DIGITAL FINANCE

This project has received funding from the Horizon Europe research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 101119635

Schweizerische Eidgenossenschaft
Confédération suisse
Confederazione Svizzera
Confederaziun svizra

State Secretariat for Education, Research and Innovation SERI

Funded by the European Union

MARIE CURIE ACTIONS

DIGITAL

Explainable AI

XAI Methods (**SHAP, LIME**) Deep Dive

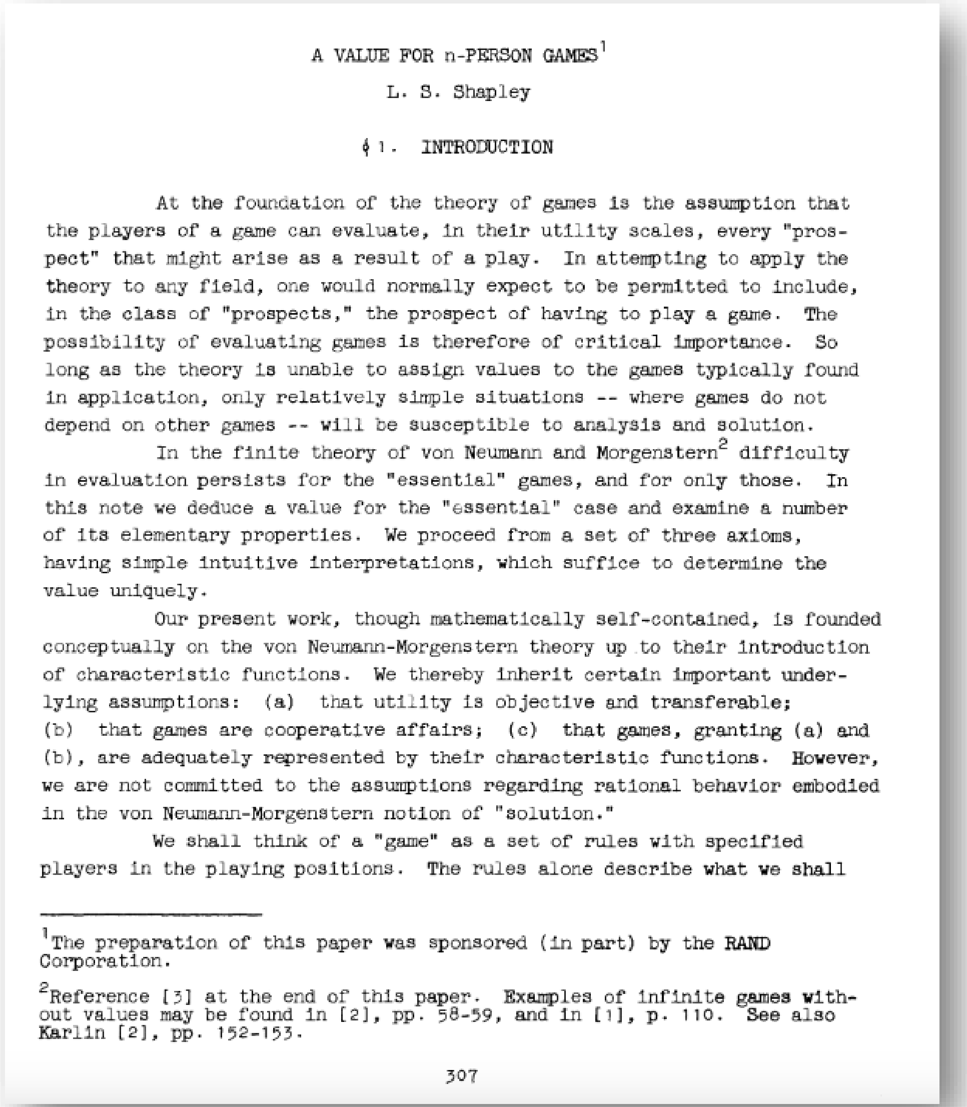
Prof. Dr. Branka Hadji Misheva
Bern University of Applied Science **(BFH)**

Funded by
the European Union

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

The original Shapley values article by Lloyd Shapley (1953) is **foundational for explainable AI (XAI)**, as it provides a game-theoretic framework that underpins many of today's most widely used model explanation methods.

Shapley, L. (1953) ***A Value for n-Person Games***. In: Kuhn, H. and Tucker, A., Eds., Contributions to the Theory of Games II, Princeton University Press, Princeton, 307-317.
https://doi.org/10.1515/9781400881970-018



A VALUE FOR n-PERSON GAMES[1]

L. S. Shapley

§ 1. INTRODUCTION

At the foundation of the theory of games is the assumption that the players of a game can evaluate, in their utility scales, every "prospect" that might arise as a result of a play. In attempting to apply the theory to any field, one would normally expect to be permitted to include, in the class of "prospects," the prospect of having to play a game. The possibility of evaluating games is therefore of critical importance. So long as the theory is unable to assign values to the games typically found in application, only relatively simple situations -- where games do not depend on other games -- will be susceptible to analysis and solution.

In the finite theory of von Neumann and Morgenstern[2] difficulty in evaluation persists for the "essential" games, and for only those. In this note we deduce a value for the "essential" case and examine a number of its elementary properties. We proceed from a set of three axioms, having simple intuitive interpretations, which suffice to determine the value uniquely.

Our present work, though mathematically self-contained, is founded conceptually on the von Neumann-Morgenstern theory up to their introduction of characteristic functions. We thereby inherit certain important underlying assumptions: (a) that utility is objective and transferable; (b) that games are cooperative affairs; (c) that games, granting (a) and (b), are adequately represented by their characteristic functions. However, we are not committed to the assumptions regarding rational behavior embodied in the von Neumann-Morgenstern notion of "solution."

We shall think of a "game" as a set of rules with specified players in the playing positions. The rules alone describe what we shall

[1]The preparation of this paper was sponsored (in part) by the RAND Corporation.

[2]Reference [3] at the end of this paper. Examples of infinite games without values may be found in [2], pp. 58-59, and in [1], p. 110. See also Karlin [2], pp. 152-153.

307

(Mathematical) Background

- Shapley (1953) provides a framework for a **fair distribution** of payout in a collaborative game where players work together for a common goal but maybe do not contribute **equally**.

What properties would a fair distribution of payouts have?

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

(Mathematical) Background

Efficiency	The sum of all players' contribution must be **equal** to the payout
Additivity	In a game with multiple subgames, each having a **separate payout**, the contribution of a player to the combined game is **equal to the sum** of contributions to each individual subgame
Null Player	If a player **does not contribute** to any coalition, their share of the playout is **0**
Symmetry	If two players contribute the **same** to all coalitions, they should receive **equal** payout

The **Math**

$$\boxed{\varphi_i(v)} = \sum_{S \subseteq N\{i\}} \frac{|S|!\,(n - |S| - 1)!}{n!} \left(v(S \cup \{i\}) - v(S)\right)$$

Shapley value for a given player i

We calculate the contribution of each player to a game

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

The **Math**

$$\varphi_i(v) = \boxed{\sum_{S \subseteq N \{i\}}} \frac{|S|!(n-|S|-1)!}{n!}(v(S \cup \{i\}) - v(S))$$

Sum over all possible coalitions that do not contain i

The Shapley value aims to measure the average contribution of player i to the game, **considering all possible scenarios where i could join a coalition**

DIGITAL

The **Math**

$$\varphi_i(v) = \sum_{S \subseteq N\{i\}} \frac{|S|!(n-|S|-1)!}{n!}(v(S \cup \{i\}) - \boxed{v(S)})$$

Coalition without player i

The **Math**

$$\varphi_i(v) = \sum_{S \subseteq N\{i\}} \frac{|S|!\,(n - |S| - 1)!}{n!} \boxed{(v(S \cup \{i\})} - v(S))$$

Coalition with player i

The **Math**

$$\varphi_i(v) = \sum_{S \subseteq N\{i\}} \frac{|S|!\,(n-|S|-1)!}{n!} \boxed{(v(S \cup \{i\}) - v(S))}$$

Marginal contribution of i to the coalition

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

The **Math**

$$\varphi_i(v) = \sum_{S \subseteq N\{i\}} \boxed{\frac{|S|!\,(n-|S|-1)!}{n!}}\,(v(S \cup \{i\}) - v(S))$$

Weighting the contributions of i by its share in the number of total coalitions

- |S| is the **size of the coalition** S (excluding feature i)
- n is the total **number of players**

The **Math**

Pretend we line up all the players in a random order, and then measure how much player i adds when it joins the players before it.

A coalition S is just the set of players that happen to be before i in that ordering.

The **weight is the probability that S shows up before i**.

$$\varphi_i(v) = \sum_{S \subseteq N\{i\}} \boxed{\frac{|S|!\,(n - |S| - 1)!}{n!}} (v(S \cup \{i\}) - v(S))$$

If i is first

- i comes in right at the start of the ordering
- **Only one possible coalition – hence, takes the full weight**

If i is in the middle

- i comes in somewhere in the middle
- **More possible coalitions – hence, the weight is divided between them**

If i is last

- i comes in at the very end of the ordering
- **Only one possible coalition – hence, takes the full weight**

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Let's calculate **manually** …

Let's imagine a case in which we are combining different drugs (**A**, **B** and **C**), and we want to calculate the contribution to each drug on the likelihood of surviving.

All together	Separately	Pair-wise coalitions
When Drug A, B and C are given together, this leads to a **survival likelihood of 90%.**	Drug A: 40% Drug B: 50% Drug C: 60%	Drug A and B: 70% Drug A and C: 65% Drug B and C: 80%

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Let's calculate **manually** …

HINT: Start by identifying all coalitions to which **Drug A** can can be added. Then for each, apply the formula: $\phi_i(v) = \sum_{S \subseteq N\{i\}} \frac{|S|!(n-|S|-1)!}{n!}(v(S \cup \{i\}) - v(S))$

What is the fair contribution of **Drug A** to the triple combination likelihood of 90%?

Use Shapley Values to Explain **Predictor Importance**

- The literature offers many attempts to use **Shapley values for the purpose of fairly quantifying the contribution of features to a prediction task**.

- Let's consider a simple case: a multivariate regression case!

$$y_i = ß_1 X_{1i} + ß_2 X_{2i} + ß_3 X_{3i} + \ \dots . + ß_n X_{ni} + \ e_i$$

What are we usually interested in besides prediction accuracy?

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Use Shapley Values to Explain **Predictor Importance**

- The literature offers many attempts to use **Shapley values for the purpose of fairly quantifying the contribution of features to a prediction task**.
- Let's consider a simple case: a multivariate regression case!

$$y_i = ß_1 X_{1i} + ß_2 X_{2i} + ß_3 X_{3i} + \ .... + ß_n X_{ni} + e_i$$

What do you think happens to regression coefficients when predictors are highly correlated?

Use Shapley Values to Explain **Predictor Importance**

$$y_i = ß_1 X_{1i} + ß_2 X_{2i} + ß_3 X_{3i} + \; \ldots . + ß_n X_{ni} + \; e_i$$

- coefficients become **very large** (inflated standard errors)
- **flip signs** (e.g., one positive, one negative, even though both predictors are positively correlated with Y)
- or **change dramatically** with small changes in the data

Suppose X1 and X2 are both positively correlated with Y, but also highly correlated with each other. What might happen if we include both in a regression?

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Use Shapley Values to Explain **Predictor Importance**

$$y_i = ß_1 X_{1i} + ß_2 X_{2i} + ß_3 X_{3i} + \; \ldots . + ß_n X_{ni} + \; e_i$$

If we can't trust the signs or magnitude of coefficients, how can we trust our variable importance ranking?

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Use Shapley Values to Explain **Predictor Importance**

Lipovetsky and Conklin (2001): propose using **Shapley values to compute each predictor's average marginal contribution to $\boldsymbol{R^2}$** over all possible subsets:

$$\varphi_i = \sum_{S \subseteq N\{i\}} \frac{|S|!(n-|S|-1)!}{n!} \left(\boxed{R^2}(S \cup \{i\}) - \boxed{R^2}(S)\right)$$

- If predictors are correlated, the *total* variance explained doesn't change.
- What we can change is how that variance gets attributed across predictors.
- A Shapley decomposition of $\boldsymbol{R^2}$ produces stable, positive contributions that **always sum exactly to $\boldsymbol{R^2}$** even when predictors are correlated.

In practice …

Suppose you have predictors A, B, and C.

STEP 1 **Calculate marginal contributions**

- **Subsets:** Ø, A, B, C, AB, AC, BC, ABC
- For each subset, compute R2 of the model.
  - A: The marginal contribution of A to the subset BC, for example, is: R^2(ABC)− R^2(BC). You then average A's marginal contribution across all such subsets where A appears. That gives A's Shapley Value. **Repeat for all variables.**

STEP 2 **Propose adjusted coefficients**

- After computing Shapley Values for each variable, **calculate adjusted coefficients** for each feature so that they reflect these stable net effects.
- Assume standardized data (so each predictor and the target has mean 0 and SD 1), then define:

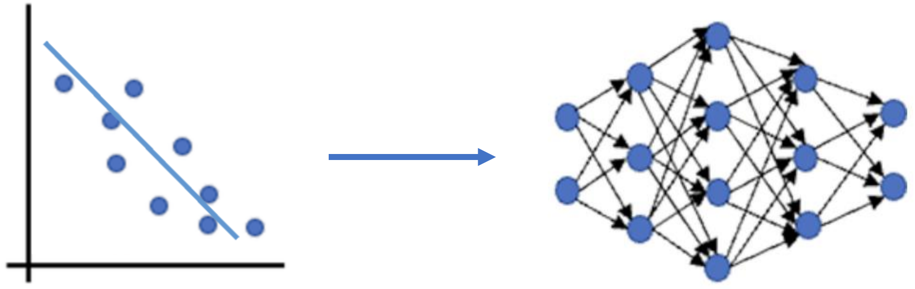
$$a_j = \frac{SV_j}{r_j}$$

where SV_j is the Shapley-based net effect & r_j is the correlation between x_j and y

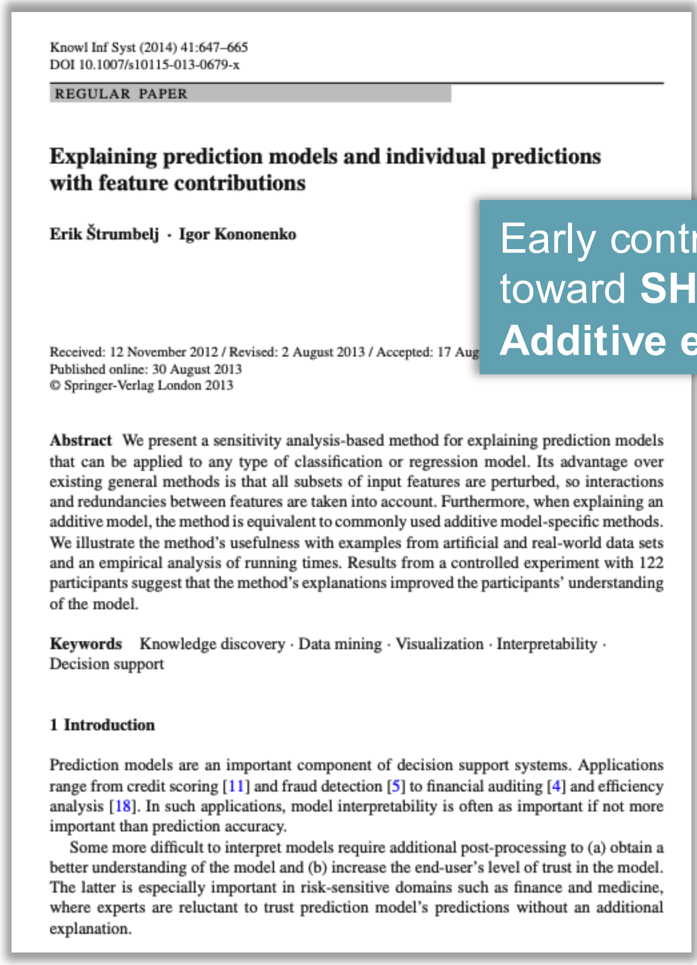
DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Shapley Values for **ML models:** early work



Among the first attempts to generalize Shapley ideas to **ANY** predictive model, neural nets, SVMs, decision trees, whatever you throw at it.



Knowl Inf Syst (2014) 41:647–665
DOI 10.1007/s10115-013-0679-x

REGULAR PAPER

Explaining prediction models and individual predictions with feature contributions

Erik Štrumbelj · Igor Kononenko

Received: 12 November 2012 / Revised: 2 August 2013 / Accepted: 17 Aug
Published online: 30 August 2013
© Springer-Verlag London 2013

Abstract We present a sensitivity analysis-based method for explaining prediction models that can be applied to any type of classification or regression model. Its advantage over existing general methods is that all subsets of input features are perturbed, so interactions and redundancies between features are taken into account. Furthermore, when explaining an additive model, the method is equivalent to commonly used additive model-specific methods. We illustrate the method's usefulness with examples from artificial and real-world data sets and an empirical analysis of running times. Results from a controlled experiment with 122 participants suggest that the method's explanations improved the participants' understanding of the model.

Keywords Knowledge discovery · Data mining · Visualization · Interpretability · Decision support

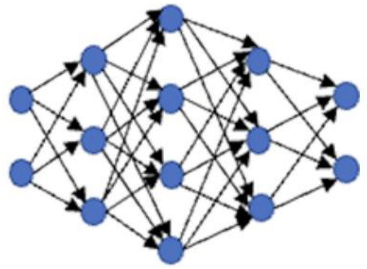
1 Introduction

Prediction models are an important component of decision support systems. Applications range from credit scoring [11] and fraud detection [5] to financial auditing [4] and efficiency analysis [18]. In such applications, model interpretability is often as important if not more important than prediction accuracy.

Some more difficult to interpret models require additional post-processing to (a) obtain a better understanding of the model and (b) increase the end-user's level of trust in the model. The latter is especially important in risk-sensitive domains such as finance and medicine, where experts are reluctant to trust prediction model's predictions without an additional explanation.

Early contributions toward **SHAP (SHapley Additive exPlanations)**

Shapley Values for **ML models:** early work



We need a way to treat the **model like a black box: perturb the inputs, see how the outputs change, and use the Shapley framework to fairly allocate contributions.**

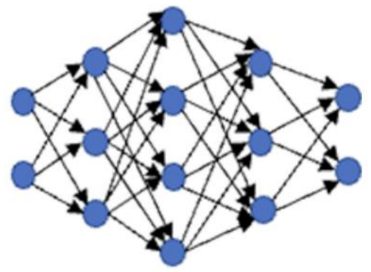
Doing this exactly would require checking every possible subset of features, which is **exponential in cost.**

The authors propose a sampling-based approach.

