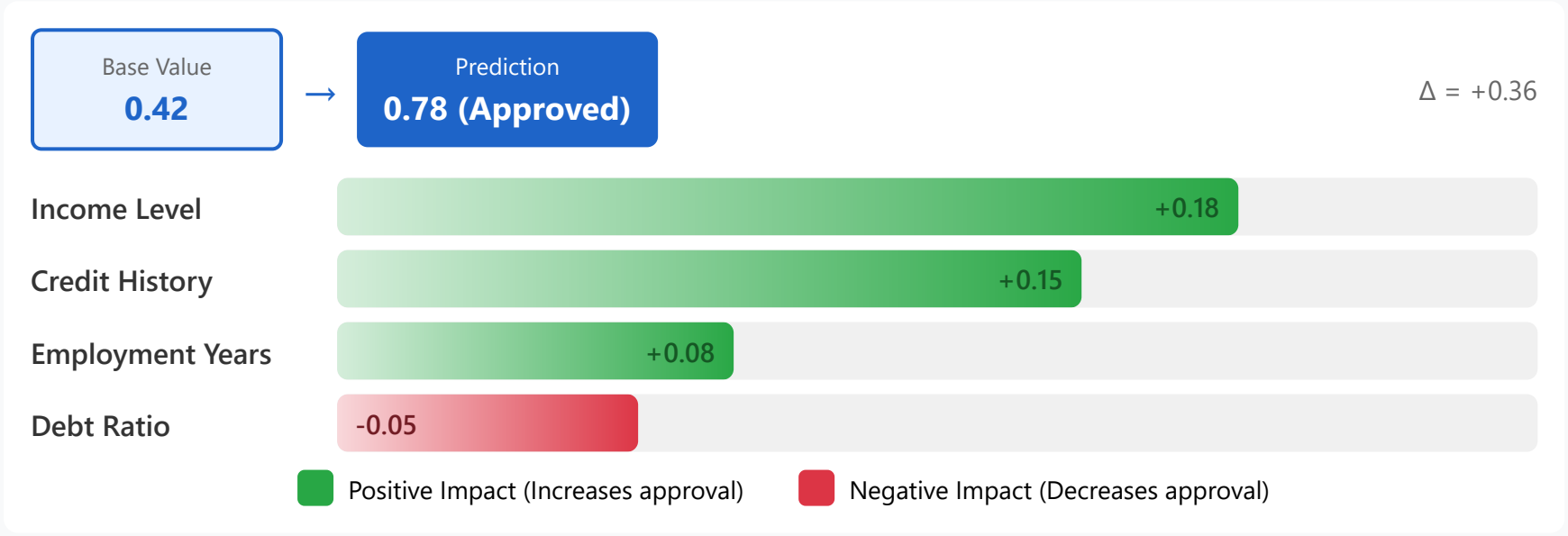
→ Critical decisions (healthcare, finance)
→ Need theoretical guarantees
→ Comparing feature importance

→ Resource constraints

→ Regulatory compliance

Interpreting SHAP Values: Credit Approval Decision

Example: Credit Approval for Applicant #247



Positive Values

Feature pushes prediction higher than base value

Negative Values

Feature pushes prediction lower than base value

Magnitude

Larger absolute value = greater importance for this prediction

Sum Property

Sum of all SHAP values = deviation from base value

Mathematical Properties of SHAP

1

Local Accuracy

$$f(x) = \varphi_0 + \Sigma\varphi_i(x)$$

Explanation matches model output exactly

2

Missingness

$$x_i = 0 \Rightarrow \varphi_i = 0$$

Absent features have zero attribution

3

Consistency

$$\Delta f_i \uparrow \Rightarrow \varphi_i \uparrow$$

Higher contribution = higher SHAP value

4

Efficiency

$$\Sigma\varphi_i = f(x) - E[f(x)]$$

Values sum to deviation from average

5

Symmetry

$$i \equiv j \Rightarrow \varphi_i = \varphi_j$$

Equivalent features get equal credit

6

Linearity

$$\varphi(f+g) = \varphi(f) + \varphi(g)$$

Additive for model ensembles

Visual Proof: Local Accuracy Property

Base Value	+	Feature 1	+	Feature 2	+	Feature 3	=	Model Output
$\varphi_0 = 100$		$\varphi_1 = +30$		$\varphi_2 = +20$		$\varphi_3 = -10$		$f(x) = 140$

$$100 + 30 + 20 - 10 = 140$$

✓ Local Accuracy Verified: $\varphi_0 + \Sigma\varphi_i = f(x)$

Hands-on: First SHAP Analysis Workflow

1 **Setup**
`pip install shap`

2 **Load Data**
`load_boston()`

3 **Train Model**
`RandomForest`

4 **Explain**
`TreeExplainer`

↓

💻 Complete Code Example

```
import shap
from sklearn.ensemble import
RandomForestRegressor
from sklearn.datasets import load_boston


# Load dataset
X, y = load_boston(return_X_y=True)


# Train model
model =
RandomForestRegressor(n_estimators=100)
model.fit(X, y)
```

📈 Key Insights

🎯 What You'll See:

- Waterfall plot showing feature contributions
- Base value (average prediction)
- Each feature's push/pull effect
- Final prediction value

💡 Interpretation Tips:

- Red bars = increase prediction
- Blue bars = decrease prediction
- Bar length = importance magnitude
- Sum of all = deviation from base

⚙️ Model Options:

- TreeExplainer: XGBoost, LightGBM, RF
- KernelExplainer: Any model (slower)

```python
# Initialize explainer
explainer = shap.TreeExplainer(model)

# Compute SHAP values
shap_values = explainer.shap_values(X)

# Visualize single prediction
shap.waterfall_plot(
    shap.Explanation(
        values=shap_values[0],
        base_values=explainer.expected_value
    )
)
```

- DeepExplainer: Neural networks

Part 2/4:

