# 4.LOCAL MODEL-AGNOSTIC METHODS: LIME & SHAP

# Working of LIME

The Intuition:

- Ignore the original training data and treat the model purely as a black box.
- Create variations of the instance you want to explain and probe the model with these samples.
- Observe how the model's predictions change in response to these variations.
- Fit an interpretable model to the sampled data, weighting samples more heavily when they are closer to the instance of interest.

Mathematical Formulation:

$$\text{explanation}(\mathbf{x}) = \arg\min_{g \in G} L(\hat{f}, g, \pi_{\mathbf{x}}) + \Omega(g)$$

Components:

- x: Instance to explain
- g: Explanation model (e.g., linear regression, decision tree)
- G: Family of possible explanations
- L: Loss measuring closeness to original model f̂
- π_x: Proximity measure (defines neighborhood around x)
- Ω(g): Model complexity penalty (prefer fewer features)

# Algorithm Steps

**Step 1: Select Instance**

Choose instance x for which you want explanation

**Step 2: Generate Perturbations**

- Create variations of x (method depends on data type)
- Get black box predictions for all perturbed samples

**Step 3: Weight by Proximity**

- Calculate proximity of each perturbed sample to x
- Closer samples get higher weights
- Uses exponential smoothing kernel: $\pi_x$

**Step 4: Train Interpretable Model**

- Fit weighted model on perturbed dataset
- Common choice: Lasso (for feature selection)
- Select K features: high $\lambda \rightarrow$ decrease $\lambda$ until K features remain

**Step 5: Extract Explanation**

- Interpret the local model (coefficients, feature importance, rules)
- This explains the original black box prediction for x

# Creating Variations by Data Type

**Tabular Data:**

•Perturb each feature individually
•Draw from normal distribution with feature's mean and standard deviation
•Issue: Samples from training data center (not around x)
•Issue: Ignores feature correlations → unrealistic data points

**Text Data:**

•Start with original text
•Randomly remove words
•Representation: Binary features (1 = word included, 0 = removed)
•Each variation = subset of original words

**Image Data:**

•Segment image into "superpixels" (connected similar-colored pixels)
•Turn superpixels on or off
•Off = replace pixels with user-defined color (e.g., gray)
•Specify probability for turning off each superpixel
•Why not individual pixels? Too many pixels contribute to classes

**Strengths:**

- Works across tabular, text, AND images (rare)
- Model replacement flexibility (swap models, keep explanation method)
- Short, selective explanations (human-friendly)
- Easy to use: Python (lime), R (iml, DALEX)
- Can use interpretable features different from training features
- Fidelity measure shows reliability of approximation

**Critical Limitations:**

- Neighborhood choice unsolved (biggest problem—no principled method)
- Instability: Repeating sampling → different explanations
- Unrealistic sampling: Ignores feature correlations
- Can be manipulated: Hide biases (Slack et al. 2020)
- Complexity (K) must be defined in advance

# Shapley Values - Game Theory Foundation

**The Game Analogy:**

- **Game:** The prediction task for a single instance is treated as a cooperative game.
- **Players:** The feature values act as players that work together to produce the final prediction.
- **Gain:** The gain is defined as the model's actual prediction for the instance minus the model's average prediction.
- **Goal:** Determine how to fairly distribute this gain among the feature values.

**Shapley Value Definition:**

- Average marginal contribution of feature across all possible coalitions

Mathematical Formula:

$$\phi_j(val) = \sum_{S \subseteq \{1,\ldots,p\} \setminus \{j\}} \frac{|S|!\,(p - |S| - 1)!}{p!} (val\,(S \cup \{j\}) - val(S))$$

Where:

- S = subset of features (coalition without feature j)
- p = total number of features
- val(S) = prediction using only features in S
- Weight term = $|S|!(p-|S|-1)!/p!$ accounts for all possible orderings

$$val_{\mathbf{x}}(S) = \int \hat{f}(x_1, \ldots, x_p) d\mathbb{P}_{X_C} - \mathbb{E}[\hat{f}(\mathbf{X})]$$

Where:

XC = features NOT in coalition S (marginalized out)

**Example Apartment Price Prediction**

Scenario:

- Model prediction: €300,000
- Average prediction: €310,000
- Difference (gain to distribute): −€10,000

**Feature Contributions (Shapley Values):**

- park-nearby: +€30,000 (increases value)
- area-50: +€10,000 (increases value)
- floor-2nd: €0 (no effect)
- cat-banned: −€50,000 (decreases value)

Verification:

- Sum: 30,000 + 10,000 + 0 − 50,000 = −€10,000 ✓
- Matches prediction difference (Efficiency property)

**Interpretation:**

- Cat ban most negatively impacts this apartment
- Park proximity most positively impacts
- All contributions relative to average prediction

# Shapley Values - Four Properties

**Property 1: Efficiency (Completeness)**

$$\sum_{j=1}^{p} \phi_j = \hat{f}(\mathbf{x}) - \mathbb{E}[\hat{f}(\mathbf{X})]$$

**Property 2: Symmetry**

•If two features contribute equally to all coalitions, they receive equal Shapley values
•Formally: If val(S ∪ {j}) = val(S ∪ {k}) for all S, then ϕj = ϕk

**Property 3: Dummy**

•Features not changing prediction receive Shapley value of zero
•Formally: If val(S ∪ {j}) = val(S) for all S, then ϕj = 0

**Property 4: Additivity**

•For combined models, sum Shapley values per model
•Useful for ensemble methods (e.g., random forests)

Uniqueness Theorem:

•Shapley values are ONLY solution satisfying all four properties
•Provides solid theoretical foundation

# Shapley Estimation Algorithm

**Challenge**:

•Exact computation requires 2^p coalitions (exponential)

•Infeasible for models with many features

**Solution**: Monte Carlo Sampling (Štrumbelj & Kononenko, 2014)

**Algorithm for Feature j:**

**Step 1:** Draw random instance z from dataset

**Step 2:** Choose random permutation o of all features

**Step 3**: Create x{+j}

•Features appearing before j in permutation o → take from x

•Feature j → take from x

•Features appearing after j → take from z

**Step 4**: Create x{-j}

•Same as x_{+j} but feature j also taken from z

**Step 5**: Compute marginal contribution

•ϕj^(m) = f̂(x_{+j}) − f̂(x_{-j})

**Step 6**: Repeat M times (e.g., M = 1000)

**Step 7**: Shapley value for j = average over M samples

**Note**: Repeat entire process for ALL features

$$\hat{\phi}_j = \frac{1}{M} \sum_{m=1}^{M} \left( \hat{f}(\mathbf{x}_{+j}^{(m)}) - \hat{f}(\mathbf{x}_{-j}^{(m)}) \right)$$

# Shapley Values Evaluation

**Strengths:**
•Fair distribution: Efficiency guarantees complete explanation
•Contrastive: Compare to average, subset, or single point
•Solid theory: Only method satisfying four axioms
•Model-agnostic: Works for any model with prediction function

**Limitations:**
•Computational cost: Requires approximation (2^p coalitions)
•Misinterpretation risk: Not effect of removing from training, but contribution given current features
•Not local in gradient sense: Positive value ≠ increasing feature increases prediction
•Not sparse: Uses ALL features (humans prefer selective explanations)
•Not a model: Returns values only, cannot predict for new inputs
•Unrealistic data points: When features correlated, samples unlikely combinations

# What SHAP Adds Beyond Shapley values:

- SHAP introduces new, faster methods for estimating Shapley values, making it more practical for real-world use.
- It provides a unifying theory that connects LIME, Shapley values, and other attribution methods.
- SHAP comes with a rich ecosystem of visualizations, helping you interpret and present explanations more effectively.
- It supports applications not just for tabular data, but also for text and image models.
- With SHAP, there has been a massive increase in popularity and adoption of Shapley-based explanations.
- Its key innovation is representing explanations as a **linear additive model**, where contributions from each feature (the Shapley values) sum up to approximate the model prediction locally

$$g(\mathbf{z}') = \phi_0 + \sum_{j=1}^{M} \phi_j z'_j$$

Where:

- $z' \in \{0,1\}^M$ = coalition vector (1 = feature present, 0 = absent)
- $\phi_0 = E[\hat{f}(X)]$ = base value (average prediction across dataset)
- $\phi_j$ = SHAP value for feature j (same as Shapley value)
- M = maximum coalition size

# SHAP Estimation - KernelSHAP

**Step 1**: Sample K coalition vectors z'k ∈ {0,1}^M (e.g., K=1000)
**Step 2**: Convert coalitions to original feature space
•Map z'k to valid data instance using hx
•1's → values from x, 0's → values from random instance
**Step 3**: Get predictions for all coalitions: f̂(hx(z'k))
**Step 4**: Compute SHAP kernel weights πx(z'k)
**Step 5**: Fit weighted linear regression
•Target: predictions from Step 3
•Features: coalition vectors z'k
•Weights: SHAP kernel weights
**Step 6**: Extract SHAP values from regression coefficients

$$\pi_{\mathbf{x}}(\mathbf{z}') = \frac{(M-1)}{\binom{M}{|\mathbf{z}'|}|\mathbf{z}'|(M-|\mathbf{z}'|)}$$