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Shapley Values for **ML models:** early work



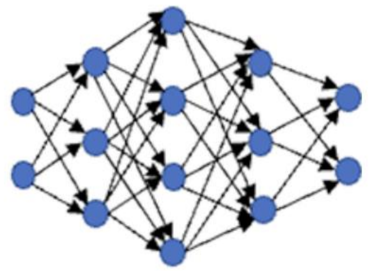
The authors propose a sampling-based approach.

1. We don’t consider all possible coalitions but only a subset.

2. We don’t retrain models.

3. We evaluate what the model would produce for an instance using different coalitions of features. Thus, we evaluate the model with a combination of **known** and **unknown** features.

Shapley Values for **ML models:** early work

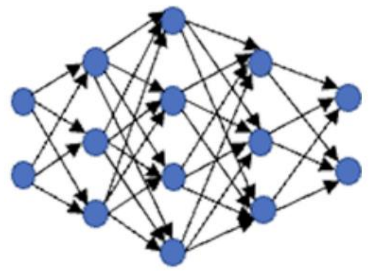


The authors propose a sampling-based approach.

Imagine that you have trained a model to predict Y based on three features: A, B, C. Evaluating the contribution of A, requires us checking what the model would produce under different coalitions – if only A is **known**, if A and B are **known** …

But the model was trained on all three features. How do you evaluate what the model would give if you assume that only A is **known**? What value do you give to B and C?

Shapley Values for **ML models:** early work



The authors propose a sampling-based approach.

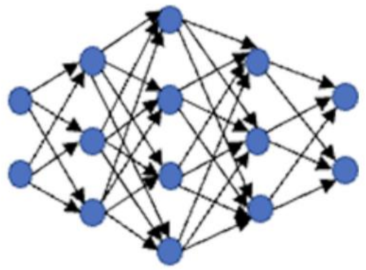
Imagine that you have trained a model to predict Y based on three features: A, B, C. Evaluating the contribution of A, requires us checking what the model would produce under different coalitions – if only A is known, if A and B are known …

We use **BACKGROUND DATA** → sample values for the **unknown** features from the training dataset

We approximate the model's expected prediction for any coalition of known features.

Those expectations are not something we can calculate exactly, because in principle we'd have to integrate over all possible combinations of the unknown features.

Shapley Values for **ML models:** early work



The authors propose a sampling-based approach.

If we just **sample blindly**, some features will converge quickly, and others, especially those with more variable effects, will take a lot longer.

Adaptive sampling

- We don't need the same number of samples for every feature.
- Adaptive sampling monitors the variance of the contribution estimates.

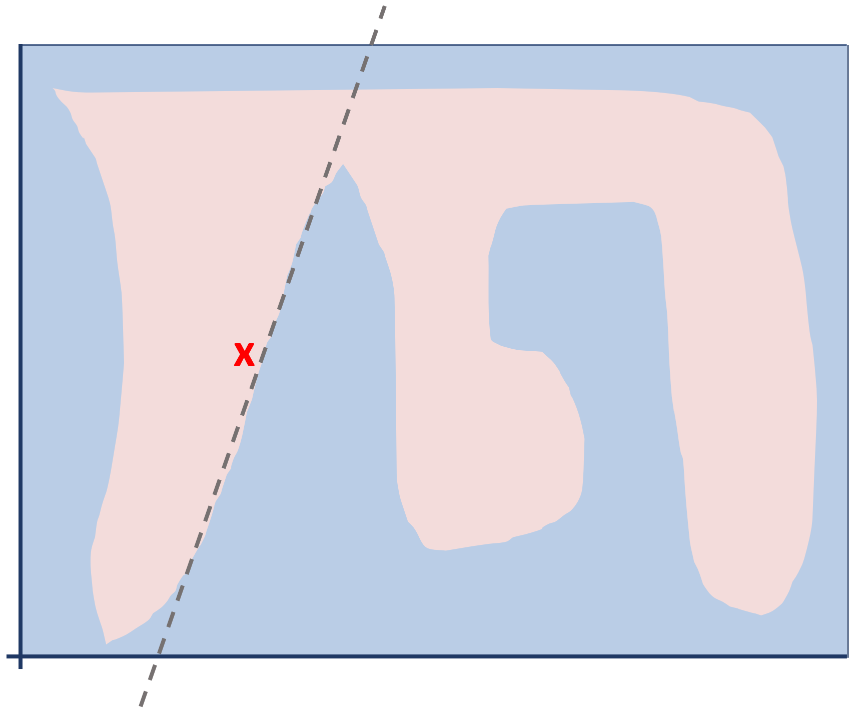
Quasi-random sampling

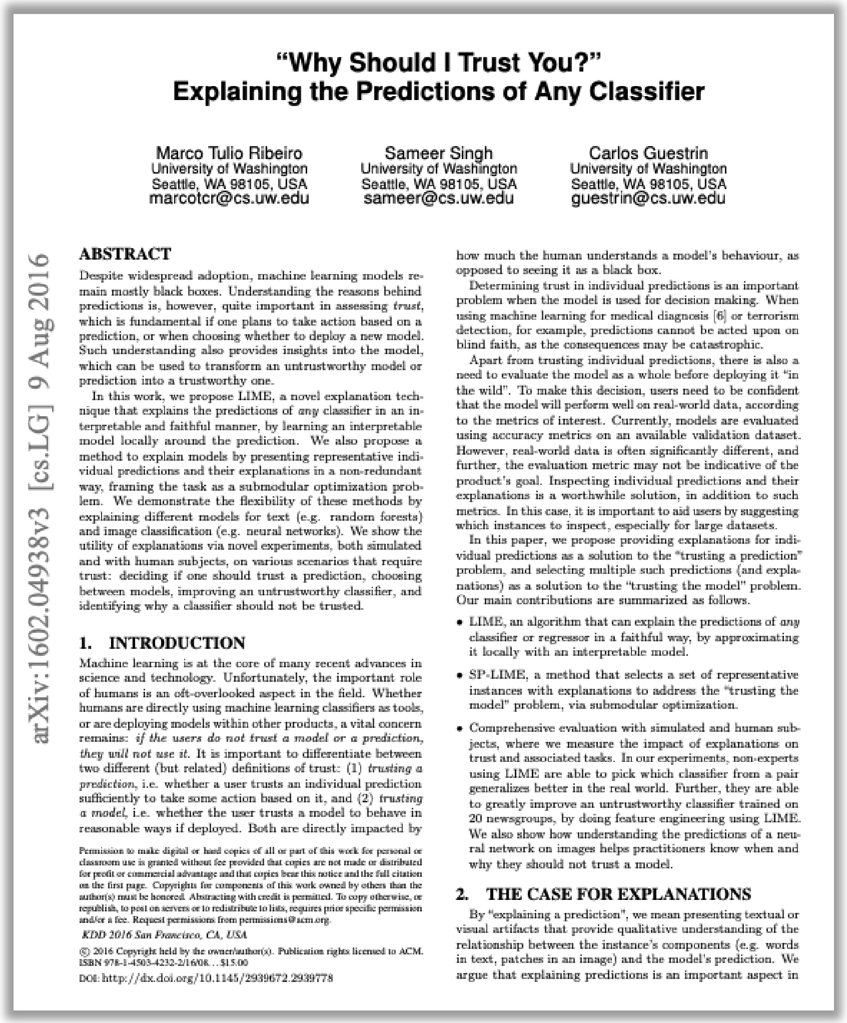
- Use Sobol sequences.
- Cover the space more evenly than ordinary random draws.

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

LIME





“Why Should I Trust You?” Explaining the Predictions of Any Classifier

Marco Tulio Ribeiro
University of Washington
Seattle, WA 98105, USA
marcotcr@cs.uw.edu

Sameer Singh
University of Washington
Seattle, WA 98105, USA
sameer@cs.uw.edu

Carlos Guestrin
University of Washington
Seattle, WA 98105, USA
guestrin@cs.uw.edu

ABSTRACT

Despite widespread adoption, machine learning models remain mostly black boxes. Understanding the reasons behind predictions is, however, quite important in assessing *trust*, which is fundamental if one plans to take action based on a prediction, or when choosing whether to deploy a new model. Such understanding also provides insights into the model, which can be used to transform an untrustworthy model or prediction into a trustworthy one.

In this work, we propose LIME, a novel explanation technique that explains the predictions of *any* classifier in an interpretable and faithful manner, by learning an interpretable model locally around the prediction. We also propose a method to explain models by presenting representative individual predictions and their explanations in a non-redundant way, framing the task as a submodular optimization problem. We demonstrate the flexibility of these methods by explaining different models for text (e.g. random forests) and image classification (e.g. neural networks). We show the utility of explanations via novel experiments, both simulated and with human subjects, on various scenarios that require trust: deciding if one should trust a prediction, choosing between models, improving an untrustworthy classifier, and identifying why a classifier should not be trusted.

1. INTRODUCTION

Machine learning is at the core of many recent advances in science and technology. Unfortunately, the important role of humans is an oft-overlooked aspect in the field. Whether humans are directly using machine learning classifiers as tools, or are deploying models within other products, a vital concern remains: *if the users do not trust a model or a prediction, they will not use it*. It is important to differentiate between two different (but related) definitions of trust: (1) *trusting a prediction*, i.e. whether a user trusts an individual prediction sufficiently to take some action based on it, and (2) *trusting a model*, i.e. whether the user trusts a model to behave in reasonable ways if deployed. Both are directly impacted by how much the human understands a model's behaviour, as opposed to seeing it as a black box.

Determining trust in individual predictions is an important problem when the model is used for decision making. When using machine learning for medical diagnosis [6] or terrorism detection, for example, predictions cannot be acted upon on blind faith, as the consequences may be catastrophic.

Apart from trusting individual predictions, there is also a need to evaluate the model as a whole before deploying it “in the wild”. To make this decision, users need to be confident that the model will perform well on real-world data, according to the metrics of interest. Currently, models are evaluated using accuracy metrics on an available validation dataset. However, real-world data is often significantly different, and further, the evaluation metric may not be indicative of the product's goal. Inspecting individual predictions and their explanations is a worthwhile solution, in addition to such metrics. In this case, it is important to aid users by suggesting which instances to inspect, especially for large datasets.

In this paper, we propose providing explanations for individual predictions as a solution to the “trusting a prediction” problem, and selecting multiple such predictions (and explanations) as a solution to the “trusting the model” problem. Our main contributions are summarized as follows.

- LIME, an algorithm that can explain the predictions of *any* classifier or regressor in a faithful way, by approximating it locally with an interpretable model.
- SP-LIME, a method that selects a set of representative instances with explanations to address the “trusting the model” problem, via submodular optimization.
- Comprehensive evaluation with simulated and human subjects, where we measure the impact of explanations on trust and associated tasks. In our experiments, non-experts using LIME are able to pick which classifier from a pair generalizes better in the real world. Further, they are able to greatly improve an untrustworthy classifier trained on 20 newsgroups, by doing feature engineering using LIME. We also show how understanding the predictions of a neural network on images helps practitioners know when and why they should not trust a model.

2. THE CASE FOR EXPLANATIONS

By “explaining a prediction”, we mean presenting textual or visual artifacts that provide qualitative understanding of the relationship between the instance's components (e.g. words in text, patches in an image) and the model's prediction. We argue that explaining predictions is an important aspect in

KDD 2016 San Francisco, CA, USA

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4232-2/16/08 . . . $15.00

DOI: http://dx.doi.org/10.1145/2939672.2939778

LIME: **Formally**

$$\xi(x) = argmin_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

- $\xi(x)$ is the **explanation function.**
- f is the **black-box model** we want to explain.
- $g \in G$ represents the set of **interpretable models** (e.g., linear regression, decision trees).
- $L(f, g, \pi_x)$ is the **loss function.**
- π_x is the **proximity function.**
- $\Omega(g)$ is a **complexity penalty**.

Molnar (2023)

LIME: **Formally**

$$\xi(x) = argmin_{g \in G} L(f, g, \pi_x) + \Omega(g) \qquad (1)$$

$$\boldsymbol{L(f, g, \pi_x)} = \sum_i \boxed{\pi_x}(x_i)\big(f(x_i) - g(x_i)\big)^2 \quad (2)$$

- We want to ensure that the interpretable model g approximates the black-box model f **locally**. The typical choice is the **weighted squared error**.
- x_i are the perturbed samples around x.
- $\pi_x(x_i)$ are their proximity weights.

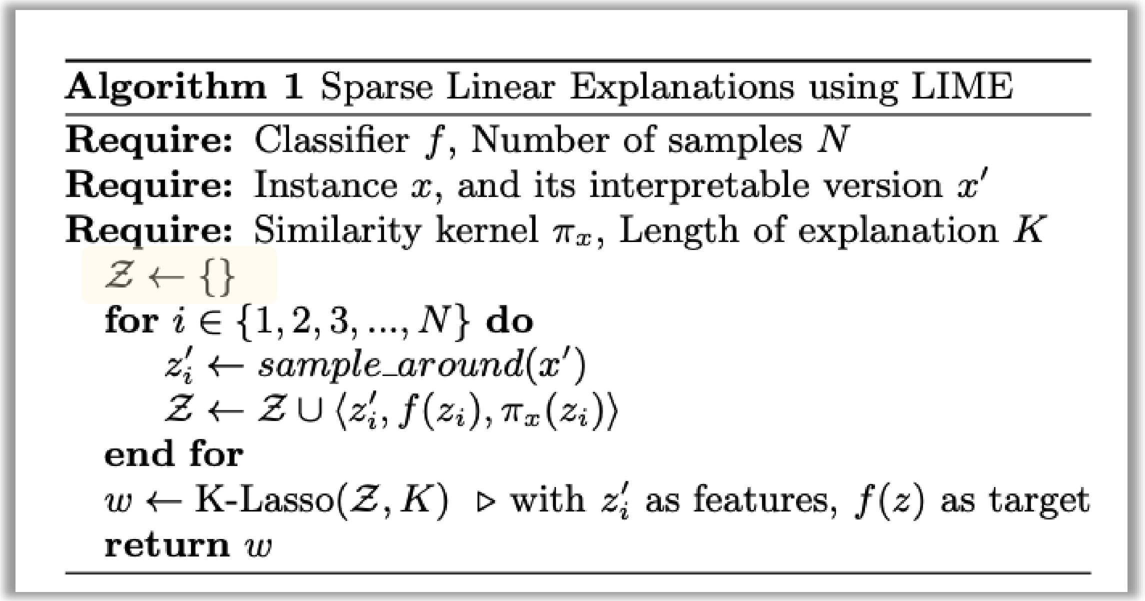
Molnar (2023)

LIME: **Formally**

$$\xi(x) = argmin_{g \in G} L(f, g, \pi_x) + \Omega(g) \qquad (1)$$

- Complexity parameter.
- Prevents the local model g from being too complex.
- Encourages simpler explanations (e.g., fewer features in a linear model).
  - Example: If g is a linear model, $\boldsymbol{\Omega(g)}$ could be the number of non-zero coefficients.

LIME: Algo

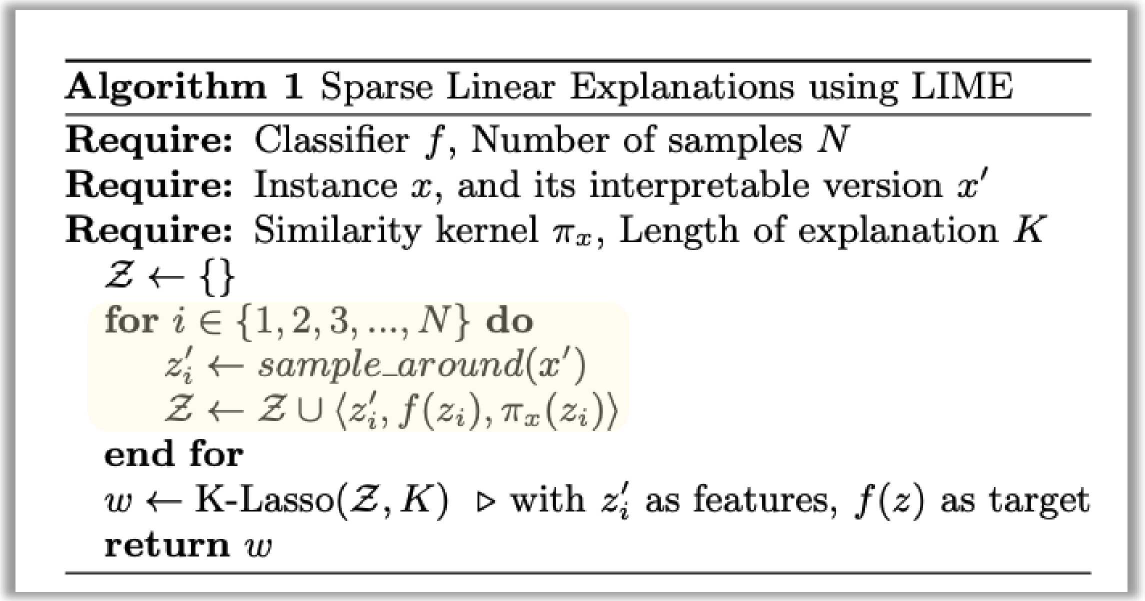


Algorithm 1 Sparse Linear Explanations using LIME

```
Require: Classifier $f$, Number of samples $N$
Require: Instance $x$, and its interpretable version $x'$
Require: Similarity kernel $\pi_x$, Length of explanation $K$
    $\mathcal{Z} \leftarrow \{\}$
    for $i \in \{1, 2, 3, ..., N\}$ do
        $z'_i \leftarrow sample\_around(x')$
        $\mathcal{Z} \leftarrow \mathcal{Z} \cup \langle z'_i, f(z_i), \pi_x(z_i) \rangle$
    end for
    $w \leftarrow$ K-Lasso($\mathcal{Z}, K$)   $\triangleright$ with $z'_i$ as features, $f(z)$ as target
    return $w$
```

Step 1. Initialize an empty dataset

LIME: Algo



```
Algorithm 1 Sparse Linear Explanations using LIME
Require: Classifier $f$, Number of samples $N$
Require: Instance $x$, and its interpretable version $x'$
Require: Similarity kernel $\pi_x$, Length of explanation $K$
  $\mathcal{Z} \leftarrow \{\}$
  for $i \in \{1, 2, 3, ..., N\}$ do
      $z'_i \leftarrow sample\_around(x')$
      $\mathcal{Z} \leftarrow \mathcal{Z} \cup \langle z'_i, f(z_i), \pi_x(z_i) \rangle$
  end for
  $w \leftarrow$ K-Lasso($\mathcal{Z}, K$)  ▷ with $z'_i$ as features, $f(z)$ as target
  return $w$
```