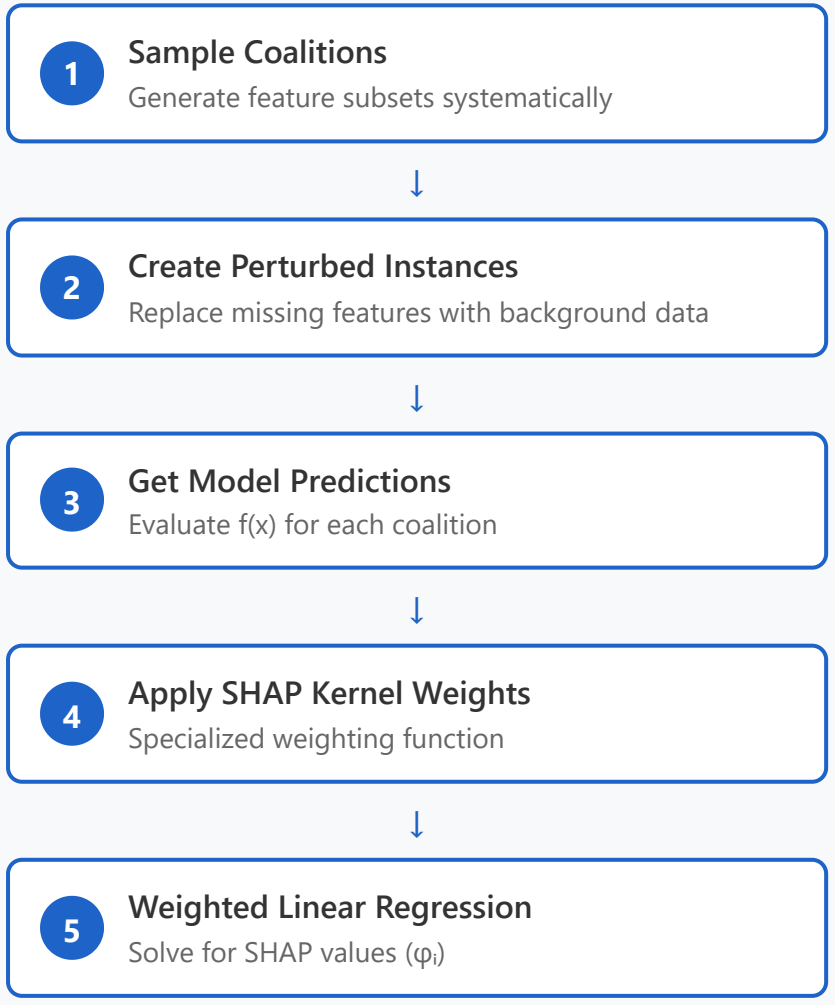
SHAP Implementation Methods

9. KernelSHAP
10. TreeSHAP
11. DeepSHAP (DeepLIFT + SHAP)
12. GradientSHAP
13. SHAP Approximation Techniques
14. SHAP Interaction Values
15. Hands-on: Comparing Different SHAP Explainers

KernelSHAP: Model-Agnostic Approximation

Sampling-based method with weighted linear regression

Algorithm Flowchart



Key Advantage

Model-Agnostic - Works with any black-box model

- Neural networks
- Ensemble methods
- Custom models

Trade-off

Computational Cost vs Accuracy

More samples → Better approximation

Fewer samples → Faster computation

Implementation

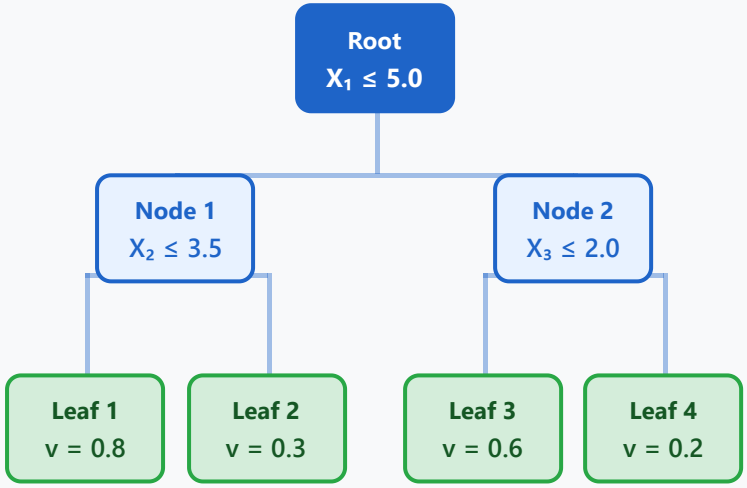
```
import shap

explainer = shap.KernelExplainer(
    model.predict,
    X_train
)
shap_values = explainer.shap_values(X_test)
```

TreeSHAP: Optimized for Tree-Based Models

Polynomial time algorithm with exact Shapley values

Tree Traversal Example



Complexity Analysis

$$O(TLD^2) \text{ vs } O(2^M)$$

T: trees, L: leaves, D: max depth, M: features

Speed Advantage

10-100x faster than KernelSHAP for tree models

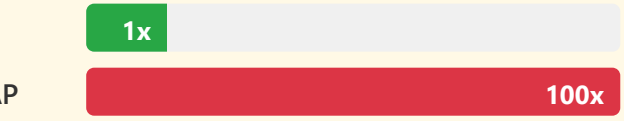
Exact Values

Computes exact Shapley values using tree structure

Model Support

Random Forest, XGBoost, LightGBM, CatBoost

Speed Comparison



TreeSHAP Computation Principles

1 Tree Structure Exploitation

TreeSHAP efficiently computes Shapley values by leveraging the hierarchical structure of tree-based models. It tracks decision paths at each node while reusing computations for feature combinations that share the same subtree.

$$\varphi_i = \Sigma_{S \subseteq M \setminus \{i\}} \, |S|!(M-|S|-1)! \, / \, M! \times [f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S)]$$

Original Shapley value definition

2 Recursive Computation

The algorithm recursively traverses the tree from root to leaves, tracking two pieces of information at each node:

🔑 Tracked Information

- **Coverage**: The proportion of possible feature combinations passing through the current node
- **Contribution**: The impact each feature has on the prediction value

$$\texttt{RECURSE}(j, m, p_z, p_o, v)$$

j: current node, m: unique path, p_z: zero probability, p_o: one probability, v: node value

3 Path Integration

When the same feature is used across multiple paths, the final Shapley value is calculated by taking a weighted average of each path's contribution. This process obtains exact values without explicitly enumerating all possible paths in the tree.