Where:
•|z'| = number of present features (coalition size)
•Small coalitions (few 1's) → large weight
•Large coalitions (many 1's) → large weight
•Medium coalitions → small weight
Why This Weighting?
•Learn most about features in isolation (small coalitions)
•Learn about features with all others present (large coalitions)
•Medium coalitions provide less information about individual contributions

# SHAP Estimation - TreeSHAP

**Speed Improvement:**

- Classic approach: O(TL·2^M) complexity
- TreeSHAP: $O(TL \cdot D^2)$ complexity
- T = number of trees, L = maximum leaves, D = maximum depth

**Why It's Fast:**

- Leverages tree structure
- Only considers coalitions that change predictions
- Tree with depth 5 has max 32 distinct predictions (not 2^M coalitions)

**Two Versions:**

**Interventional TreeSHAP (Current Default):**

- Computes classic Shapley values
- Uses marginal distribution P(X_S)
- Consistent with original Shapley definition

**Tree-Path Dependent TreeSHAP:**

- Computes conditional SHAP values
- Uses conditional distribution from tree structure
- Was original implementation
- Solves extrapolation but slightly different game

**Applicable To:**

- Decision trees
- Random forests
- Gradient boosted trees (XGBoost, LightGBM, CatBoost)

# SHAP Estimation - Permutation Method

**Key Idea:**

•Create random feature permutations
•Build coalitions incrementally from both directions
•Reuse predictions across coalition steps

**Algorithm:**

**Step 1**: Generate m random permutations of features (e.g., m=10)
**Step 2**: For each permutation o = (xo(1), xo(2), ..., xo(M)):
**Forward Pass (Left → Right):**
•Coalition 1: ∅
•Coalition 2: {xo(1)}
•Coalition M: {xo(1), ..., xo(M-1)}

$$\hat{\phi}_j^{(i)} = \frac{1}{2m} \sum_{k=1}^{m} (\hat{\Delta}_{o(k),j} + \hat{\Delta}_{-o(k),j})$$

**Backward Pass (Right → Left):**
•Coalition 1: ∅
•Coalition 2: {xo(M)}
•Coalition M: {x_o(2), ..., x_o(M)}

**Step 3:** Compute marginal contributions from both passes
**Step 4:** Average across all permutations:

# Visualizations

**Force Plot**

Visual representation of how features push prediction from baseline to final value

Each feature = force (arrow) pointing up or down

**Components:**

- Base value ($\phi_0$): Starting point = average prediction across dataset
- Red arrows: Features increasing prediction (push right/up)
- Blue arrows: Features decreasing prediction (push left/down)
- Final prediction: Where all forces balance

**Example (Penguin Classification):**

Base: P(Adelie) = 0.5 (average)

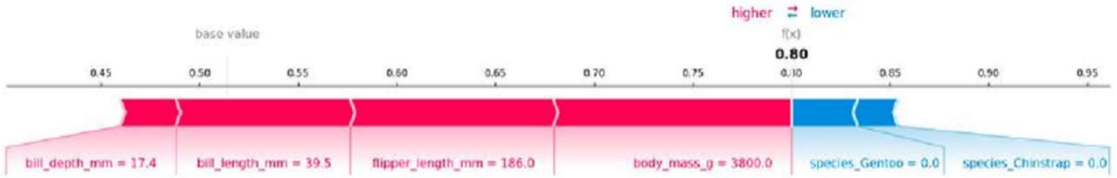
Bill length short (+0.3) → pushes toward Adelie

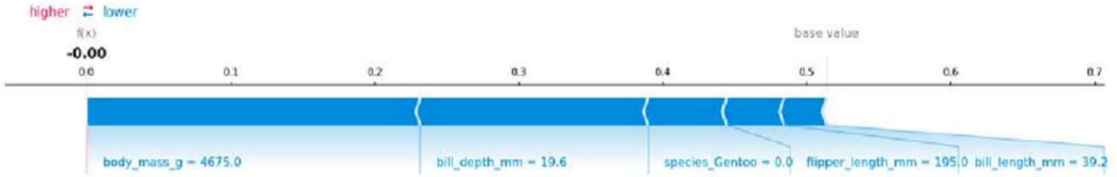
Body mass high (-0.15) → pushes away from Adelie