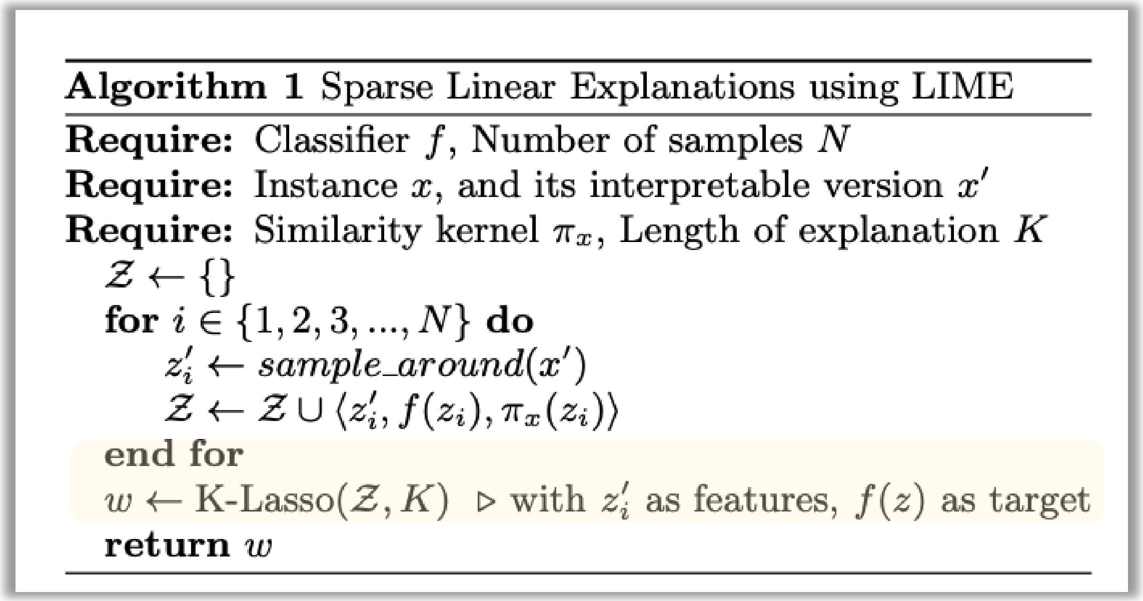
Step 1. Initialize an empty dataset

Step 2. For each of the N samples:
- Generate perturbed sample z'_i around x' using sample_around(x')
- Get:
  - The prediction
  - The similarity $\pi_x(z_i)$

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

LIME: **Algo**



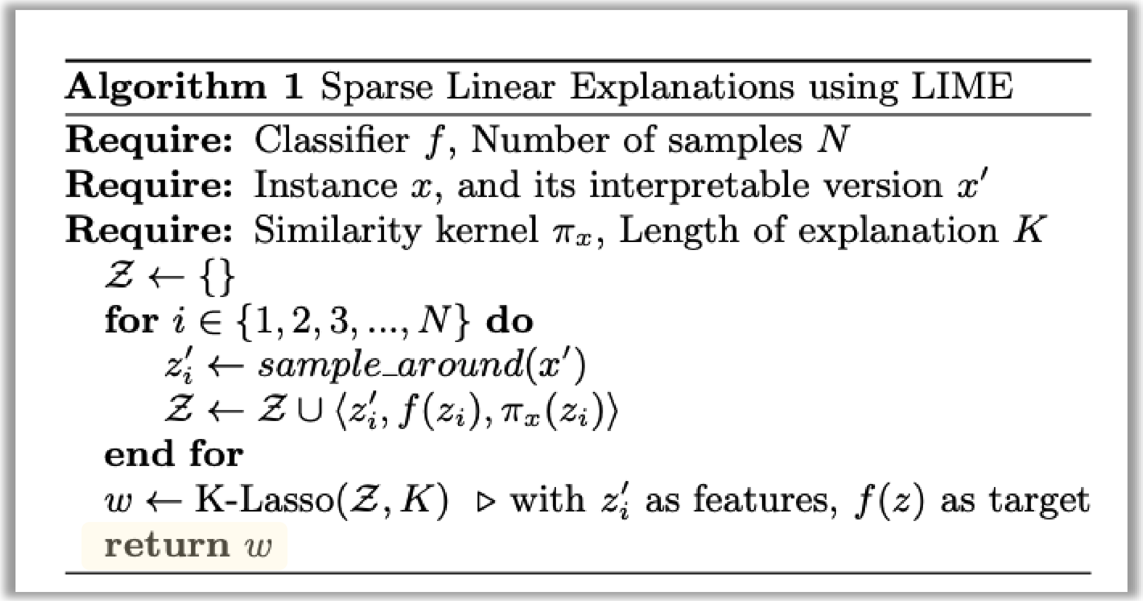
```
Algorithm 1 Sparse Linear Explanations using LIME
Require: Classifier $f$, Number of samples $N$
Require: Instance $x$, and its interpretable version $x'$
Require: Similarity kernel $\pi_x$, Length of explanation $K$
  $\mathcal{Z} \leftarrow \{\}$
  for $i \in \{1, 2, 3, ..., N\}$ do
      $z'_i \leftarrow sample\_around(x')$
      $\mathcal{Z} \leftarrow \mathcal{Z} \cup \langle z'_i, f(z_i), \pi_x(z_i) \rangle$
  end for
  $w \leftarrow$ K-Lasso$(\mathcal{Z}, K)$  ▷ with $z'_i$ as features, $f(z)$ as target
  return $w$
```

Step 1. Initialize an empty dataset

Step 2. For each of the N samples:
- Generate perturbed sample z'_i around x' using sample_around(x')
- Get:
  - The prediction
  - The similarity $\pi_x(z_i)$

Step 3. Fit the weighted linear model using K-Lasso
- Use z'_i as features
- Use $f(z_i)$ as target
- Weigh each sample using the $\pi_x(z_i)$
- Restrict to k features

LIME: Algo



Algorithm 1 Sparse Linear Explanations using LIME

```
Require: Classifier $f$, Number of samples $N$
Require: Instance $x$, and its interpretable version $x'$
Require: Similarity kernel $\pi_x$, Length of explanation $K$
  $\mathcal{Z} \leftarrow \{\}$
  for $i \in \{1, 2, 3, ..., N\}$ do
      $z'_i \leftarrow sample\_around(x')$
      $\mathcal{Z} \leftarrow \mathcal{Z} \cup \langle z'_i, f(z_i), \pi_x(z_i) \rangle$
  end for
  $w \leftarrow$ K-Lasso($\mathcal{Z}, K$)  ▷ with $z'_i$ as features, $f(z)$ as target
  return $w$
```

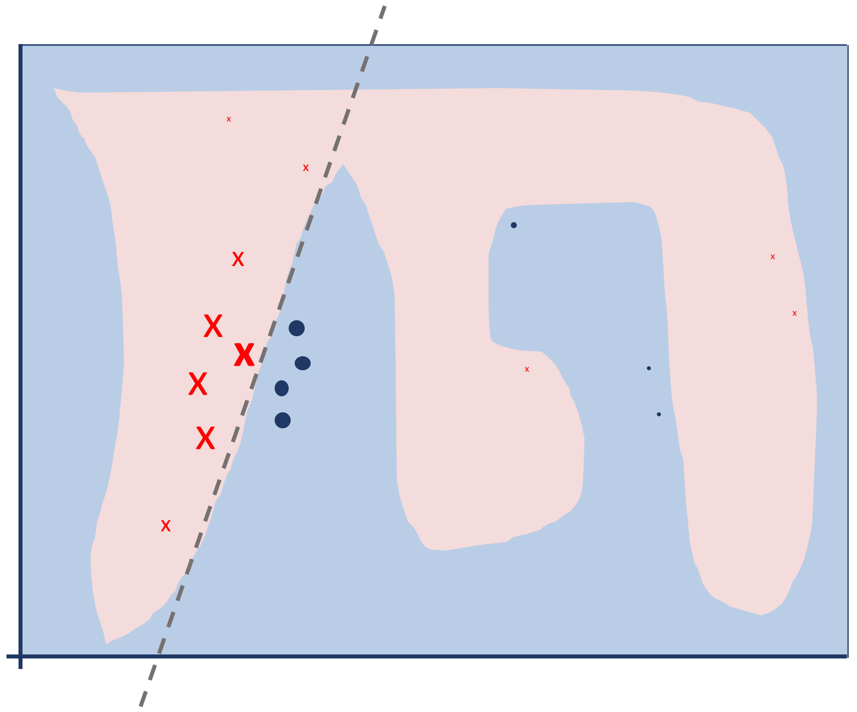
Step 1. Initialize an empty dataset

Step 2. For each of the N samples:
- Generate perturbed sample z'_i around x' using sample_around(x')
- Get:
  - The prediction
  - The similarity $\pi_x(z_i)$

Step 3. Fit the weighted linear model using K-Lasso
- Use z'_i as features
- Use $f(z_i)$ as target
- Weigh each sample using the $\pi_x(z_i)$
- Restrict to k features

Step 4. Return w: weights i.e. local explanations

LIME: So, what are **all the steps**?



Pick an observation, **create and permute data**

Calculate similarity between the original observations and the permutations

Make predictions on new data using your black box

Fit a simple model to the permuted data with k features and similarity scores as weights

Coefficients from the simple model serve as an explanation of the model behavior at the local level

LIME: So, what are **all the steps**?

Sampling Step

Pick an observation, **create and permute data**

Weighting Step

Calculate similarity between the original observations and the permutations

Local Model Step

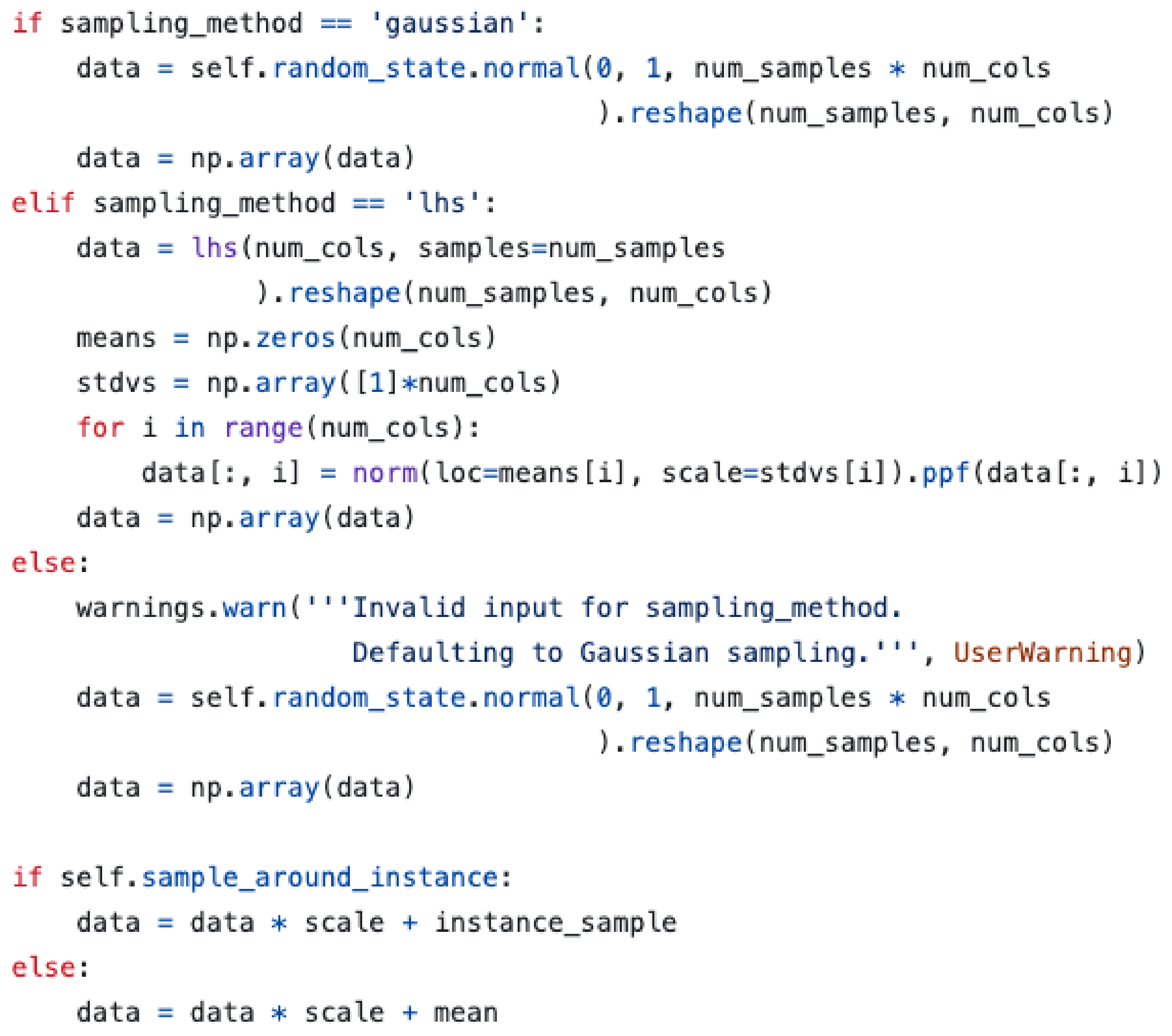
Make predictions on new data using your black box

Fit a simple model to the permuted data with n features and similarity scores as weights

Generation step (tabular data)

lime/lime/lime_tabular.py

- LIME **samples each feature independently** from a normal distribution **centred at the (instance's) feature value**

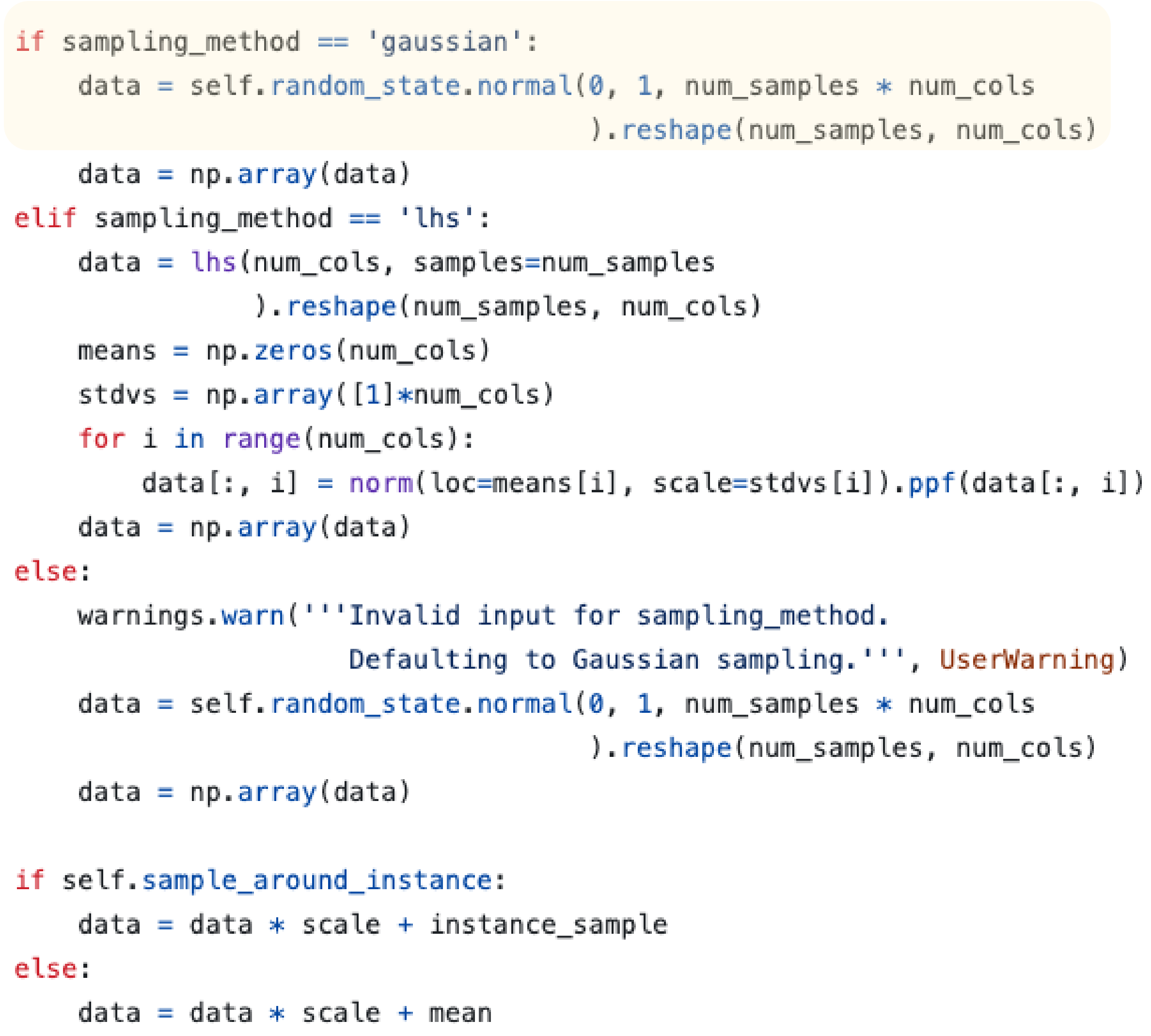


```
if sampling_method == 'gaussian':
    data = self.random_state.normal(0, 1, num_samples * num_cols
                                    ).reshape(num_samples, num_cols)
    data = np.array(data)
elif sampling_method == 'lhs':
    data = lhs(num_cols, samples=num_samples
               ).reshape(num_samples, num_cols)
    means = np.zeros(num_cols)
    stdvs = np.array([1]*num_cols)
    for i in range(num_cols):
        data[:, i] = norm(loc=means[i], scale=stdvs[i]).ppf(data[:, i])
    data = np.array(data)
else:
    warnings.warn('''Invalid input for sampling_method.
                     Defaulting to Gaussian sampling.''', UserWarning)
    data = self.random_state.normal(0, 1, num_samples * num_cols
                                    ).reshape(num_samples, num_cols)
    data = np.array(data)

if self.sample_around_instance:
    data = data * scale + instance_sample
else:
    data = data * scale + mean
```

Generation step (tabular data)

lime/lime/lime_tabular.py

- LIME **samples each feature independently** from a normal distribution **centred at the (instance's) feature value**
- **Steps:**
    - **Draws independent standard Gaussian noise** for each feature