$$\varphi_i = \Sigma_{\text{paths}} \, w_{\text{path}} \times \Delta v_{i,\text{path}}$$

w_{path}: path weight, $\Delta v_{i,path}$: contribution of feature i along the path

4 Ensemble Integration

For ensemble models like Random Forest or Gradient Boosting, the Shapley values are computed independently for each tree and then averaged to generate the final explanation.

$$\varphi_i^{ensemble} = (1/T) \times \Sigma_{t=1}^{T} \varphi_i^{(t)}$$

T: number of trees, $\varphi_i^{(t)}$: Shapley value of feature i for the t-th tree

✓ TreeSHAP vs KernelSHAP

TreeSHAP

Computation: Direct use of tree structure

Complexity: $O(TLD^2)$

Accuracy: Exact Shapley values

Speed: Very fast (polynomial time)

Application: Tree-based models only

KernelSHAP

Computation: Sampling-based approximation

Complexity: $O(2^M)$

Accuracy: Approximate values

Speed: Slow (exponential time)

Application: Any model

💡 Key Insights

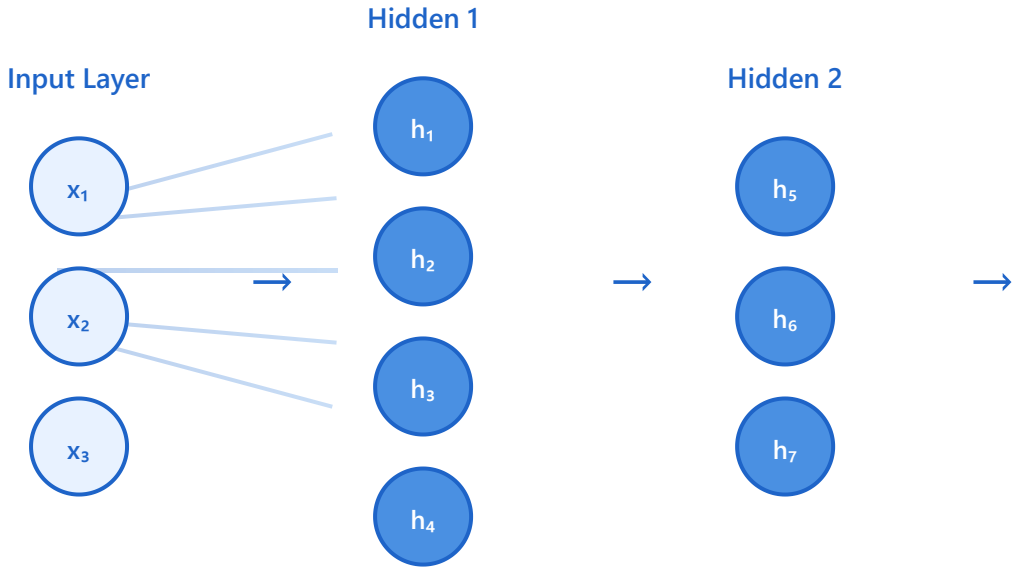
- TreeSHAP reduces exponential complexity to polynomial complexity by efficiently computing conditional expectations in trees

- It leverages dynamic programming principles to eliminate redundant computations and maximize memory efficiency
- Unlike model-agnostic KernelSHAP, it enables accurate and fast computation by exploiting the model's internal structure

DeepSHAP: DeepLIFT + SHAP for Neural Networks

Backpropagation of reference activation differences

Neural Network Computation Flow



DeepSHAP Computation Steps

→ Compute reference activations (average)

→ Forward pass: input → output

→ Backward pass: propagate differences

Key Features

Combines DeepLIFT with Shapley value sampling

- Layer-wise decomposition
- Handles nonlinear activations
- Reference-based differences

tivations

Supports various activation functions

- ReLU
- Sigmoid
- Tanh

Reference Value

Typically uses average of training data as baseline

Implementation

→ Assign contributions to each input

```python
import shap

# Create explainer
explainer = shap.DeepExplainer(
    model,
    X_train[:100]
)

# Compute SHAP values
shap_values = explainer.shap_values(X_test)
```

DeepSHAP Computation Principle

📐 Core Formulas

$$\varphi_i = \Delta\text{output} \times (\Delta x_i \,/\, \Sigma\Delta x_j)$$

φ_i: Feature i의 SHAP value
Δoutput: Output difference from reference
Δx_i: Input i difference from reference
$\Sigma\Delta x_j$: Sum of all input differences

Layer-wise propagation:

$$C_{ij} = m_{ij} \times \Delta x_j \Delta h_i$$

C_{ij}: Contribution from neuron j to i
m_{ij}: Multiplier (weight effect)
Δx_j: Input activation difference
Δh_i: Hidden activation difference

🔢 Numerical Example

Step 1: Set Reference

Reference (baseline): $x_0 = [0, 0, 0]$
Input (actual): $x = [1, 2, 1]$
Differences: $\Delta x = [1, 2, 1]$

Step 2: Forward Pass

Hidden layer: $h = \text{ReLU}(W \cdot x + b)$
Reference: $h_0 = \text{ReLU}(W \cdot x_0 + b) = [0, 0]$
Actual: $h = [2, 3]$
Differences: $\Delta h = [2, 3]$

Step 3: Output Difference

Reference output: $y_0 = 0.3$
Actual output: $y = 0.8$
Output difference: $\Delta y = 0.5$

Step 4: Backward Propagation

Calculate each input's contribution via backpropagation:

$\varphi_1 = 0.5 \times (1/4) = 0.125$

$\varphi_2 = 0.5 \times (2/4) = 0.250$

$\varphi_3 = 0.5 \times (1/4) = 0.125$

✅ Final Result Interpretation

- Feature x_2 has the largest impact ($\varphi_2 = 0.250$)
- Sum of all SHAP values = 0.5 = Δy ✓
- Features x_1, x_3 have equal contributions (0.125)