Final: P(Adelie) = 0.65

**When to Use:**

- Explaining single prediction to stakeholders
- Understanding which features dominated decision
- Communicating model behavior intuitively



**Strengths:**

- Solid theory: Inherits all Shapley value advantages
- Unifies methods: Connects LIME, Shapley, and other approaches
- Fast tree implementation: TreeSHAP key to widespread adoption
- Global interpretations: Feature importance, dependence, interactions, clustering
- Rich visualizations: Force, summary, dependence, interaction plots

**Limitations:**

- KernelSHAP slow: Impractical for many instances
- Feature dependence: KernelSHAP ignores correlations (like other permutation methods)
- Tree-path dependent issues: Different value function, can give non-zero values to non-influential features
- Can be misinterpreted: Same issues as Shapley values (not removal effect)
- Possible manipulation: Can create misleading explanations to hide biases (Slack et al. 2020)

| Aspect | LIME | SHAP |
|---|---|---|
| Foundation | Local linear approximation | Game theory (Shapley values) |
| Weighting | Proximity to instance (exponential kernel) | Coalition size (SHAP kernel) |
| Theory | No formal guarantees | Four axioms (efficiency, symmetry, dummy, additivity) |
| Sparsity | Select K features (sparse) | Uses all features |
| Speed | Fast (one local fit) | Varies: TreeSHAP fast, KernelSHAP slow |
| Stability | Unstable (different runs vary) | More stable (especially TreeSHAP) |
| Completeness | Partial explanation | Complete (sum = prediction difference) |
| Neighborhood | User-defined kernel (problematic) | Defined by Shapley formula |
| Global use | Limited | Rich ecosystem |
| Explanation type | Model (can predict) | Values only |

# Feature Importance

Formula:

$$I_j = \frac{1}{n}\sum_{i=1}^{n} | \phi_j^{(i)} |$$

**Calculation of feature importance:**

First, compute the SHAP values $\phi_j^{(i)}$ for **every** instance i in your dataset (with n instances)

- For each feature j, take the absolute value of its SHAP value $| \phi_j^{(i)} |$, so that positive and negative contributions both count.
- Then, average these absolute values over all instances to get the global importance $I_j$ .
- Finally, sort the features in descending order of $I_j$ features with larger average absolute SHAP values are considered more important.

**What It Measures:**

- Average magnitude of feature's impact on predictions
- NOT direction (uses absolute values)
- Features with high Ij have strong effects (positive or negative)

**Example (Penguin Classification):**

- Body mass: I = 0.25 (changes prediction by 25 percentage points on average)
- Bill length: I = 0.18
- Flipper length: I = 0.12

**Comparison to Other Methods:**

- Permutation importance: Measures performance drop when feature shuffled
- SHAP importance: Measures average magnitude of feature attributions
- Both valid, different perspectives

# Summary Plot

**What It Shows:**

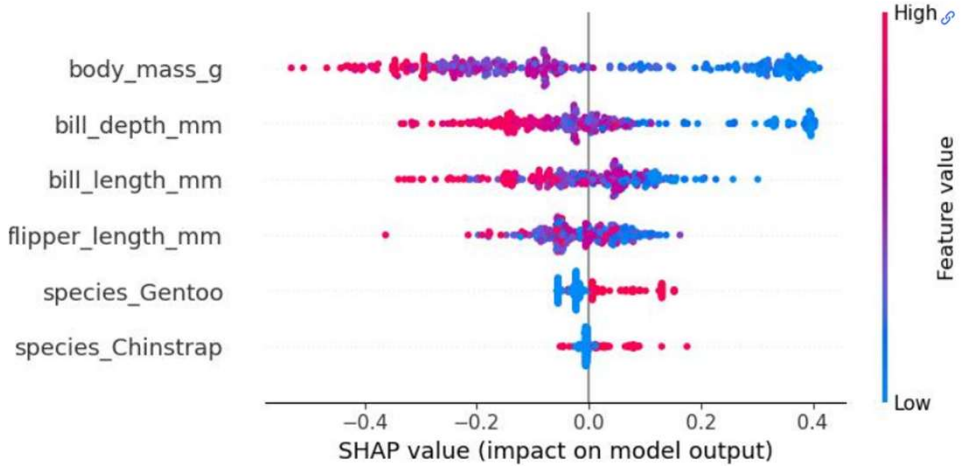
•Y-axis: Features (ordered by importance, top = most important)
•X-axis: SHAP value (impact on prediction)
•Each dot: One instance
•Color: Feature value (red = high, blue = low)
•Horizontal spread: Range of SHAP values for that feature