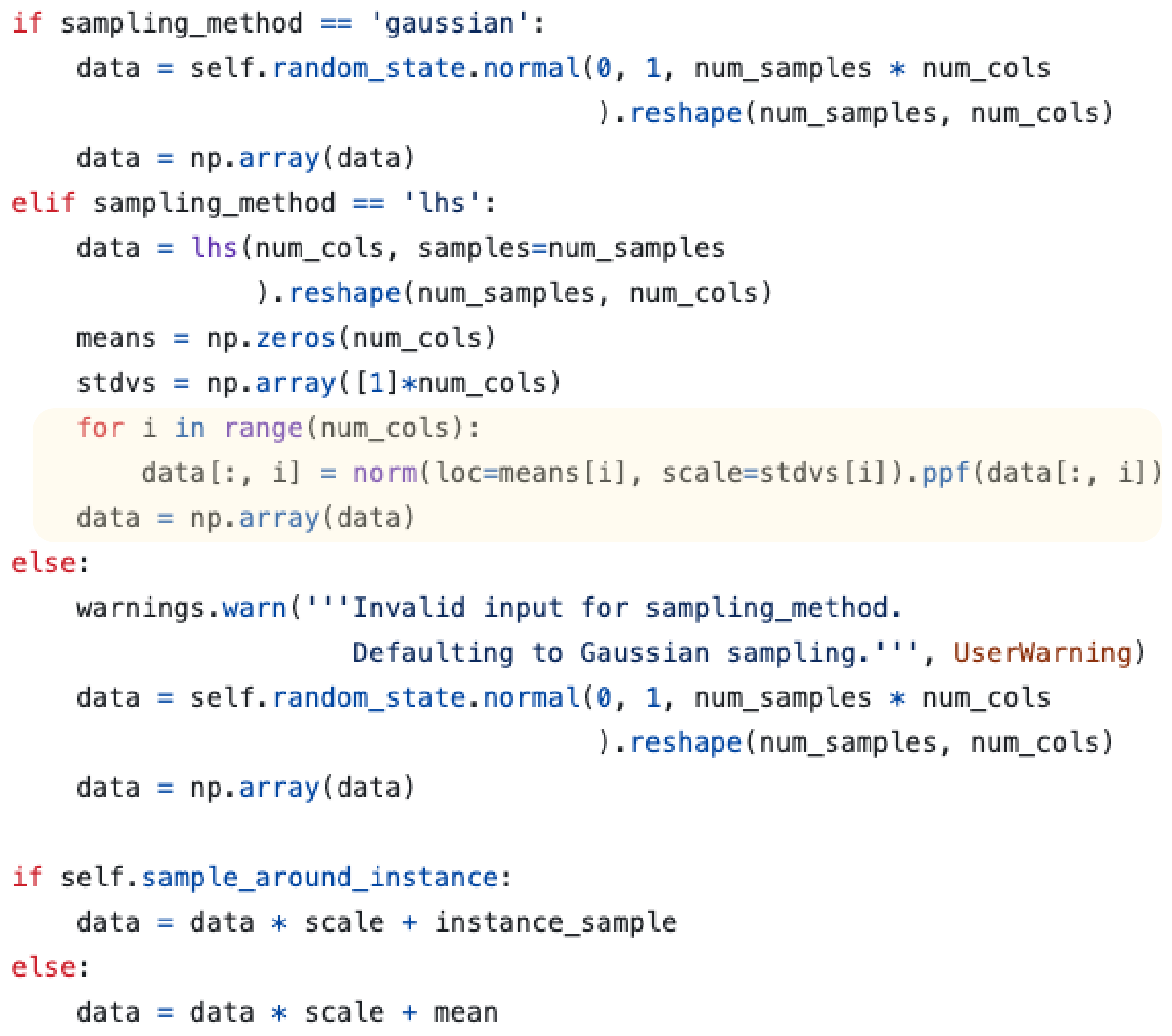
```
if sampling_method == 'gaussian':
    data = self.random_state.normal(0, 1, num_samples * num_cols
                                    ).reshape(num_samples, num_cols)
    data = np.array(data)
elif sampling_method == 'lhs':
    data = lhs(num_cols, samples=num_samples
               ).reshape(num_samples, num_cols)
    means = np.zeros(num_cols)
    stdvs = np.array([1]*num_cols)
    for i in range(num_cols):
        data[:, i] = norm(loc=means[i], scale=stdvs[i]).ppf(data[:, i])
    data = np.array(data)
else:
    warnings.warn('''Invalid input for sampling_method.
                     Defaulting to Gaussian sampling.''', UserWarning)
    data = self.random_state.normal(0, 1, num_samples * num_cols
                                    ).reshape(num_samples, num_cols)
    data = np.array(data)

if self.sample_around_instance:
    data = data * scale + instance_sample
else:
    data = data * scale + mean
```

DIGITAL

Generation step (tabular data)

[lime/lime/lime_tabular.py]

- LIME **samples each feature independently** from a normal distribution **centred at the (instance's) feature value**
- **Steps:**
  - **Draws independent standard Gaussian noise** for each feature
  - The sampled values are multiplied by the **standard deviation** of each feature

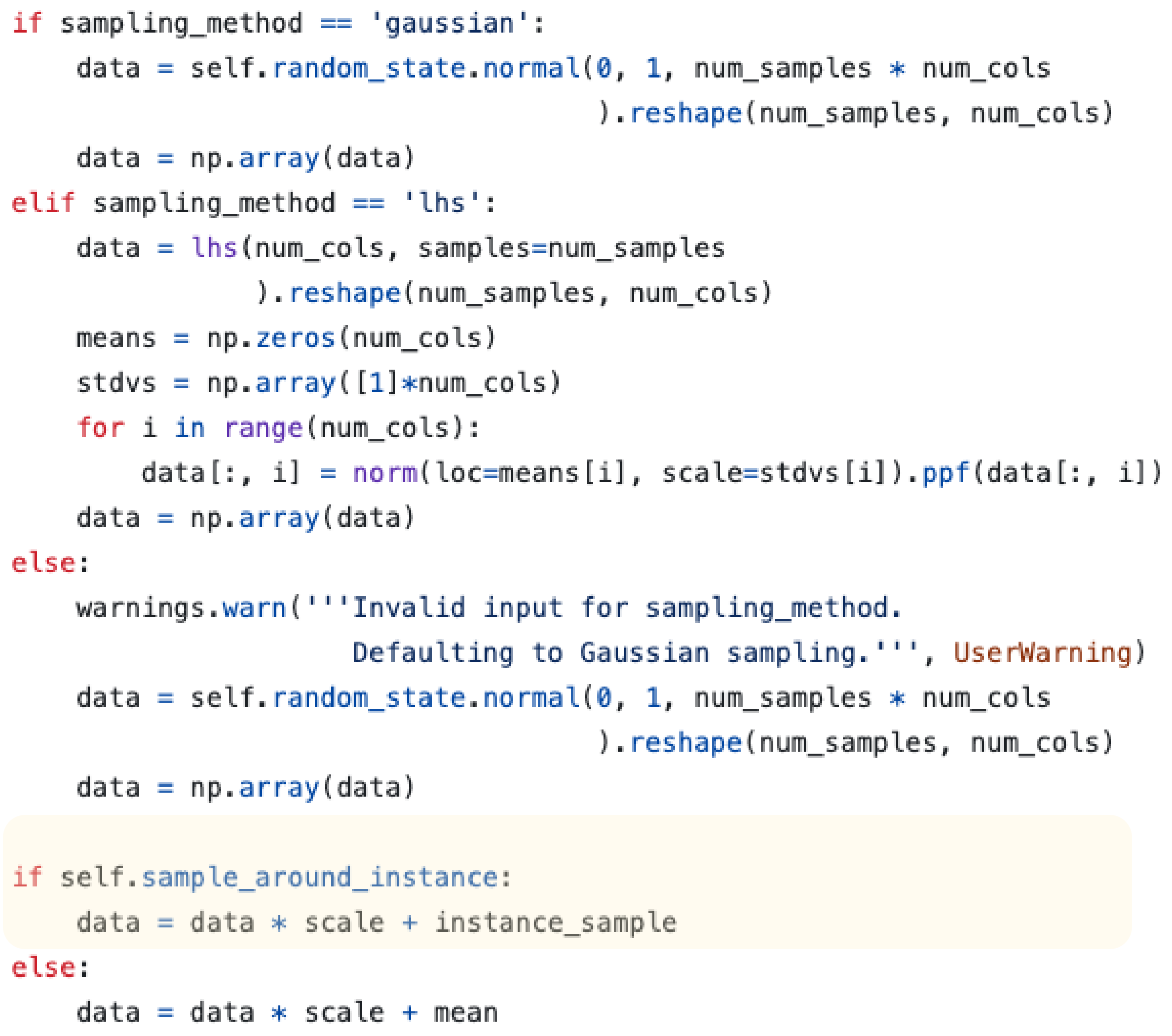


```
if sampling_method == 'gaussian':
    data = self.random_state.normal(0, 1, num_samples * num_cols
                                    ).reshape(num_samples, num_cols)
    data = np.array(data)
elif sampling_method == 'lhs':
    data = lhs(num_cols, samples=num_samples
               ).reshape(num_samples, num_cols)
    means = np.zeros(num_cols)
    stdvs = np.array([1]*num_cols)
    for i in range(num_cols):
        data[:, i] = norm(loc=means[i], scale=stdvs[i]).ppf(data[:, i])
    data = np.array(data)
else:
    warnings.warn('''Invalid input for sampling_method.
                     Defaulting to Gaussian sampling.''', UserWarning)
    data = self.random_state.normal(0, 1, num_samples * num_cols
                                    ).reshape(num_samples, num_cols)
    data = np.array(data)

if self.sample_around_instance:
    data = data * scale + instance_sample
else:
    data = data * scale + mean
```

Generation step (tabular data)

lime/lime/lime_tabular.py

- LIME **samples each feature independently** from a normal distribution **centred at the (instance's) feature value**
- **Steps:**
    - **Draws independent standard Gaussian noise** for each feature
    - The sampled values are multiplied by the **standard deviation** of each feature
    - Then the noise is **added to** either:
        - the **instance value** (instance_sample) → if sample_around_instance=True

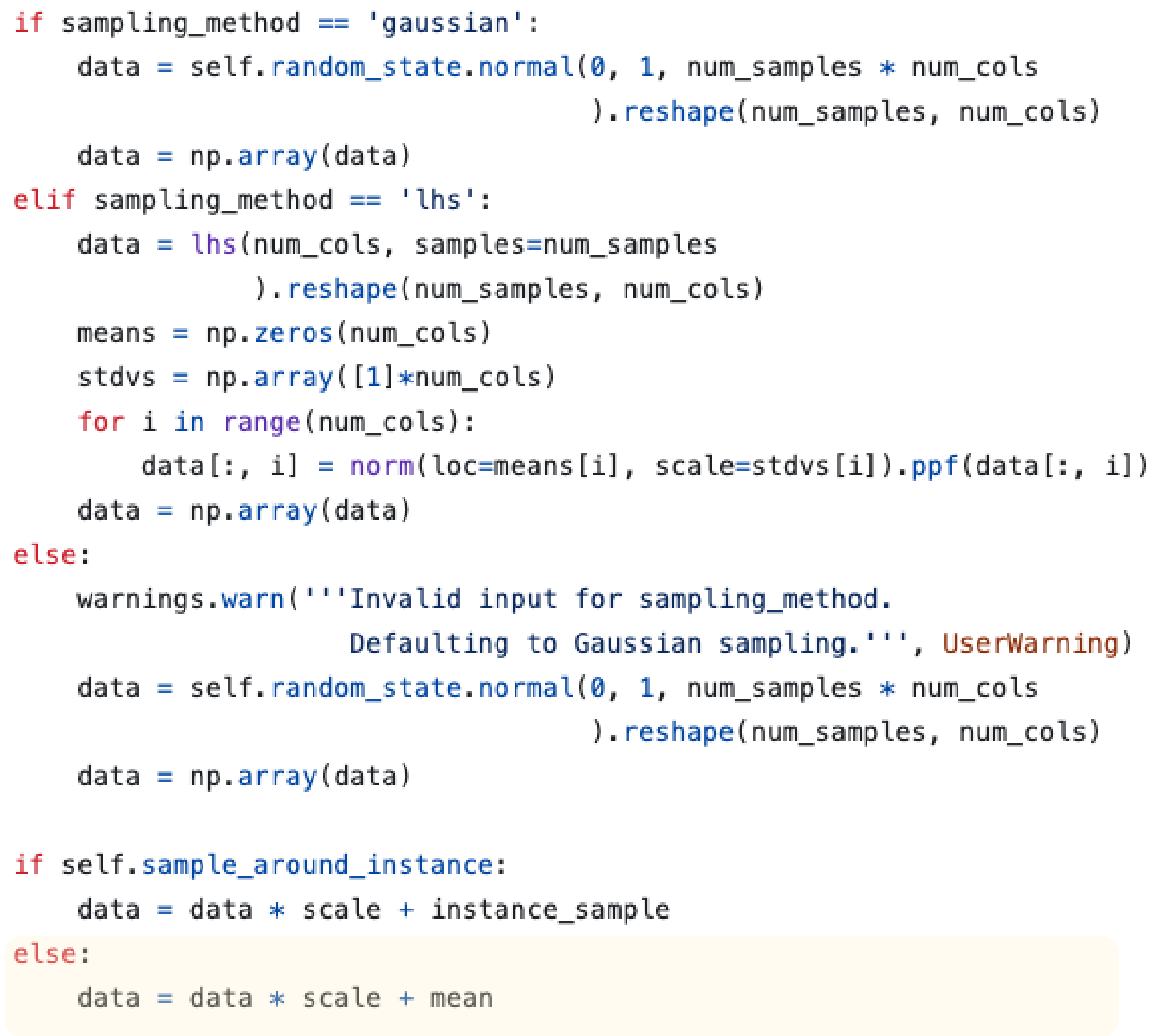


```python
if sampling_method == 'gaussian':
    data = self.random_state.normal(0, 1, num_samples * num_cols
                                    ).reshape(num_samples, num_cols)
    data = np.array(data)
elif sampling_method == 'lhs':
    data = lhs(num_cols, samples=num_samples
               ).reshape(num_samples, num_cols)
    means = np.zeros(num_cols)
    stdvs = np.array([1]*num_cols)
    for i in range(num_cols):
        data[:, i] = norm(loc=means[i], scale=stdvs[i]).ppf(data[:, i])
    data = np.array(data)
else:
    warnings.warn('''Invalid input for sampling_method.
                     Defaulting to Gaussian sampling.''', UserWarning)
    data = self.random_state.normal(0, 1, num_samples * num_cols
                                    ).reshape(num_samples, num_cols)
    data = np.array(data)

if self.sample_around_instance:
    data = data * scale + instance_sample
else:
    data = data * scale + mean
```

Generation step (tabular data)

lime/lime/lime_tabular.py

- LIME **samples each feature independently** from a normal distribution **centred at the (instance's) feature value**
- **Steps:**
  - **Draws independent standard Gaussian noise** for each feature
  - The sampled values are multiplied by the **standard deviation** of each feature
  - Then the noise is **added to** either:
    - the **instance value** (instance_sample) → if sample_around_instance=True
    - the **feature mean** (mean) → if sample_around_instance=False

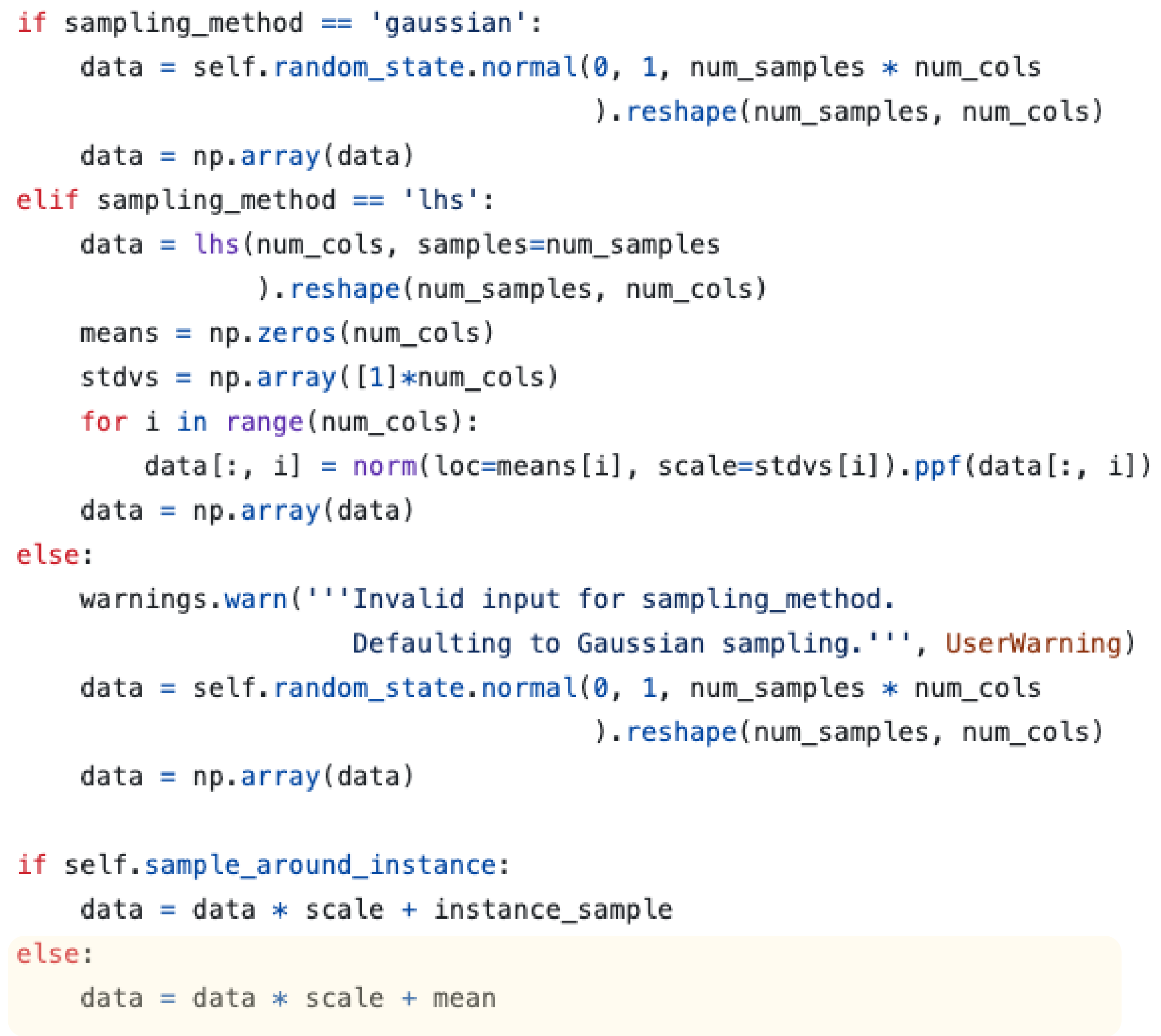


```python
if sampling_method == 'gaussian':
    data = self.random_state.normal(0, 1, num_samples * num_cols
                                    ).reshape(num_samples, num_cols)
    data = np.array(data)
elif sampling_method == 'lhs':
    data = lhs(num_cols, samples=num_samples
               ).reshape(num_samples, num_cols)
    means = np.zeros(num_cols)
    stdvs = np.array([1]*num_cols)
    for i in range(num_cols):
        data[:, i] = norm(loc=means[i], scale=stdvs[i]).ppf(data[:, i])
    data = np.array(data)
else:
    warnings.warn('''Invalid input for sampling_method.
                     Defaulting to Gaussian sampling.''', UserWarning)
    data = self.random_state.normal(0, 1, num_samples * num_cols
                                    ).reshape(num_samples, num_cols)
    data = np.array(data)

if self.sample_around_instance:
    data = data * scale + instance_sample
else:
    data = data * scale + mean
```