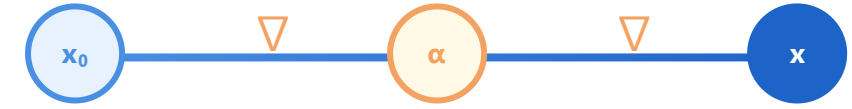
GradientSHAP: Gradient-Based Approximation

Combines Integrated Gradients with Shapley sampling

$$\varphi_i = E[\nabla f(x) \times (x_i - x_{baseline})]$$

Expectation over multiple baselines improves stability

Integration Path from Baseline to Input



Baseline (Reference)

Input (Target)

Computation Steps

1. Sample multiple random baselines
2. Create interpolated points along path
3. Compute gradients at each point
4. Multiply by input difference
5. Average over all baselines

Efficiency

Fast computation for differentiable models

- Uses automatic differentiation
- GPU acceleration supported
- Scales well with features

Multiple Baselines

Random sampling improves robustness

- Reduces variance
- Better approximation
- More stable estimates

Non-linear Interactions

Captures complex feature relationships

- Gradient-based attribution
- Handles neural networks
- Integrated Gradients basis

📊 Calculation Principle & Examples

Example 1: Image Classification Model

🎯 Problem Setup

Cat image classification (features: ears, whiskers, eyes)

```
x = [0.8, 0.6, 0.9]
x₀ = [0, 0, 0] (black image)
```

📐 Step 1: Generate Path

Intermediate point at α = 0.5:

```
x' = x₀ + 0.5(x - x₀)
= [0.4, 0.3, 0.45]
```

🔍 Step 2: Compute Gradients

```
▽f(x') = [0.7, 0.4, 0.8]
(importance of each feature)
```

✖️ Step 3: Multiply with Difference

$$\varphi_i = \nabla f(x') \times (x_i - x0_i)$$

$$\varphi_1 = 0.7 \times 0.8 = 0.56$$

Example 2: Multiple Baselines

🎲 Problem Setup

Credit score prediction (features: income, debt)

```
x = [80k, 20k]
Sample 3 baselines
```

📊 Calculate for Each Baseline

```
Baseline 1: x₀¹ = [30k, 10k]
→ φ¹ = [0.45, -0.15]

Baseline 2: x₀² = [50k, 15k]
→ φ² = [0.38, -0.12]

Baseline 3: x₀³ = [40k, 5k]
→ φ³ = [0.52, -0.18]
```

📈 Step 4: Average Results

$$\varphi_i = E[\varphi^n] = (\varphi^1 + \varphi^2 + \varphi^3) / 3$$

φ_2 = 0.4 × 0.6 = 0.24

φ_3 = 0.8 × 0.9 = 0.72

Final Attribution

Eyes(0.72) > Ears(0.56) > Whiskers(0.24)

```
Income: (0.45+0.38+0.52)/3 = 0.45
Debt: (-0.15-0.12-0.18)/3 = -0.15
```

Stable Attribution

Income: +0.45 | Debt: -0.15

Interpreting Results

- ✅ **Positive values:** Feature contributes to increasing the prediction (e.g., higher income → higher credit score)
- ❌ **Negative values:** Feature contributes to decreasing the prediction (e.g., higher debt → lower credit score)
- 📊 **Magnitude:** Larger absolute value indicates stronger influence (e.g., |0.72| > |0.56| → eyes more important than ears)
- 🎯 **Multiple baselines:** Using multiple reference points reduces variance and provides more stable results

Attention Mechanisms as Explanations: Transformer Visualization

Self-Attention Matrix (BERT Example)

High (>0.35) Med-High Medium Low Very Low

Token Attention Example: "movie" attending to:

Self-Attention

Shows token-to-token importance in context

→ Query-Key-Value mechanism

→ Context-aware weights

→ Bidirectional relationships

Multi-Head Attention

Different heads capture different aspects

→ Parallel attention layers

→ Diverse relationship types

→ Aggregate for full picture

Attention Methods

→ **Rollout:** Aggregate across layers

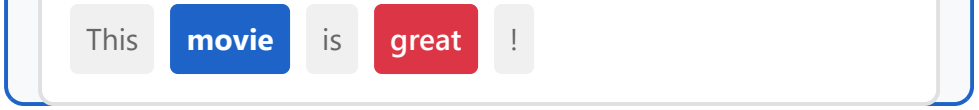
→ **Flow:** Trace information path

→ **Gradient:** Combine with gradients

⚠ Important Limitation

Attention ≠ Explanation

Recent research shows attention weights don't always reflect true



model reasoning. Combine with gradient-based methods for better insights.

SHAP Interaction Values: Capturing Feature Interactions

φ_{ij} measures joint effect between features i and j

Interaction Matrix: Credit Scoring Example

	Age	Income	Debt	Credit
Age	0.15	0.08	0.02	0.04
Income	0.08	0.22	0.05	0.03
Debt	0.02	0.05	-0.18	0.02
Credit	0.04	0.03	0.02	0.12

Main Effect Strong (>0.06) Moderate (0.04-0.06) Weak (<0.04)

$$\varphi_{total} = \varphi_i + \Sigma_j \varphi_{ij}$$

Total effect = Main effect + Interactions

Age × Income Interaction

Strong interaction (0.08) indicates that age and income jointly affect credit approval more than their individual effects suggest.

Scenario:

Young + High Income → Higher approval than expected
Old + Low Income → Lower approval than expected

Matrix Properties

- Symmetric: $\varphi_{ij} = \varphi_{ji}$
- Diagonal: main effects (φ_{ii})
- Off-diagonal: interactions
- TreeSHAP: exact computation

Interpretation Guide

- Positive: synergistic effect
- Negative: antagonistic effect

- Large magnitude: strong coupling
- Near zero: independent features

Hands-on: Comparing Different SHAP Explainers

1 Load Dataset → 2 Train Models → 3 Apply Explainers → 4 Compare Results

Explainer	Speed (Relative)	Accuracy	Best Use Case
TreeSHAP Tree models	Fast (1x)	Exact	Random Forest, XGBoost, LightGBM
DeepSHAP Neural nets	Moderate (5x)	High	Deep learning, CNN, RNN models
GradientSHAP Differentiable	Moderate (4x)	High	Neural networks with gradients
KernelSHAP Any model	Slow (20x)	Approx	Model-agnostic, black-box models