**How to Read It:**

Example - Body Mass (Top Feature):
•Wide horizontal spread → large range of effects
•Red dots (high body mass) on left (negative SHAP) → decreases P(female)
•Blue dots (low body mass) on right (positive SHAP) → increases P(female)
•Clear pattern: higher value → negative effect

**What You Learn:**

1.Feature importance (Y-axis ordering)
2.Direction of effects (X-axis: left=decrease, right=increase)
3.Relationship between feature value and effect (color pattern)
4.Variance in effects (horizontal spread)



# Dependence Plot

What You Plot:

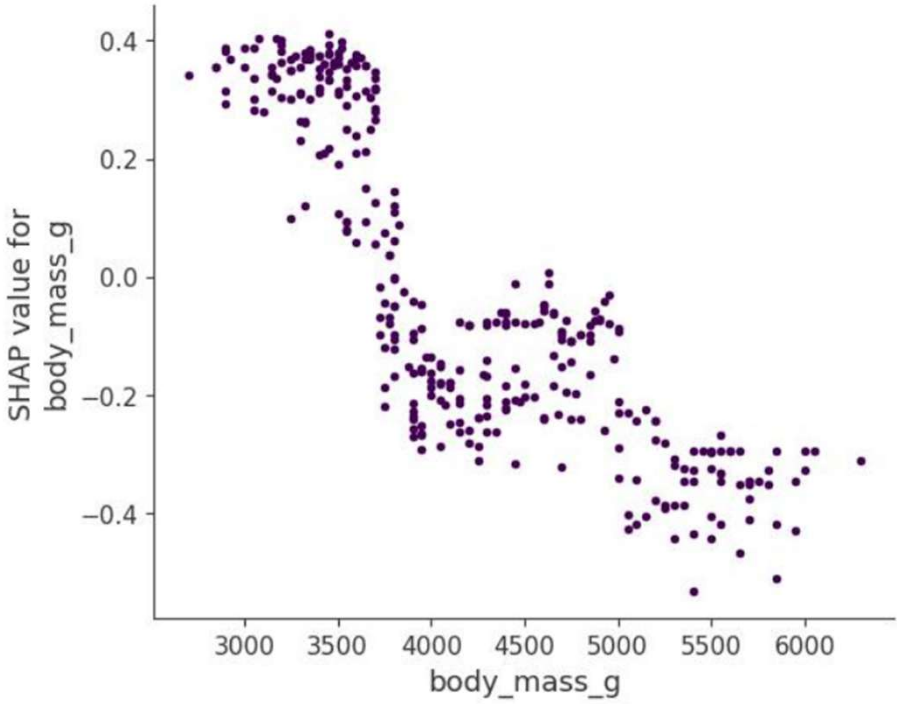
- X-axis: Feature value ($x_j^{(i)}$)
- Y-axis: SHAP value for that feature ($\phi_j^{(i)}$)
- Each point: One instance from dataset

Mathematical Representation:

- Dataset of points: $\{(x_j^{(1)}, \phi_j^{(1)}), (x_j^{(2)}, \phi_j^{(2)}), ..., (x_j^{(n)}, \phi_j^{(n)})\}$

Example - Body Mass:

- X-axis: Body mass (3000g to 6000g)
- Y-axis: SHAP value for body mass
- Pattern: Clear negative relationship
- Interpretation: Heavier penguins → lower P(female)



# Interaction Values

**Shapley Interaction Index:**

$$\phi_{i,j} = \sum_{S \subseteq \{1,\ldots,M\} \setminus \{i,j\}} \frac{|S|!\,(M - |S| - 2)!}{2(M-1)!}\,\delta_{ij}(S)$$

Where:

$$\delta_{ij}(S) = \hat{f}(S \cup \{i,j\}) - \hat{f}(S \cup \{i\}) - \hat{f}(S \cup \{j\}) + \hat{f}(S)$$

**What It Measures:**

- Pure interaction effect between features i and j
- Effect of having BOTH features beyond their individual contributions
- Positive: Features amplify each other
- Negative: Features cancel each other

**Output:**

- M × M matrix per instance
- Diagonal: Main effects (individual SHAP values)
- Off-diagonal: Pairwise interactions

**Practical Use:**

1. Identify strongest interactions automatically
2. Color dependence plots by interaction feature
3. Understand complex feature relationships