Generation step (tabular data)

lime/lime/lime_tabular.py

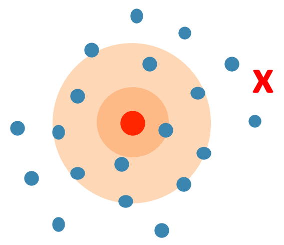
- LIME **samples each feature independently** from a normal distribution **centred at the (instance's) feature value**
- **Steps:**
  - **Draws independent standard Gaussian noise** for each feature
  - The sampled values are multiplied by the **standard deviation** of each feature
  - Then the noise is **added to** either:
    - the **instance value** (instance_sample) → if **sample_around_instance=True**
    - the **feature mean** (mean) → if **sample_around_instance=False**



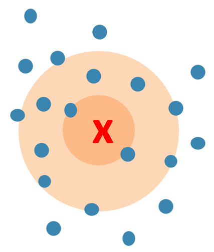
```
if sampling_method == 'gaussian':
    data = self.random_state.normal(0, 1, num_samples * num_cols
                                    ).reshape(num_samples, num_cols)
    data = np.array(data)
elif sampling_method == 'lhs':
    data = lhs(num_cols, samples=num_samples
               ).reshape(num_samples, num_cols)
    means = np.zeros(num_cols)
    stdvs = np.array([1]*num_cols)
    for i in range(num_cols):
        data[:, i] = norm(loc=means[i], scale=stdvs[i]).ppf(data[:, i])
    data = np.array(data)
else:
    warnings.warn('''Invalid input for sampling_method.
                     Defaulting to Gaussian sampling.''', UserWarning)
    data = self.random_state.normal(0, 1, num_samples * num_cols
                                    ).reshape(num_samples, num_cols)
    data = np.array(data)

if self.sample_around_instance:
    data = data * scale + instance_sample
else:
    data = data * scale + mean
```

Why the options?

sample_around_instance=**False**



sample_around_instance=**True**



Simpler coverage - samples explore the overall data distribution.

More stable if the instance is an outlier. Avoids extrapolation into sparse or unseen areas.

Not truly local - perturbations may be far from the instance.

Violates the initiation of LIME

Perturbations cluster near the instance - **more faithful local surrogate.**

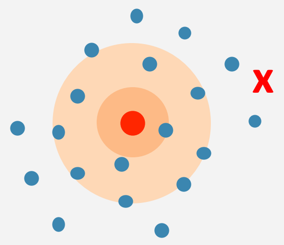
Aligns with LIME's core idea

If the instance is near the edge of the data distribution, **sampled points may fall in low-density or unrealistic regions**

What is better?

Why the **options**?

sample_around_instance=**False**



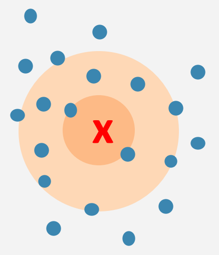
Simpler coverage - samples explore the overall data distribution.

More stable if the instance is an outlier. Avoids extrapolation into sparse or unseen areas.

Not truly local - perturbations may be far from the instance.

Violates the initiation of LIME

sample_around_instance=**True**



Perturbations cluster near the instance - **more faithful local surrogate.**

Aligns with LIME's core idea

data

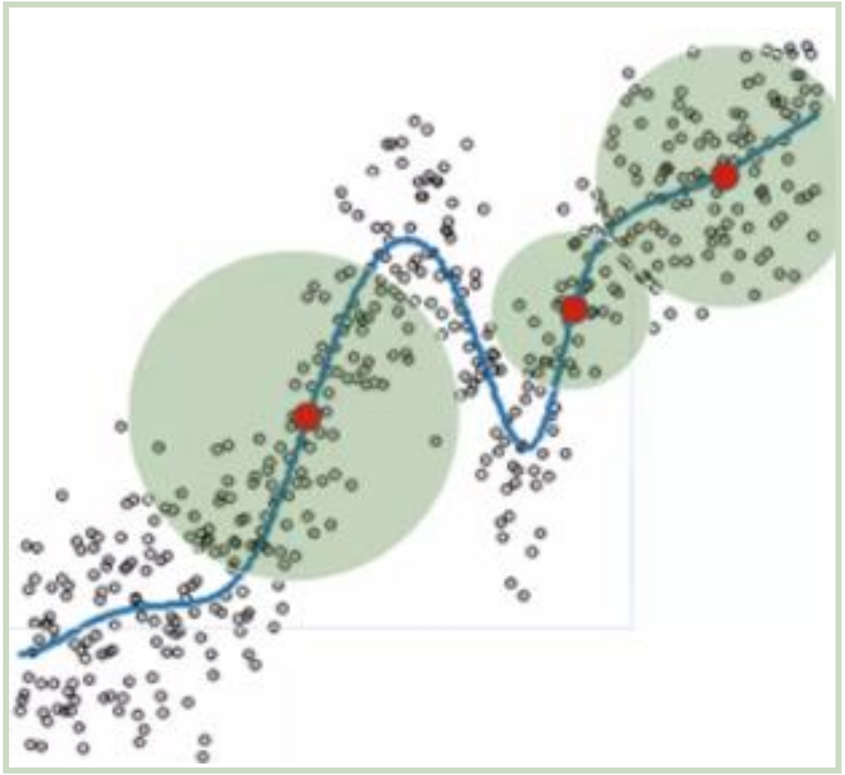
low-

density or unrealistic regions

The "right" sampling radius depends on the local shape of the model.

What is better?

(some) Guidelines



Ideally, sampled points should lie in a **meaningful neighbourhood** around the instance.

But how big should that neighbourhood be? That's tricky.

- Proper size ... depends on the reference point

The best neighbourhood **is not fixed but** it **depends on how curvy the model is nearby**.

Near flat regions → you can sample wider.

- i.e. if the function around the point is flat, a **larger neighbourhood** can still be well-approximated linearly.

Near sharp bends → you must stay narrow to preserve locality.

- i.e., if the function around the point has high curvature (nonlinear), a **small neighbourhood** is needed to get a linear fit.

Weighting step

$$\xi(x) = argmin_{g \in G} L(f, g, \pi_x) + \Omega(g) \qquad (1)$$

$$L(f, g, \pi_x) = \sum_i \boldsymbol{\pi_x(x_i)} \big(f(x_i) - g(x_i)\big)^2 \qquad (2)$$

$$\boldsymbol{\pi_x(x_i)} = \exp\left(\frac{-D(x, x_i)^2}{\sigma^2}\right) \qquad (3)$$

- Controls which points are considered more relevant for the explanation.
- $D(x, x_i)^2$ is the **Euclidean distance** between the perturbed point x and the original instance.
- σ controls the **scale of locality** (how fast weights decrease as distance increases).

Weighting step

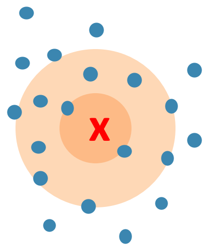
$$\xi(x) = argmin_{g \in G} L(f, g, \pi_x) + \Omega(g) \qquad (1)$$

$$L(f, g, \pi_x) = \sum_i \pi_x(x_i)\big(f(x_i) - g(x_i)\big)^2 \qquad (2)$$

$$\pi_x(x_i) = \exp\left(\frac{-D(x, x_i)^2}{\sigma^2}\right) \qquad (3)$$

The proximity parameter attributes a value in the range [0, 1], the higher the closer to the reference point.

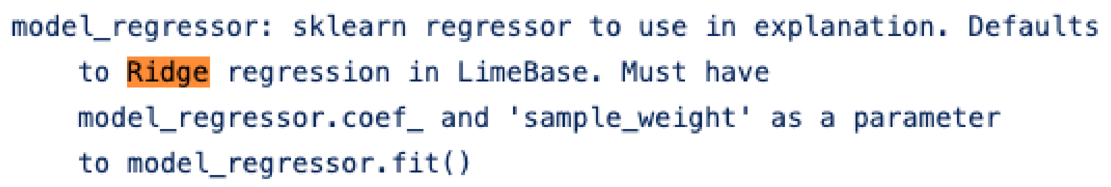
The kernel width σ parameter decides how large is the circle of the meaningful weights around the red dot.



Local model step

- As the last step, LIME uses a **surrogate model to approximate the ML model in the small region** around our reference red dot, determined by the weights.
- We may choose **any kind of explainable model** for the approximation (Decision Trees, Logistic Regression, GLM, GAM, etc.)
- The **default surrogate model** in LIME's Python implementation is **Ridge Regression**

lime/lime/lime_tabular.py



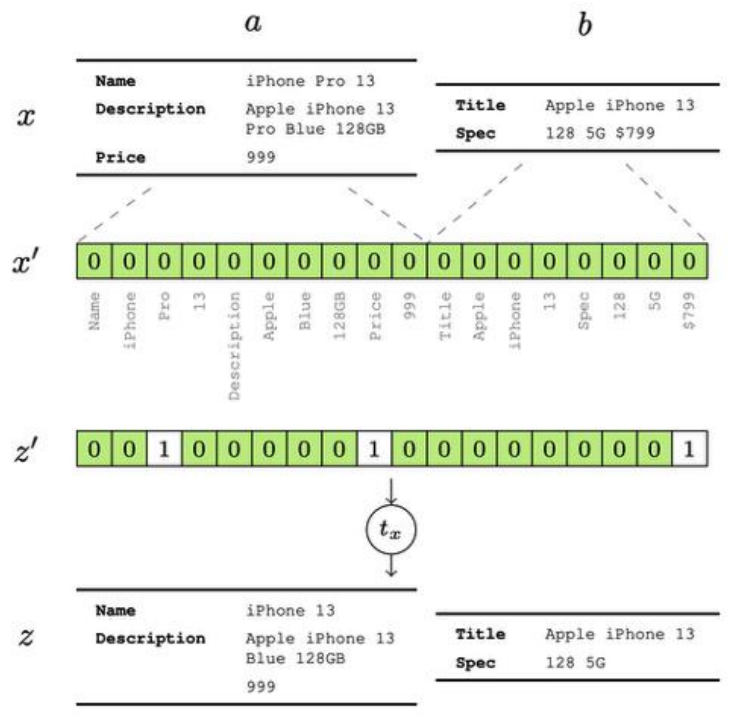
```
model_regressor: sklearn regressor to use in explanation. Defaults
    to Ridge regression in LimeBase. Must have
    model_regressor.coef_ and 'sample_weight' as a parameter
    to model_regressor.fit()
```

Ridge Regression is a type of **linear regression** that adds **L2 regularization** to prevent overfitting by penalizing large coefficients.

DIGITAL

Detour: What about text?

- Text perturbation in LIME - **LIME perturbs text by randomly removing words** from the original instance.



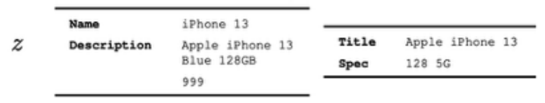
https://arxiv.org/abs/2110.00516

Tokenize words and generate binary vectors

Create perturbed samples (by masking certain words)

Detour: What about text?

- Text perturbation in LIME - **LIME perturbs text by randomly removing words** from the original instance.

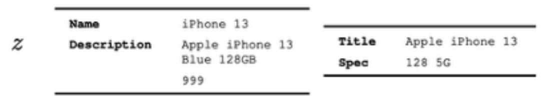


z

Name	iPhone 13
Description	Apple iPhone 13 Blue 128GB
	999

Title	Apple iPhone 13
Spec	128 5G

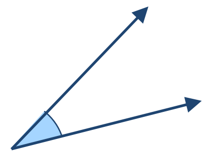
Get BB predictions for the perturbed samples



z

Name	iPhone 13
Description	Apple iPhone 13 Blue 128GB
	999

Title	Apple iPhone 13
Spec	128 5G



Calculate cosine similarity

Train the surrogate model → **a weighted linear regression is trained** on the binary vectors and their corresponding predictions.

DeepLIFT

- DeepLIFT (**Deep L**earning **I**mportant **F**ea**T**ures), a method for decomposing the output prediction of a neural network on a specific input by backpropagating the contributions of all neurons in the network to every feature of the input.

- Other methods for explaining the output of NNs often rely of **gradients.**
  - **Q:** *How sensitive is the output to a small change in input?*

Saliency	$\partial Output / \partial Input$
Integrated gradients	Gradients + Integrated path
…	….

More on this later!

- Instead of looking only at the gradients, **DeepLIFT** compares the activation of each neuron for a given input to the activation for a reference input (like a baseline).

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

DeepLIFT

- Let’s say you care about a specific output of the model, like a classification score. Call that output $\boldsymbol{t}$.
- Next, we define $\boldsymbol{\Delta t} = \boldsymbol{t} - \boldsymbol{t}_0$, where $\boldsymbol{t}_0$ is the model's output for the reference input.
- DeepLIFT assigns scores to each input feature (or neuron in an intermediate layer) to explain this Δt.
- It ensures that:

$$\sum_{i=1}^{n} C_{\Delta x_i \Delta \mathrm{t}} = \Delta t$$

- This is called the **summation-to-delta** property. It means: *all the contribution scores add up exactly to the output difference from reference*.

DeepLIFT

Gradients can be **zero** even when a feature matters (due to ReLU or saturation). DeepLIFT doesn't suffer from this; it gives non-zero contributions by using **differences instead of derivatives**.

STEP 1

Pick a reference input (e.g., an all-zeros image or the mean input). This is chosen by the user.

STEP 2

Compute the output difference between your actual input and this reference.

STEP 3

Backpropagate the contribution of that output difference through the network, assigning *blame* to each neuron along the way.

STEP 4

Each input feature gets **a score** telling you how much it *contributed* to the difference from the reference.

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

DeepLIFT

STEP 1

Pick a reference input (e.g., an all-zeros image or the mean input). This is chosen by the user.

STEP 2

Compute the output difference between your actual input and this reference.

STEP 3

Backpropagate the contribution of that output difference through the network, assigning *blame* to each neuron along the way.

STEP 4

Each input feature gets **a score** telling you how much it *contributed* to the difference from the reference.

Multipliers & the Chain Rule

- Let's say you're tracking how **a change in input x** led to a **change in output t**.
- We define a multiplier:

$$m_{\Delta x \to \Delta t} = \frac{C_{\Delta x \to \Delta t}}{\Delta x}$$

Where:
$\Delta x = x - x^{ref}$
$\Delta t = t - t^{ref}$
$C_{\Delta x \to \Delta t}$ is the contribution of Δx to Δt

Think of it like a finite difference version of a partial derivative:
Partial derivative: $\frac{\partial t}{\partial x}$ (infinitesimal change)
Multiplier: $\frac{\Delta t}{\Delta x}$ (finite change)

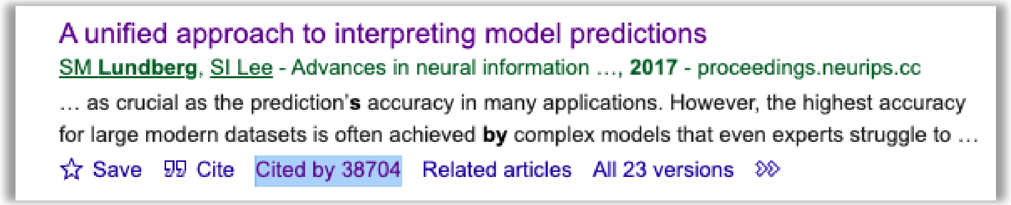
- Next, we want to trace how input x_i affects output t, **through** hidden neurons y_i. We apply a chair rule:

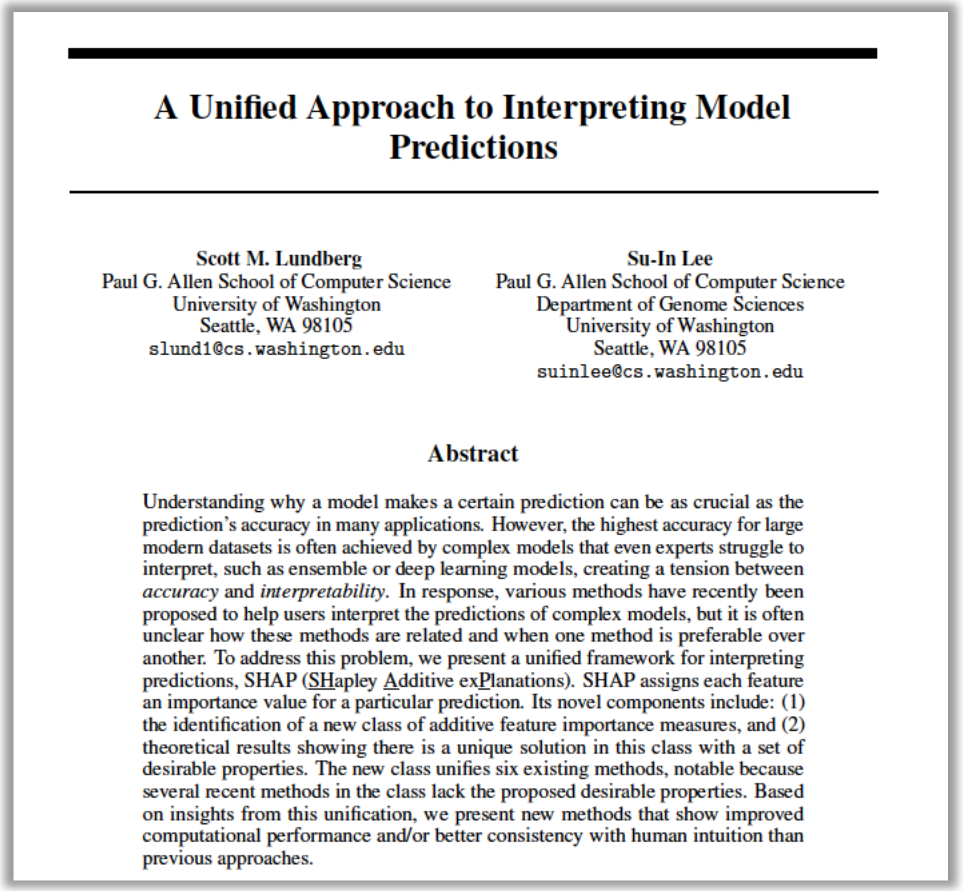
$$m_{\Delta x \to \Delta t} = \sum_j m_{x_i \to y_i} * m_{y_i \to t}$$