Implementation Examples with Timing

TreeSHAP (XGBoost)

```
# Fast and exact
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X)
# Time: ~0.5s for 1000 samples
```

DeepSHAP (Neural Network)

```
# Optimized for deep learning
explainer = shap.DeepExplainer(model, X_bg)
shap_values = explainer.shap_values(X)
# Time: ~2.5s for 1000 samples
```

GradientSHAP

```
# Gradient-based approximation
explainer = shap.GradientExplainer(model, X_bg)
shap_values = explainer.shap_values(X)
# Time: ~2s for 1000 samples
```

KernelSHAP (Any Model)

```
# Most flexible, slowest
explainer = shap.KernelExplainer(model.predict, X_bg)
shap_values = explainer.shap_values(X)
# Time: ~10s for 1000 samples
```

Part 3/4:

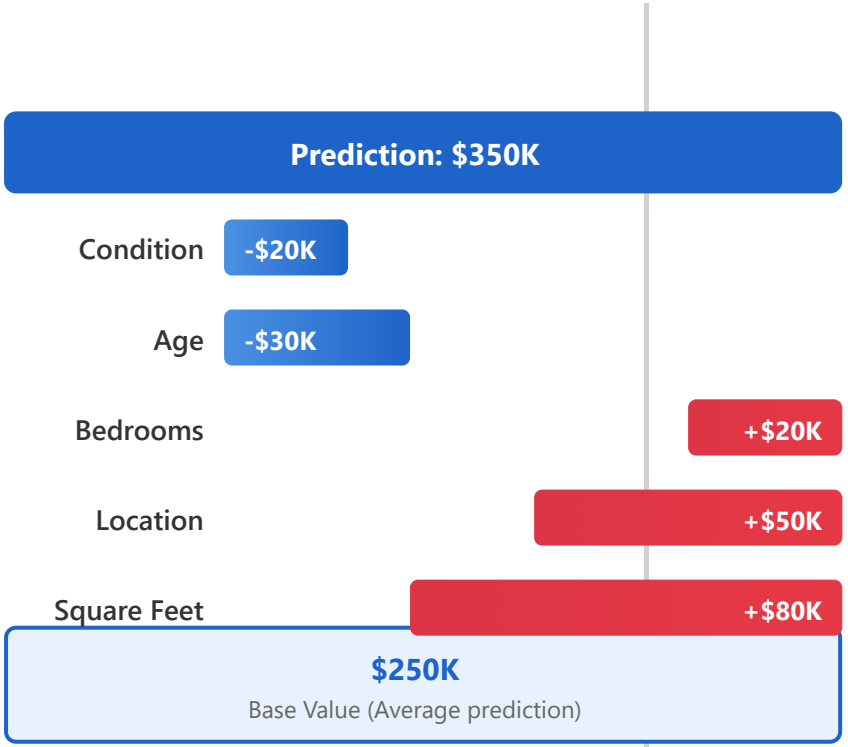
SHAP Visualization and Analysis

16. Waterfall Plot
17. Force Plot and Decision Plot
18. Summary Plot and Dependence Plot
19. SHAP for Time Series
20. SHAP for Text and Images

Waterfall Plot: Feature Contribution Breakdown

Visualizing how each feature contributes to a single prediction

Example: House Price Prediction ($350K)



How to Read

→ Start at base value (bottom)

→ Each bar adds/subtracts

→ Red = pushes higher

→ Blue = pushes lower

→ Final value at top

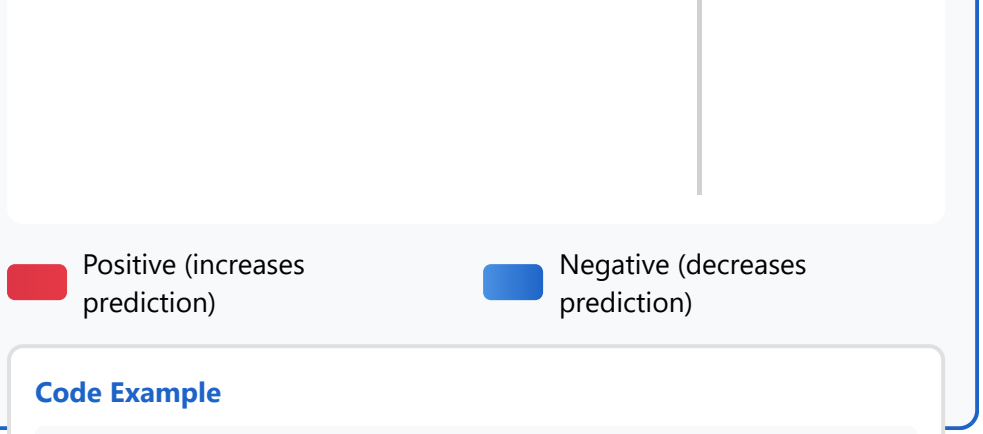
Key Insights

Waterfall plots show the additive nature of SHAP values

→ Base + contributions = prediction

→ Order: by impact magnitude

→ Top 10-15 features shown



Positive (increases prediction)

Negative (decreases prediction)

Code Example

```
import shap
shap.plots.waterfall(shap_values[0])
```

Use Cases

→ Explain individual predictions

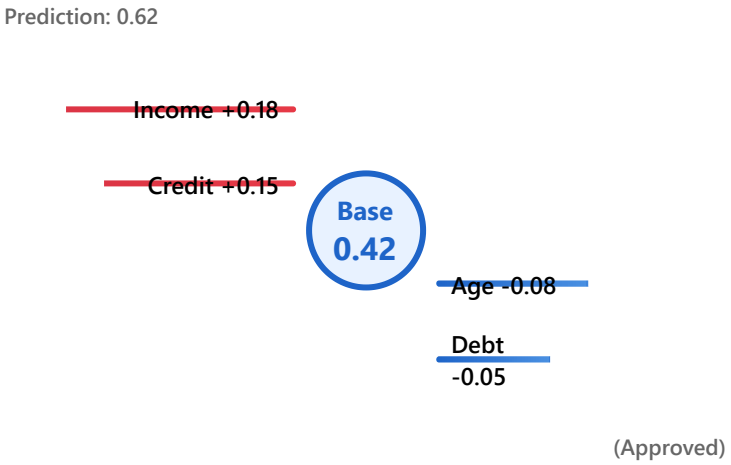
→ Identify key drivers

→ Debug model decisions

→ Communicate with stakeholders

Force Plot and Decision Plot: Interactive Visualizations

Force Plot



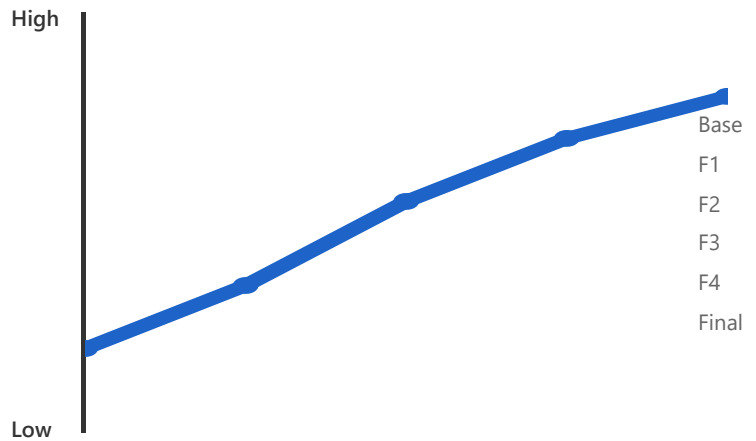
Characteristics

- Horizontal push/pull layout
- Interactive HTML output
- Shows force direction
- Color gradient by value

Code Example

```
shap.plots.force(
  base_value, shap_values[i]
)
```

Decision Plot



Characteristics

- Cumulative SHAP values
- Compare multiple predictions
- Shows decision boundaries
- Identifies critical thresholds

Code Example

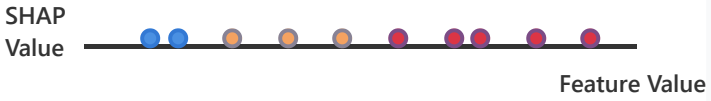
```
shap.plots.decision(
  base_value, shap_values
)
```

Summary Plot and Dependence Plot: Global Analysis

Summary Plot (Beeswarm)

Code Example

Dependence Plot



Key Insights

- Non-linear patterns
- Feature interactions
- Optimal ranges
- Threshold effects

Use Cases

- Understand feature behavior
- Identify sweet spots
- Detect interactions
- Validate model logic

Code Example

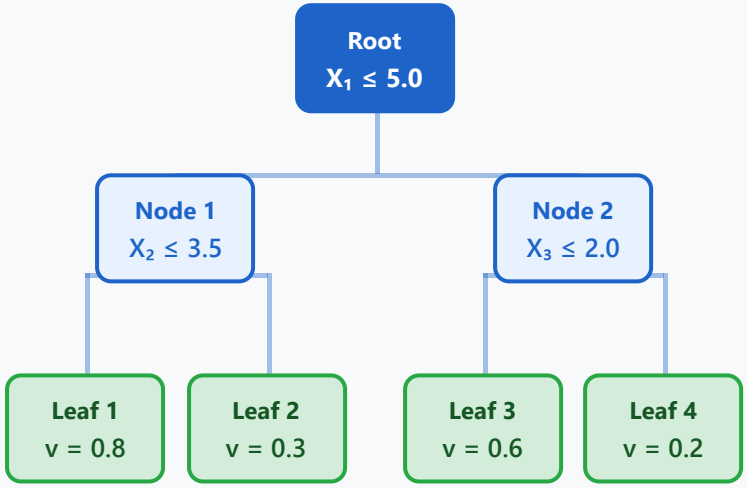
```
shap.summary_plot(shap_values, X)
```

```
shap.dependence_plot(
  "Income", shap_values, X
)
```

TreeSHAP: Optimized for Tree-Based Models

Polynomial time algorithm with exact Shapley values

Tree Traversal Example



Complexity Analysis

$$O(TLD^2) \text{ vs } O(2^M)$$

T: trees, L: leaves, D: max depth, M: features

Speed Advantage

10-100x faster than KernelSHAP for tree models

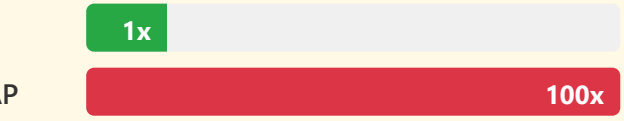
Exact Values

Computes exact Shapley values using tree structure

Model Support

Random Forest, XGBoost, LightGBM, CatBoost

Speed Comparison



TreeSHAP Computation Principles

1 Tree Structure Exploitation

TreeSHAP efficiently computes Shapley values by leveraging the hierarchical structure of tree-based models. It tracks decision paths at each node while reusing computations for feature combinations that share the same subtree.

$$\varphi_i = \Sigma_{S \subseteq M \setminus \{i\}} \, |S|!(M-|S|-1)! \, / \, M! \times [f_{S \cup \{i\}}(x_{S \cup \{i\}}) - f_S(x_S)]$$

Original Shapley value definition

2 Recursive Computation

The algorithm recursively traverses the tree from root to leaves, tracking two pieces of information at each node:

🔑 Tracked Information

- **Coverage**: The proportion of possible feature combinations passing through the current node
- **Contribution**: The impact each feature has on the prediction value

$$\texttt{RECURSE}(j, m, p_z, p_o, v)$$

j: current node, m: unique path, p_z: zero probability, p_o: one probability, v: node value

3 Path Integration

When the same feature is used across multiple paths, the final Shapley value is calculated by taking a weighted average of each path's contribution. This process obtains exact values without explicitly enumerating all possible paths in the tree.

$$\varphi_i = \Sigma_{\text{paths}} \, w_{\text{path}} \times \Delta v_{i,\text{path}}$$

w_{path}: path weight, $\Delta v_{i,path}$: contribution of feature i along the path

4 Ensemble Integration

For ensemble models like Random Forest or Gradient Boosting, the Shapley values are computed independently for each tree and then averaged to generate the final explanation.