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

- This brings us to the **first contribution** in the paper published by Lundberg and Lee (2017)


A unified approach to interpreting model predictions
SM Lundberg, SI Lee - Advances in neural information …, 2017 - proceedings.neurips.cc
… as crucial as the prediction's accuracy in many applications. However, the highest accuracy for large modern datasets is often achieved **by** complex models that even experts struggle to …
Save Cite Cited by 38704 Related articles All 23 versions


A Unified Approach to Interpreting Model Predictions

Scott M. Lundberg
Paul G. Allen School of Computer Science
University of Washington
Seattle, WA 98105
slund1@cs.washington.edu

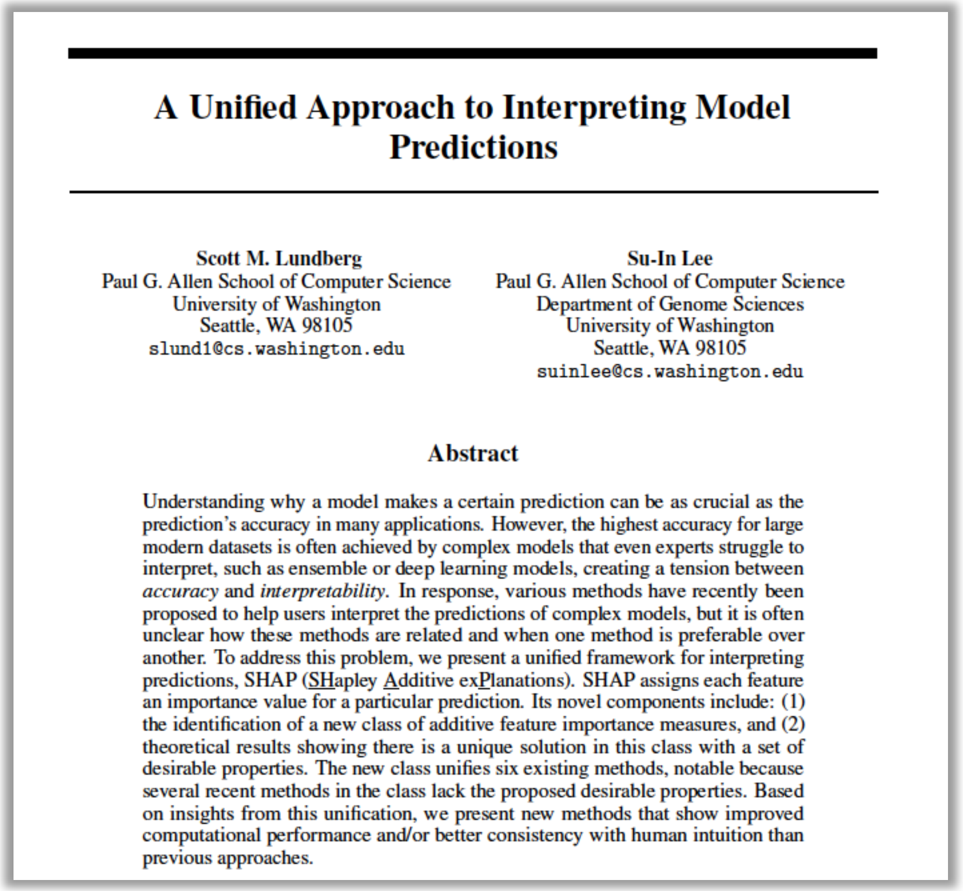
Su-In Lee
Paul G. Allen School of Computer Science
Department of Genome Sciences
University of Washington
Seattle, WA 98105
suinlee@cs.washington.edu

Abstract

Understanding why a model makes a certain prediction can be as crucial as the prediction's accuracy in many applications. However, the highest accuracy for large modern datasets is often achieved by complex models that even experts struggle to interpret, such as ensemble or deep learning models, creating a tension between *accuracy* and *interpretability*. In response, various methods have recently been proposed to help users interpret the predictions of complex models, but it is often unclear how these methods are related and when one method is preferable over another. To address this problem, we present a unified framework for interpreting predictions, SHAP (SHapley Additive exPlanations). SHAP assigns each feature an importance value for a particular prediction. Its novel components include: (1) the identification of a new class of additive feature importance measures, and (2) theoretical results showing there is a unique solution in this class with a set of desirable properties. The new class unifies six existing methods, notable because several recent methods in the class lack the proposed desirable properties. Based on insights from this unification, we present new methods that show improved computational performance and/or better consistency with human intuition than previous approaches.

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences



A Unified Approach to Interpreting Model Predictions

Scott M. Lundberg
Paul G. Allen School of Computer Science
University of Washington
Seattle, WA 98105
slund1@cs.washington.edu

Su-In Lee
Paul G. Allen School of Computer Science
Department of Genome Sciences
University of Washington
Seattle, WA 98105
suinlee@cs.washington.edu

Abstract

Understanding why a model makes a certain prediction can be as crucial as the prediction's accuracy in many applications. However, the highest accuracy for large modern datasets is often achieved by complex models that even experts struggle to interpret, such as ensemble or deep learning models, creating a tension between *accuracy* and *interpretability*. In response, various methods have recently been proposed to help users interpret the predictions of complex models, but it is often unclear how these methods are related and when one method is preferable over another. To address this problem, we present a unified framework for interpreting predictions, SHAP (SHapley Additive exPlanations). SHAP assigns each feature an importance value for a particular prediction. Its novel components include: (1) the identification of a new class of additive feature importance measures, and (2) theoretical results showing there is a unique solution in this class with a set of desirable properties. The new class unifies six existing methods, notable because several recent methods in the class lack the proposed desirable properties. Based on insights from this unification, we present new methods that show improved computational performance and/or better consistency with human intuition than previous approaches.

Contribution 1

The authors propose that many explanation methods can be described by a **common form i.e.** an explanation model that is a linear function of binary variables:

$$f(x) \approx g(z') = \phi_o + \sum_{i=1}^{M} \phi_i Z_i'$$

They show that **all well-known explanation methods** are all **additive** in this way.

Additive Feature Attribution Methods

$$\boxed{f(x)} \approx g(z') = \phi_o + \sum_{i=1}^{M} \phi_i Z'_i$$

The underlining complex model

Additive Feature Attribution Methods

$$f(x) \approx \boxed{g(z')} = \phi_o + \sum_{i=1}^{M} \phi_i Z_i'$$

Local explanation model

An interpretable model that approximates the behaviour of a complex model

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Additive Feature Attribution Methods

$$f(x) \approx g(z') = \phi_o + \sum_{i=1}^{M} \phi_i Z'_i$$

$$x = h_x(z')$$

Perturbation function h_x that maps simplified input z' to the original input space x

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Additive Feature Attribution Methods

$$f(x) \approx g(z') = \boxed{\phi_o} + \sum_{i=1}^{M} \phi_i Z'_i$$

Base value, i.e., the model's expected output when no features are present

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

Additive Feature Attribution Methods

$$f(x) \approx g(z') = \phi_o + \sum_{i=1}^{\boxed{M}} \phi_i Z'_i$$

Number of features

Additive Feature Attribution Methods

$$f(x) \approx g(z') = \phi_o + \sum_{i=1}^{M} \boxed{\phi_i} Z'_i$$

Attribution for feature i

Additive Feature Attribution Methods

$$f(x) \approx g(z') = \phi_o + \sum_{i=1}^{M} \phi_i \boxed{Z'_i}$$

Coalition $Z'_i \in \{0,1\}^M$

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

United in the additive nature …

- In *LIME*, we have:

$$\xi(x) = argmin_{g \in G} L(f, g, \pi_x) + \Omega(g)$$

- and g *follows:*

$$g(z') = \phi_o + \sum_{i=1}^{M} \phi_i Z'_i$$

Additive feature attribution method ✓

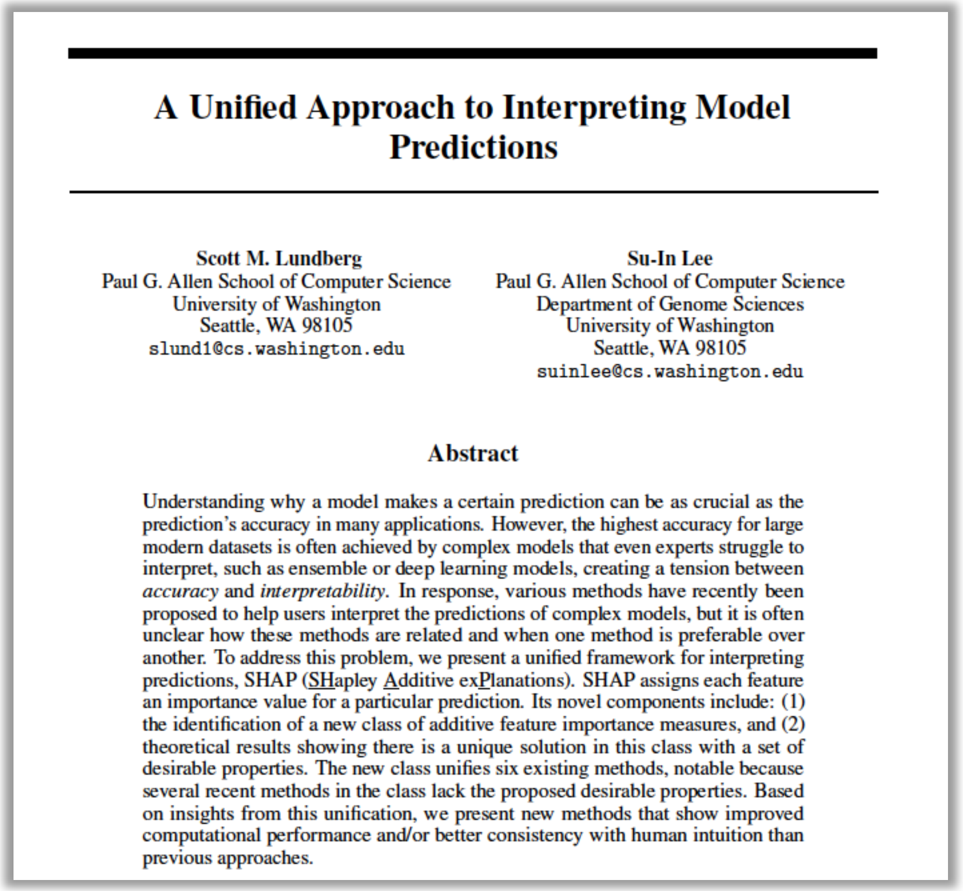
- *DeepLIFT uses a "summation-to-delta" property that states:*

$$\sum_{i=1}^{n} C_{\Delta x_i \Delta t} = \Delta t$$

Additive feature attribution method ✓

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences


A Unified Approach to Interpreting Model Predictions

Scott M. Lundberg
Paul G. Allen School of Computer Science
University of Washington
Seattle, WA 98105
slund1@cs.washington.edu

Su-In Lee
Paul G. Allen School of Computer Science
Department of Genome Sciences
University of Washington
Seattle, WA 98105
suinlee@cs.washington.edu

Abstract

Understanding why a model makes a certain prediction can be as crucial as the prediction's accuracy in many applications. However, the highest accuracy for large modern datasets is often achieved by complex models that even experts struggle to interpret, such as ensemble or deep learning models, creating a tension between *accuracy* and *interpretability*. In response, various methods have recently been proposed to help users interpret the predictions of complex models, but it is often unclear how these methods are related and when one method is preferable over another. To address this problem, we present a unified framework for interpreting predictions, SHAP (SHapley Additive exPlanations). SHAP assigns each feature an importance value for a particular prediction. Its novel components include: (1) the identification of a new class of additive feature importance measures, and (2) theoretical results showing there is a unique solution in this class with a set of desirable properties. The new class unifies six existing methods, notable because several recent methods in the class lack the proposed desirable properties. Based on insights from this unification, we present new methods that show improved computational performance and/or better consistency with human intuition than previous approaches.

Contribution 2

A surprising attribute of the class of additive feature attribution methods is the presence of a single unique solution in this class with desirable properties **(!)**

Efficiency	The **sum of attributions = model output**
Consistency	Consistency says if a feature's **contribution increases** in a newer model, its attribution should not decrease
Null Player	If a feature is **not present** in any coalition, it should get **0 contribution**

Only **Shapley values satisfy all properties**. Methods not based on Shapley values violate local accuracy and/or consistency

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

LIME & **the Properties**

It prevents misleading explanations and guarantees that no part of the prediction is left "unexplained."

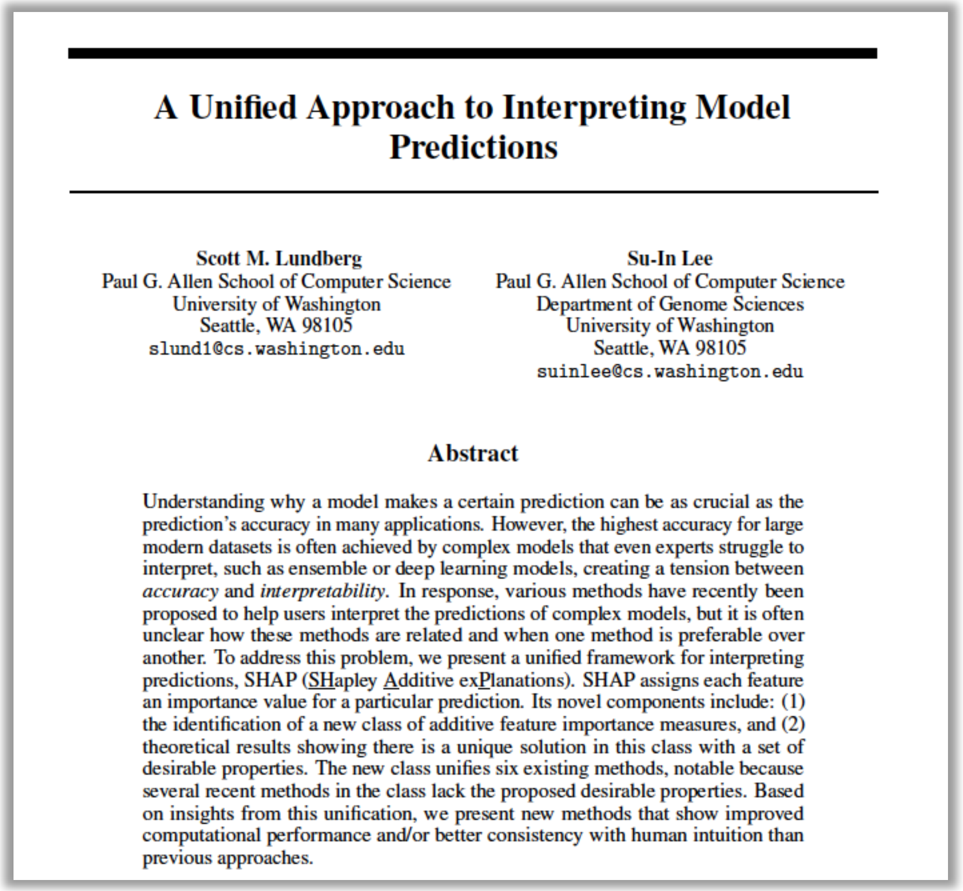
Efficiency

The **sum of attributions = model output**

- Remember: LIME uses a **local surrogate model** (linear regression).
- The weights assigned to perturbed samples are calculated using a distance-based kernel, not a theoretically grounded one like SHAP.
- Because of these heuristic weights, the surrogate model is not required to exactly match the model's output at the original input x, that is $g(1,1,\ldots 1) \neq f(x)$
- As a result, the sum of the feature attributions from the surrogate model may not equal the model's prediction.