$$\varphi_i^{ensemble} = (1/T) \times \Sigma_{t=1}^{T} \varphi_i^{(t)}$$

T: number of trees, $\varphi_i^{(t)}$: Shapley value of feature i for the t-th tree

✓ TreeSHAP vs KernelSHAP

TreeSHAP

Computation: Direct use of tree structure

Complexity: $O(TLD^2)$

Accuracy: Exact Shapley values

Speed: Very fast (polynomial time)

Application: Tree-based models only

KernelSHAP

Computation: Sampling-based approximation

Complexity: $O(2^M)$

Accuracy: Approximate values

Speed: Slow (exponential time)

Application: Any model

💡 Key Insights

- TreeSHAP reduces exponential complexity to polynomial complexity by efficiently computing conditional expectations in trees

- It leverages dynamic programming principles to eliminate redundant computations and maximize memory efficiency
- Unlike model-agnostic KernelSHAP, it enables accurate and fast computation by exploiting the model's internal structure

SHAP for Text and Images: High-Dimensional Explanations

Text (NLP Model)

Sentiment Analysis: Movie Review

This **brilliant** film captivates with stunning visuals but disappoints with **weak** plot development

Model Prediction

Mixed (0.58 Positive)

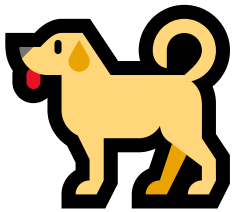
Strong Positive | Weak Positive | Neutral | Weak Negative | Strong Negative

Text SHAP Features

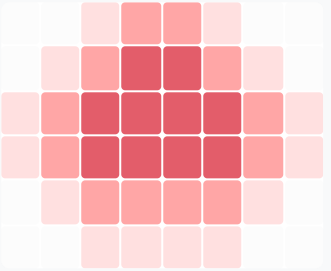
- Token-level attributions
- Color-coded importance
- Transformer + SHAP
- Context-aware explanations

Image (CNN Model)

Original Image



SHAP Heatmap



Model Prediction

Dog (0.95 confidence)

High Importance | Medium | Low | Minimal

Image SHAP Features

- Pixel/superpixel attributions
- Partition masking
- Heatmap overlay
- Region importance

Interactive Practice

Try WebSHAP →

Part 4/4:

Advanced Deep Learning XAI Techniques

21. Attention Mechanisms as Explanations
22. Gradient-based Methods
23. CAM-family Methods
24. Concept-based Explanations
25. Future of XAI and Challenges

Attention Mechanisms as Explanations: Transformer Visualization

Self-Attention Matrix (BERT Example)

	This	movie	is	great	!	[SEP]
This	**0.42**	0.18	0.08	0.15	0.05	0.02
movie	0.20	**0.35**	0.10	0.22	0.08	0.05
is	0.08	0.15	0.18	**0.45**	0.12	0.02
great	0.10	0.25	0.18	**0.38**	0.07	0.02
!	0.05	0.08	0.05	0.28	**0.48**	0.06
[SEP]	0.12	0.10	0.08	0.15	0.10	0.25

Self-Attention

Shows token-to-token importance in context

→ Query-Key-Value mechanism

→ Context-aware weights

→ Bidirectional relationships

Multi-Head Attention

Different heads capture different aspects

→ Parallel attention layers

→ Diverse relationship types

→ Aggregate for full picture

Attention Methods

→ **Rollout:** Aggregate across layers

→ **Flow:** Trace information path

High (>0.35) Med-High Medium Low Very Low

Token Attention Example: "movie" attending to:

This **movie** is great !

→ **Gradient:** Combine with gradients

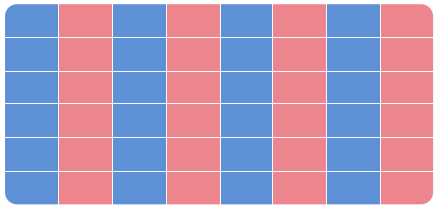
⚠ Important Limitation

Attention ≠ Explanation

Recent research shows attention weights don't always reflect true model reasoning. Combine with gradient-based methods for better insights.

Gradient-based Methods: Computing Feature Sensitivity

Vanilla Gradients



$$\partial y / \partial x$$

Basic sensitivity, noisy

Integrated Gradients

$$\int_0^1 \nabla f(x' + \alpha(x - x'))\, d\alpha$$

Path integral, smooth

SmoothGrad

$$E[\nabla f(x + N(0, \sigma^2))]$$

Averaged over noise

Grad-CAM

$$\text{ReLU}(\Sigma\ \alpha_k\ A_k)$$

Class activation map

Method Comparison

Vanilla	Fast, noisy ✗
Integrated	Smooth, slow ✓
SmoothGrad	Less noise ✓
Grad-CAM	CNN specific ✓

Strengths

- ✓ Model-agnostic
- ✓ Fast computation
- ✓ Differentiable

Limitations

- ✗ Saturation issue
- ✗ Noise in vanilla
- ✗ Requires gradients

PyTorch Implementation

Vanilla Gradients

```
x.requires_grad_()
output = model(x)
output.backward()
gradients = x.grad
```

Integrated Gradients

```
# Path from baseline to input
alphas = torch.linspace(0, 1, steps=50)
for alpha in alphas:
  x_step = baseline + alpha * (x - baseline)
  # Compute gradients at each step
```

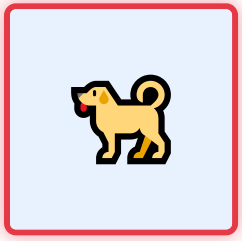
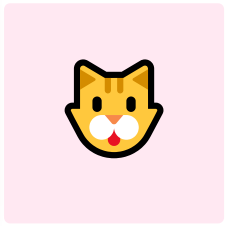
SmoothGrad

```
# Average over noisy samples
for _ in range(n_samples):
```

```
  noise = torch.randn_like(x) * sigma
  grads += compute_grad(x + noise)
smooth_grad = grads / n_samples
```