DIGITAL

Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences



A Unified Approach to Interpreting Model Predictions

Scott M. Lundberg
Paul G. Allen School of Computer Science
University of Washington
Seattle, WA 98105
slund1@cs.washington.edu

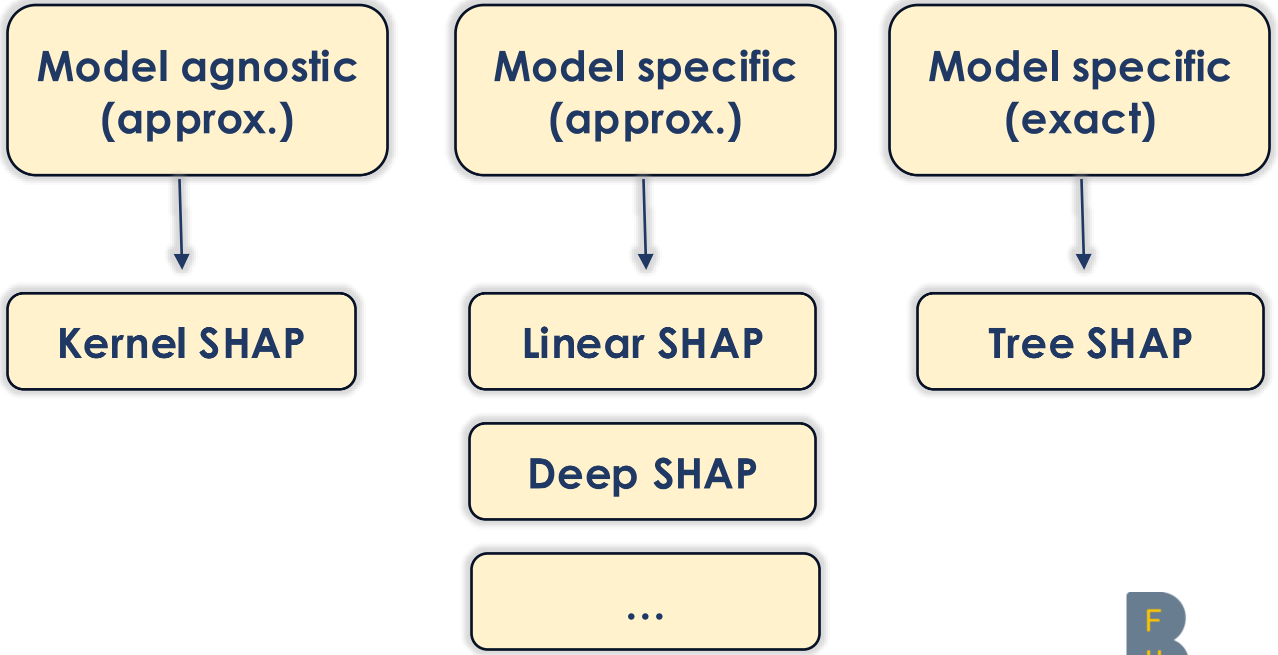
Su-In Lee
Paul G. Allen School of Computer Science
Department of Genome Sciences
University of Washington
Seattle, WA 98105
suinlee@cs.washington.edu

Abstract

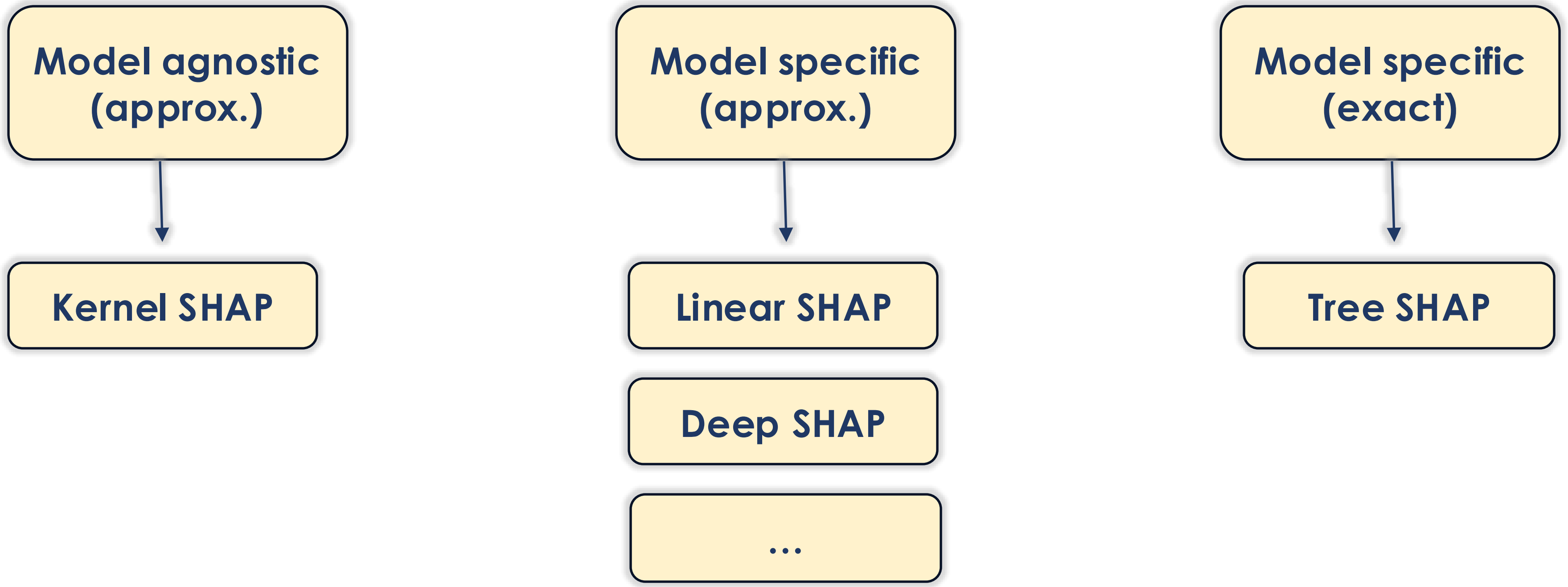
Understanding why a model makes a certain prediction can be as crucial as the prediction's accuracy in many applications. However, the highest accuracy for large modern datasets is often achieved by complex models that even experts struggle to interpret, such as ensemble or deep learning models, creating a tension between *accuracy* and *interpretability*. In response, various methods have recently been proposed to help users interpret the predictions of complex models, but it is often unclear how these methods are related and when one method is preferable over another. To address this problem, we present a unified framework for interpreting predictions, SHAP (SHapley Additive exPlanations). SHAP assigns each feature an importance value for a particular prediction. Its novel components include: (1) the identification of a new class of additive feature importance measures, and (2) theoretical results showing there is a unique solution in this class with a set of desirable properties. The new class unifies six existing methods, notable because several recent methods in the class lack the proposed desirable properties. Based on insights from this unification, we present new methods that show improved computational performance and/or better consistency with human intuition than previous approaches.

Contribution 3

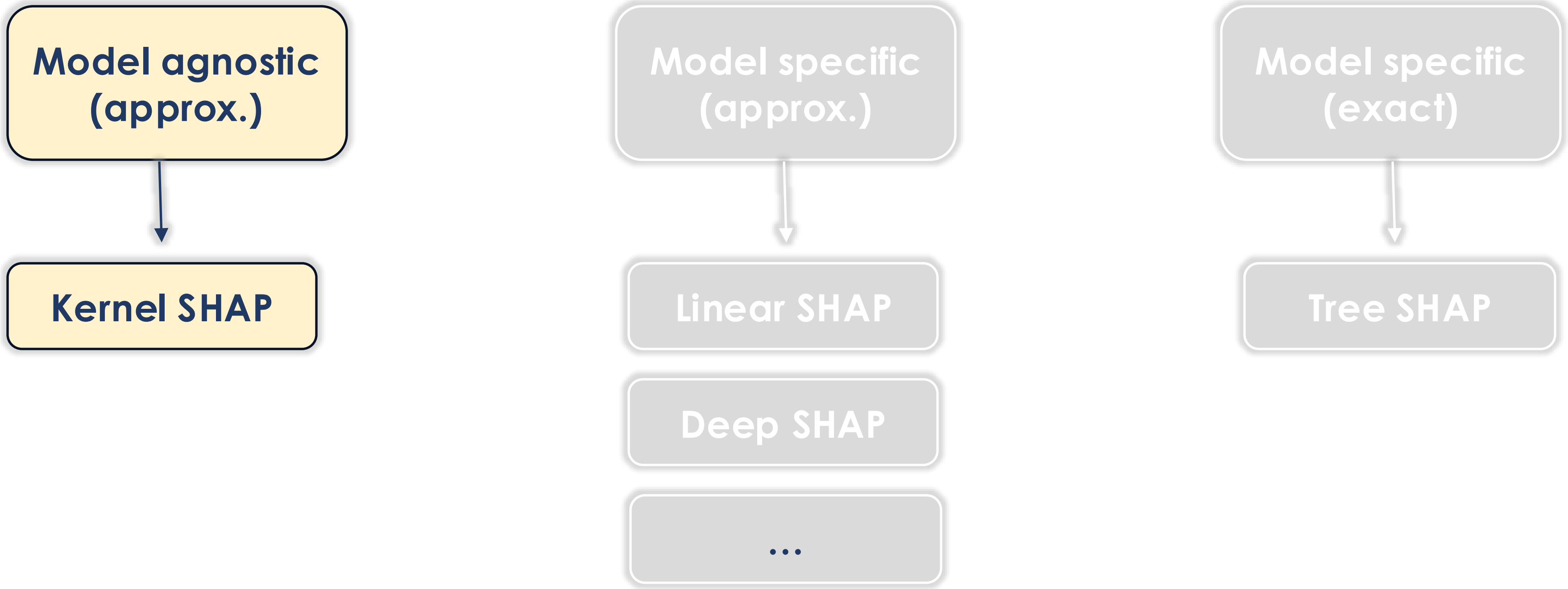
The authors also propose practical SHAP algorithms that can approximate Shapley values



SHAP Implementations



SHAP **Implementations**



KernelSHAP: Steps

Steps:

01. **Sample coalitions** (random – chain of 0s and 1s)

02. **Get predictions** from the BB model for each coalition

03. **Compute the weights** for each coalition

04. **Fit a weighted linear model**

05. **Return SHAP** values (coefficients)

For example, the vector of (1,0,1,0,0,1) means features marked as 1 are taken from x and features marked as 0 are replaced.

Marginal sampling: sample missing features from their marginal distribution (i.e., from the data directly, regardless of the present features).

Conditional sampling: model the conditional distribution $\mathrm{P}(x_{missing} \mid x_{present})$ (using k-nearest neighbours, generative models, or copulas).

Fixed baseline values: replace missing features with global baseline values (mean, mode, etc.)

KernelSHAP: **Evaluate the model & fit weighted linear model**

Coalition (z’)	**z1 (x_1)**	**z2 (x_2)**	**z3 (x_1)**	**$y_{z'} = f(h_x(z'))$ = model prediction**	**Weight $\pi(z')$**
(0,0,0)	0	0	0	0.52 (baseline prediction)	∞ (force fit φ_0)
(1,0,0)	1	0	0	0.60	π(1) = …
(0,1,0)	0	1	0	0.65	π(1) = …
(0,0,1)	0	0	1	0.70	π(1) = …
(1,1,0)	1	1	0	0.80	π(2) = …
		0	1	0.85	π(2) = …
		1	1	0.90	π(2) = …
		1	1	1.00 (full model prediction)	∞ (force fit sum $\varphi_i + \varphi_0$)

The regression is on binary coalition indicators, but the **y-values in that regression come from sampling the real feature space.**

Is the **local accuracy** satisfied?

LIME

$$\pi_x(z) = \exp\left(-\frac{D(x,z)^2}{\sigma^2}\right)$$

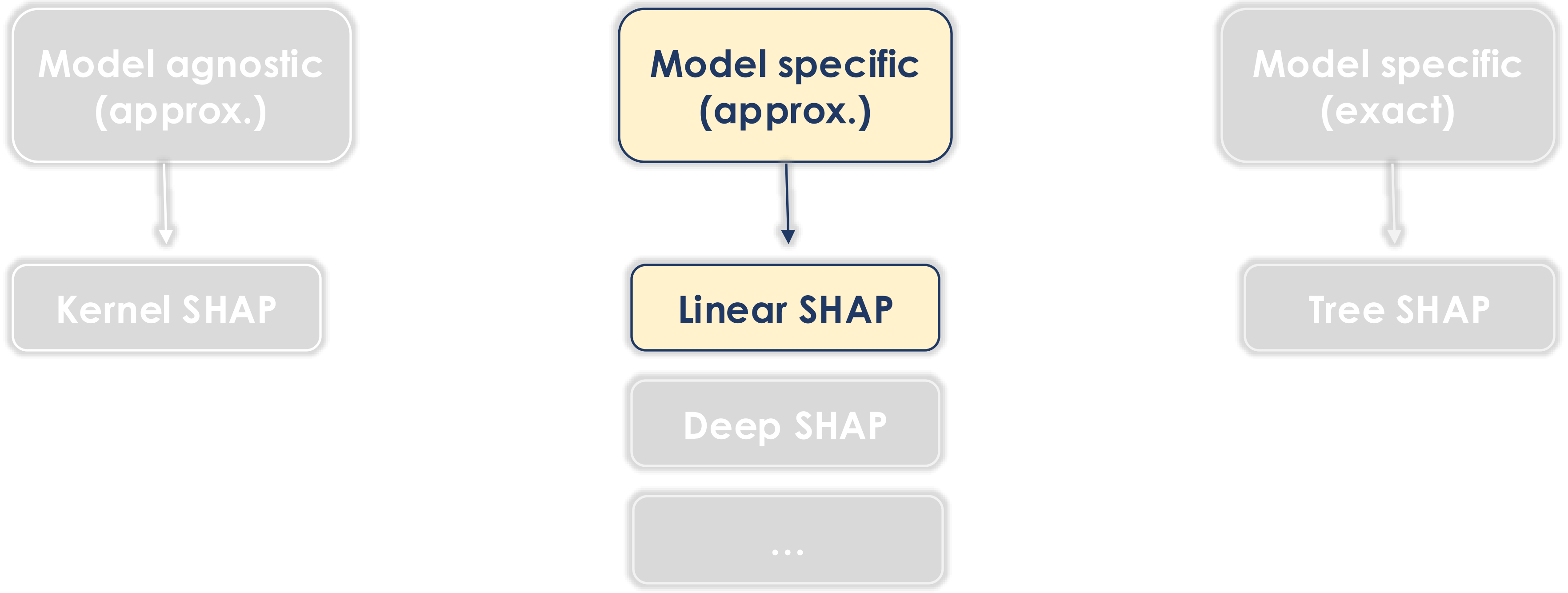
- $D(x,z)$ = distance between the original input x and the perturbed input z
- $\sigma =$ kernel width (a hyperparameter). Essentially a Gaussian kernel.

SHAP

$$\pi(z') = \frac{(M-1)}{\binom{M}{|z'|} * |z'| * (M - |z'|)}$$

- Not heuristic (!)
- Local accuracy → ensured

SHAP **Implementations**



LinearSHAP

- A **closed-form method** to compute SHAP values for **linear models.**
- There is no perturbation and no additional weighted regression.
- In ordinary regressions, we stop at the estimation of β_i but this does not tell us how much of the final prediction for **this specific instance** is due to i.
- LinearSHAP takes the coefficient β_i and scales it by how far from the instance's feature value is from the baseline expectation.

$$\phi_i = \underbrace{\beta_i}_{\text{Sensitivity}} * \underbrace{(x_i - \mathrm{E}[x_i])}_{\text{How much the feature deviates from the baseline}}$$

SHAP turns global coefficients into **instance-specific attributions**

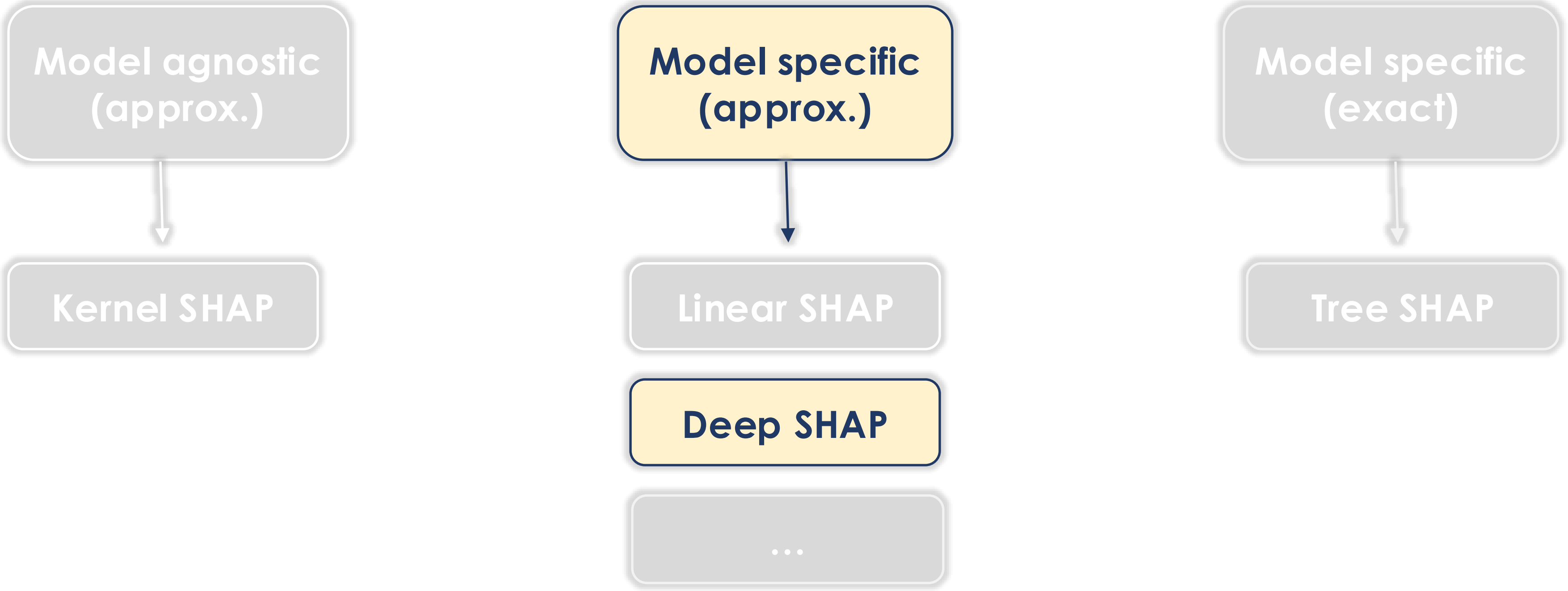
LinearSHAP

- A **closed-form method** to compute SHAP values for **linear models**
- There is no perturbation and no additional weighted regression.
- In ordinary regressions, we stop at the estimation of β_i but this does not tell us how much of the final prediction for **this specific instance** is due to i.
- LinearSHAP takes the coefficient β_i and scales it by how far from the instance's feature value is from the baseline expectation.

$$\phi_i = \underbrace{\beta_i \quad * \quad (x_i - \mathrm{E}[x_i])}_{\text{The contribution of feature } i \text{ to the prediction relative to the baseline}}$$

SHAP turns global coefficients into **instance-specific attributions**

SHAP **Implementations**



DeepSHAP

- Builds on **DeepLIFT** (as DeepLIFT already efficiently backpropagates contribution scores through a deep network using reference values)
- You pick an input x, and a **background input** (reference), x^{ref}
- It computes the **difference in model output** between your actual input and the reference:

$$\Delta f = f(x) - f(x^{ref})$$

- Then, use DeepLIFT's backpropagation rules to assign those differences to input features.