🔍 Interactive CAM Visualization Demo

📸 Select Image


🪄 Method Selection

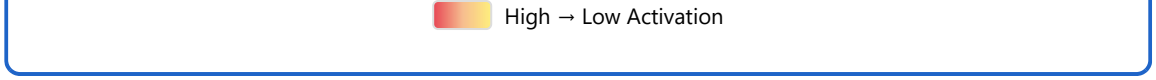
- **Grad-CAM**
- Integrated Gradients
- SmoothGrad
- Vanilla Gradients

🎨 Heatmap Settings

Opacity 60%

Focus Intensity 70%

CAM Visualization on Image


High → Low Activation

CAM-family Methods: Evolution of Class Activation Mapping

Evolution: CAM → Grad-CAM → Grad-CAM++ → Score-CAM → Layer-CAM

CAM
2016
Global average pooling required

Grad-CAM
2017
Any CNN architecture

Grad-CAM++
2018
Better localization

Score-CAM
2020
Gradient-free

Layer-CAM
2021
Layer-wise mapping

Key Methods

CAM

Global average pooling + linear layer weights

$$L_{CAM} = \Sigma\ w_k \cdot A_k$$

Grad-CAM

Gradient-weighted class activation

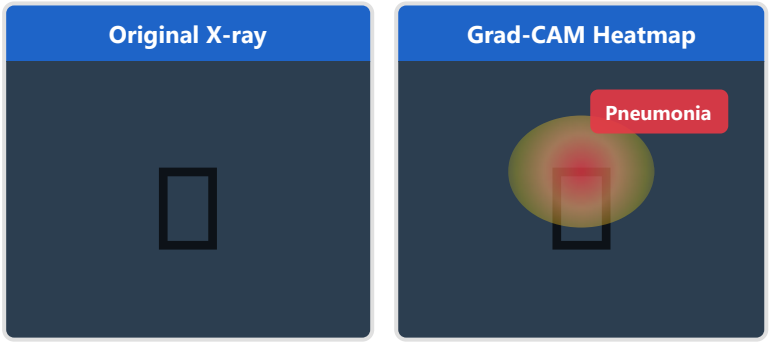
$$L = \mathrm{ReLU}(\Sigma\ \alpha_k \cdot A_k),\ \alpha_k = 1/Z\ \Sigma_i\ \Sigma_j\ \partial y^c/\partial A_k^{ij}$$

Score-CAM

Ablation-based, no gradients needed

$$L = \mathrm{ReLU}(\Sigma\ w_k \cdot A_k),\ w_k \text{ from forward pass}$$

Application: Medical X-ray



Key Applications

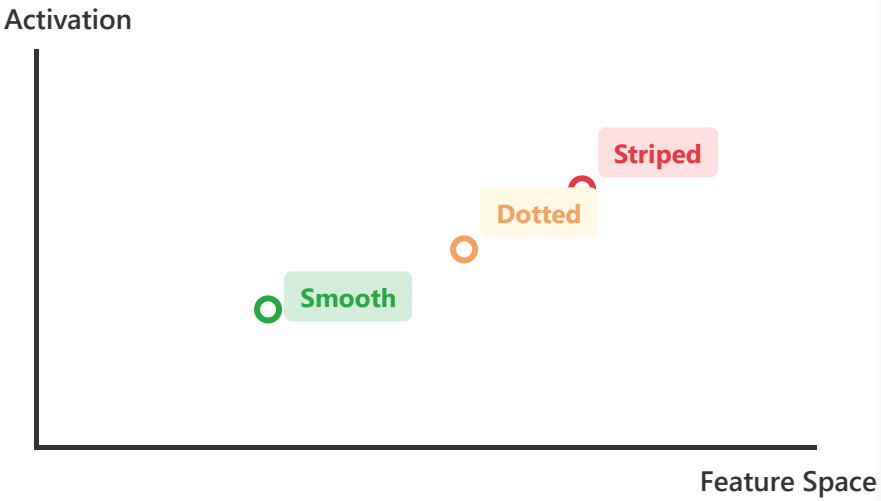
- **Medical Imaging:** Pathology localization
- **Object Detection:** Region proposals
- **Segmentation:** Boundary refinement
- **Quality Control:** Defect detection

Best for:	CNNs, visual tasks
Advantage:	**Fast** computation
Limitation:	CNN-specific

Concept-based Explanations

Human-understandable concepts vs individual features

Concept Space Visualization



Example: Texture Classification

Concept: "Striped pattern"
Model Sensitivity: How much does prediction change when moving in the direction of the striped concept?

$$S_C = \nabla_C f(x) \cdot v_C$$

Directional derivative along concept vector

Theoretical Foundation

TCAV Score:

$TCAV_{C,k,l}(x) = 1$ if $S_C^{k,l}(x) > 0$, else 0

TCAV

Testing with Concept Activation Vectors

Measures model sensitivity to user-defined concepts

- Define concept examples
- Train linear classifier
- Compute directional derivatives

ACE

Automated Concept-based Explanation

Automatically discovers important concepts without human input

- Segment activation space
- Cluster similar patterns
- Identify meaningful concepts

Concept Bottleneck

Future of XAI and Challenges: Towards Robust Explainability

Key Challenges

Standardization

Developing evaluation metrics for explanations quality and consistency

Faithfulness vs Plausibility

Do explanations truly reflect model reasoning or just appear convincing?

Counterfactual Explanations

"What if" scenarios: minimal changes needed for different outcomes

Causal Inference Integration

Combining XAI methods with causal reasoning frameworks

Explanation Stability

Ensuring robustness to small input perturbations and noise

Regulatory Requirements

- **EU AI Act:** High-risk AI transparency
- **FDA Guidelines:** Medical AI explainability

XAI Research Roadmap

2016-2018

Foundation Era

LIME, SHAP, attention mechanisms, CAM/Grad-CAM

2019-2021

Refinement Period

Integrated Gradients, improved CAM variants, faithfulness studies

2022-2024 Current

Integration Phase

Concept-based explanations, causal XAI, regulatory compliance

2025-2027 Emerging

Multimodal XAI

Cross-modal explanations, unified frameworks, interactive systems

2028+ Future

Autonomous XAI

⚖️ **GDPR:** Right to explanation

Self-explaining models, automatic explanation generation, AGI integration

Thank you

Ho-min Park

homin.park@ghent.ac.kr

powersimmani@gmail.com