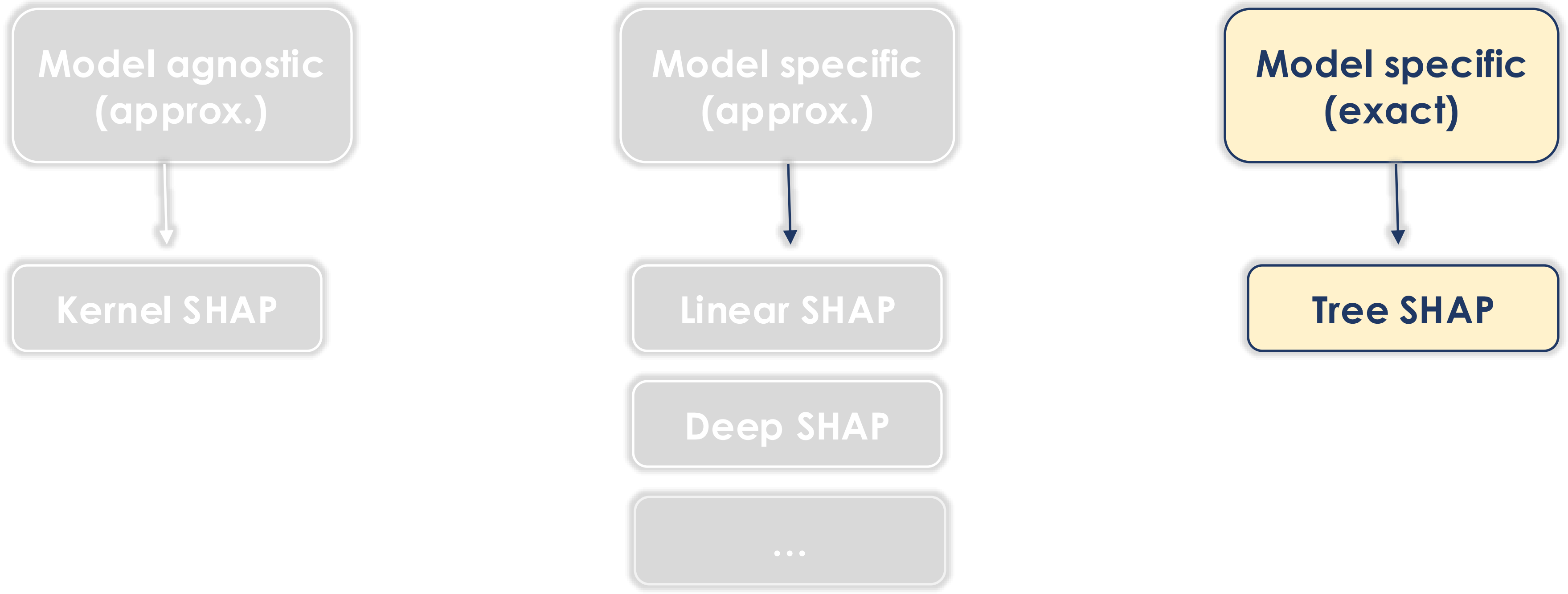
Use **DeepLIFT rules** to propagate contributions **for one coalition at a time** (efficient).

Then approximate the Shapley expectation by **sampling coalitions or reference points**.

DIGITAL

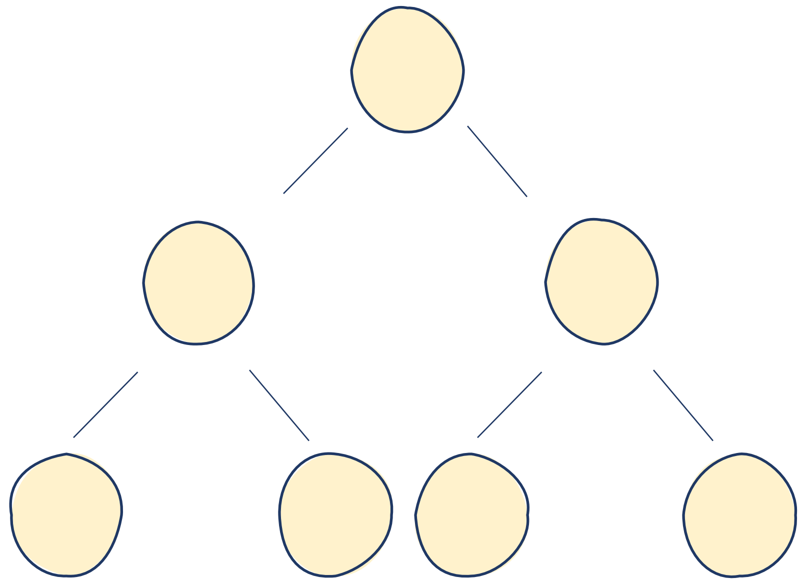
Berner Fachhochschule
Haute école spécialisée bernoise
Bern University of Applied Sciences

SHAP Implementations



TreeSHAP

- Each input x follows a **unique path** from the root to a leaf.
- Each leaf has a prediction value (e.g., probability or score).
- In Tree SHAP, the question becomes:
  - *What would happen to that path if a feature was **unknown (missing)**?*
- Some branches may **no longer be taken.**
- The model **must marginalize over those missing decisions.**



Tree SHAP calculates the **expected output** when a feature is known vs. when it is missing.

TreeSHAP: **Steps**

Let's imagine a decision tree with internal nodes splitting on features A, B, and C.

- An instance $x = (A = 1, B = 0, C = 1)$
- You want to explain the impact of feature **A** on the prediction $f(x)$

Step 1. Tree SHAP will consider **all subsets of features** (coalitions) that do *not* include A

$$S = \emptyset; S = B; S = C; S = BC$$

Step 2. For each S, we compute: $\Delta f = f\left(x_{S \cup \{A\}}\right) - f(x_S)$

- x_S: The input where we **know only the features in S**
- For the others (like A), we treat them as **unknown**

Tree SHAP **does not sample**, but rather **marginalizes over the unknowns** using the training data distribution encoded in the tree

TreeSHAP: **Steps**

Step 3: Tree Traversal with Known/Missing Features

- At each node in the tree:
  - If the split is on a **known** feature (e.g., feature in $S \cup \{A\}$, follow the path based on x's value.
  - If the split is on a **missing** feature (not in S), **take both branches**, and **weight them** by the proportion of training data that went each way.

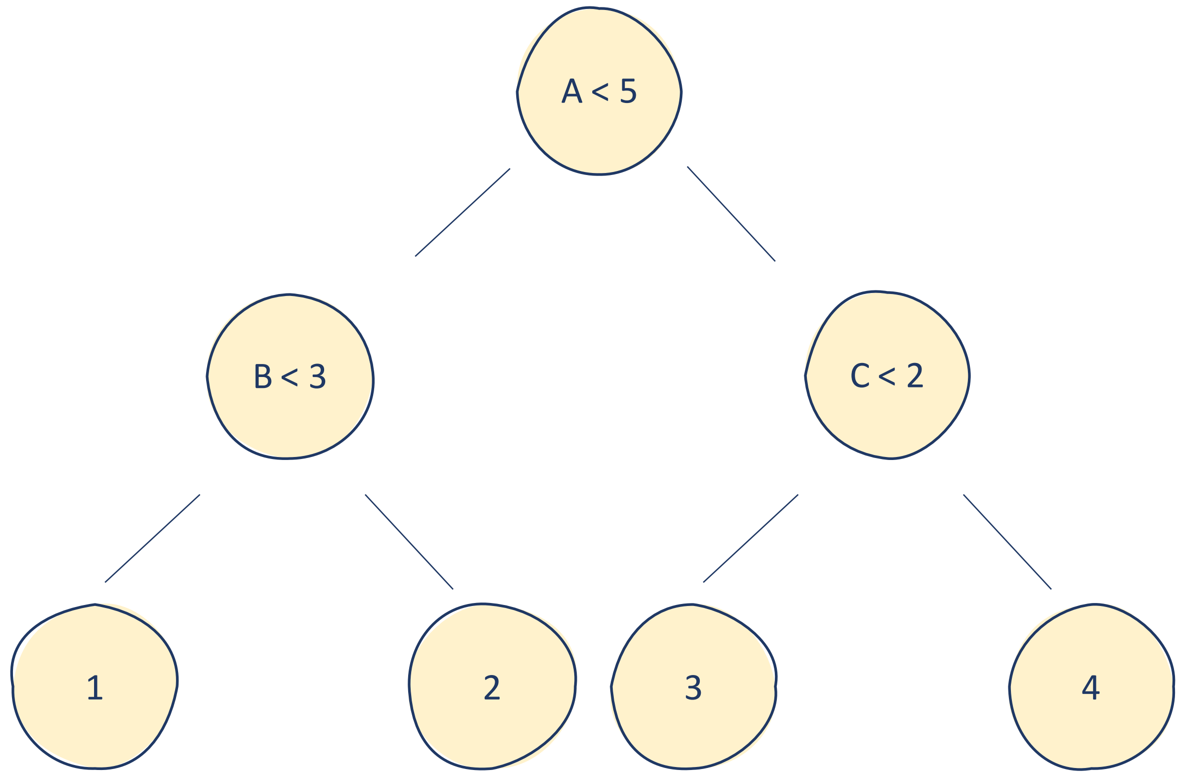
→This gives you the **expected model output** under the subset $S \cup \{A\}$, and under just S.

Step 4: Compute weighted average of differences

$$Weight = \frac{|S|!\,(2 - |S|)!}{N!}$$

S = number of features in the coalition
N = number of features

Let's **calculate** ...

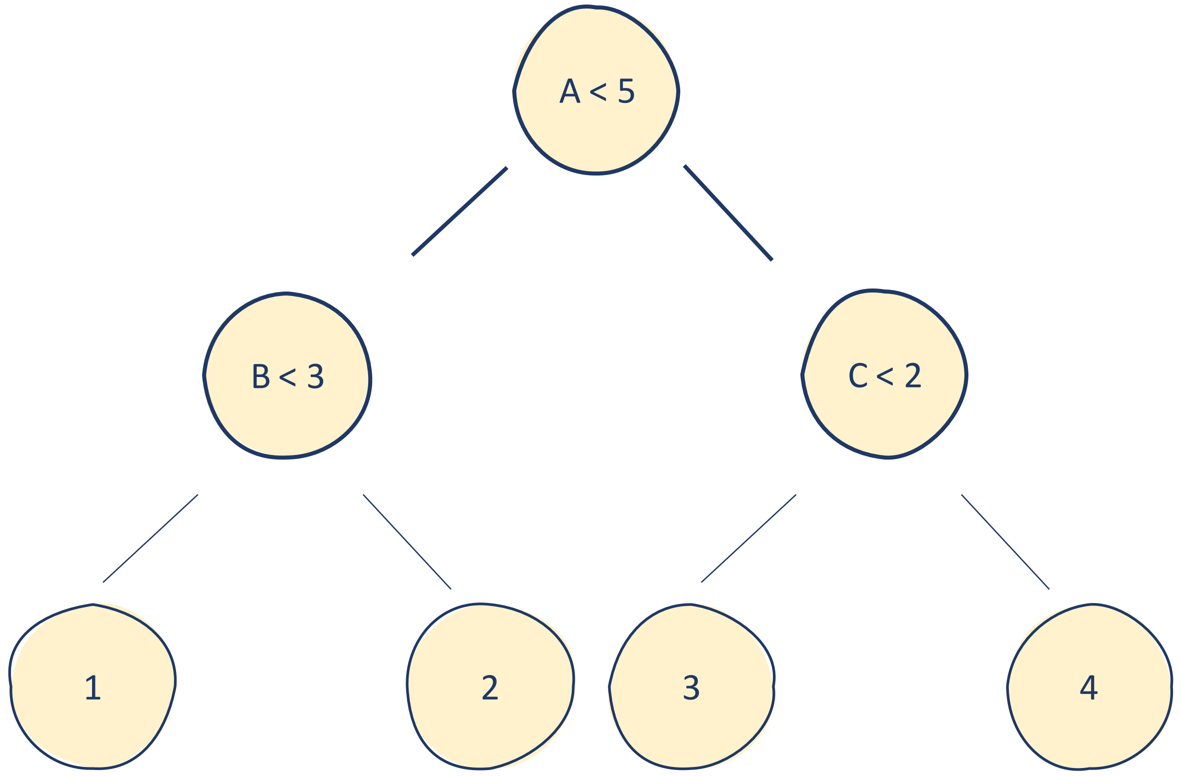


Let's assume the training data distribution is:

- At A < 5: 60% of data went **left** and40% went **right**
- At B < 3: 50%/50%
- At C < 2: 20% left, 80% right

Let’s calculate …

Input instance: $x = \{A = 6,\ B = 2,\ C = 3\}$



Q. *How do we compute the expected outcome when a feature is unknown?*

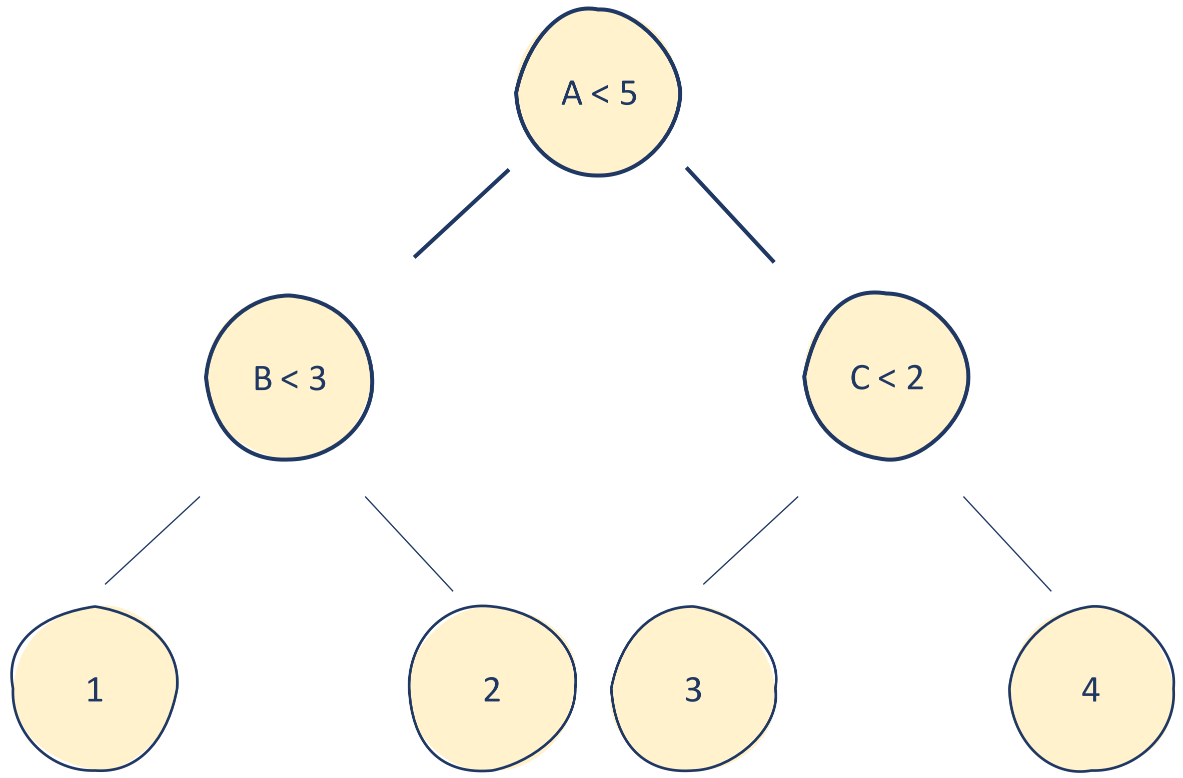
A. We take **both left and right** branches and weight by training data: Left (A < 5): 60%; Right (A ≥ 5): 40%

We compute:

$$E[f(x)|A\ unknown] = 0.6 * Left\ SubTree\ Prediction + 0.4 * Right\ SubTree\ Prediction$$

Let's **calculate** ...

Input instance: $x = \{A = 6,\ B = 2,\ C = 3\}$



We compute:

$$E[f(x)|A\ unknown] = 0.6 * Left\ SubTree\ Prediction + 0.4 * Right\ SubTree\ Prediction$$

Next, we evaluate both subtrees using known features (B and C)

- SubTree (A<5) --> B=2, value = 1
- SubTree (A>5) --> C=3, value = 2

Combine:

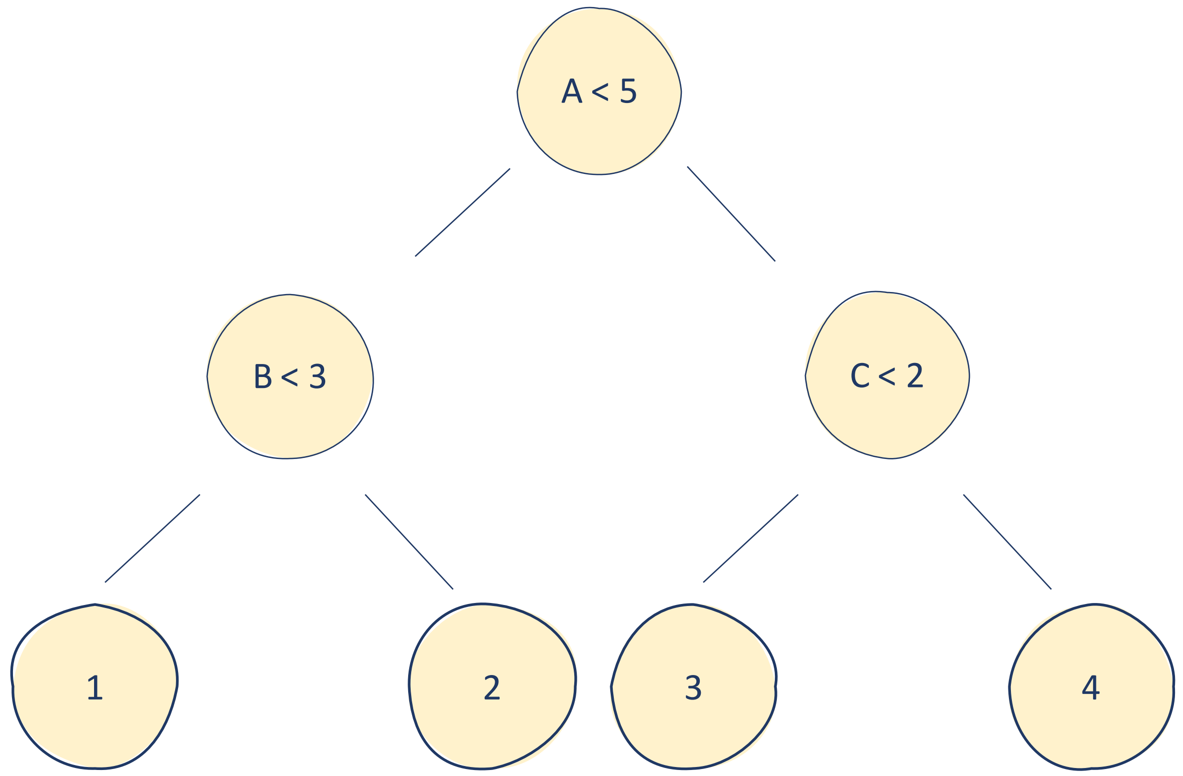
$$E[f(x)|A\ unknown] = 0.6 * 1 + 0.4 * 4 = 2.2$$

Expected model prediction **when A is unknown** but B and C are known.

Let's calculate …

Input instance: $x = \{A = 6,\ B = 2,\ C = 3\}$

$$E[f(x)|A\ unknown] = 0.6 * 1 + 0.4 * 4 = 2.2$$



Are you done? Is this the contribution of A to the prediction?

To compute the **SHAP value for feature A**, you'll repeat this logic for **all subsets of features that do not include A**, and then compare what happens **when A is added**.