DEVCONF.cz

Explainable AI for Business Processing Models

Rui Vieira

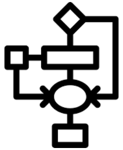
Senior Software Engineer, Red Hat

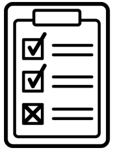
rui@redhat.com

Business process modelling
Process automation

Process automation

Modern enterprise automation





Process automation

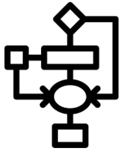
Decision Management

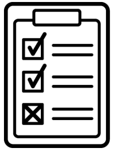
Open Standards

BPMN, DMN

Business processing models

ML/AI in business decisions





Process automation

ML
AI

Decision Management

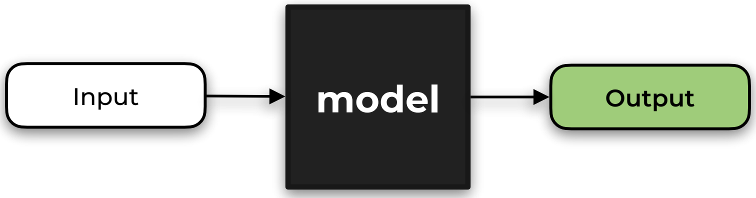
Open Standards

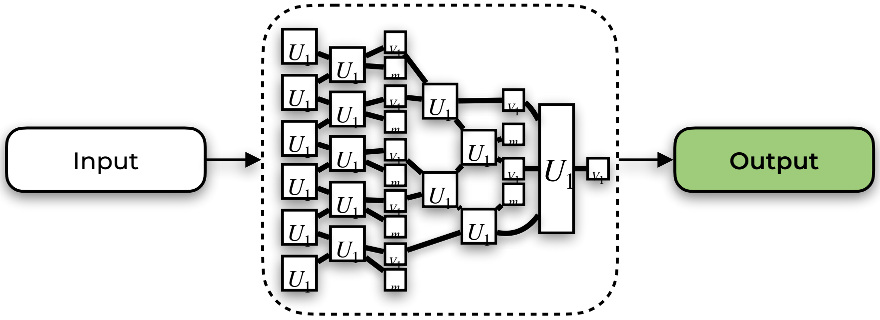
BPMN, DMN

AI?

AI and business decisions

Black-box models





AI and business decisions

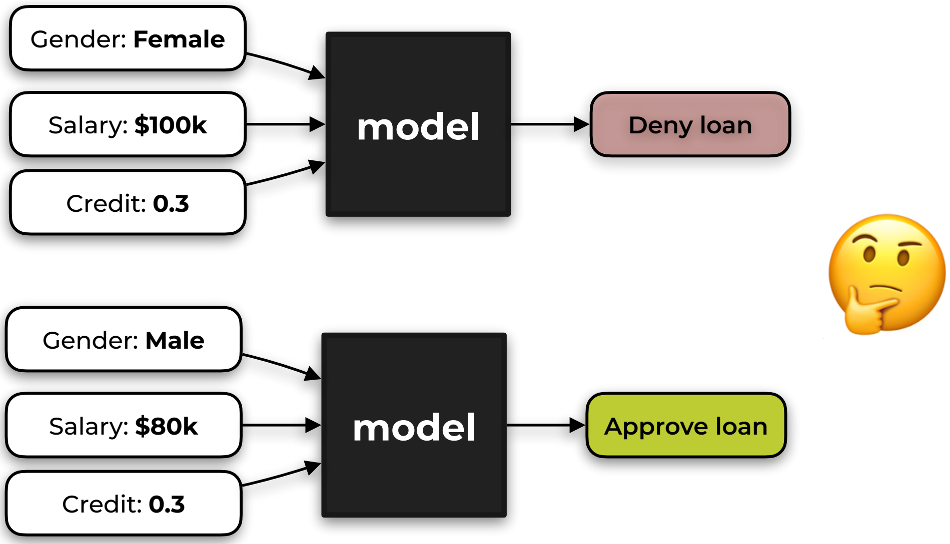
Model interpretability

Neural Network	Random Forest	K-nearest neighbours	Decision Tree	Linear regression

Less interpretable → More interpretable

AI and business decisions

Black-box models



The less we **understand**
a model,
the less we **trust** it

AI and business decisions

Regulations and ethics

- Is everyone being treated fairly?
- Are we sure our model doesn’t discriminate?
- Does it follow the law?
- Does it follow the company’s ethical guidelines?
- **Can we explain the predictions?**

General Data Protection Regulation

Explainability

Explainability

Methods

- Intrinsic
    - Restricting complexity
- **Post-hoc**
    - Apply to trained models

Explainability

Methods

- Intrinsic
    - Restricting complexity
- **Post-hoc**
    - Apply to trained models

Post-hoc

- Global
- **Local**

Explainability

Methods

- Intrinsic
  - Restricting complexity
- **Post-hoc**
  - Apply to trained models

Post-hoc

- Global
- **Local**

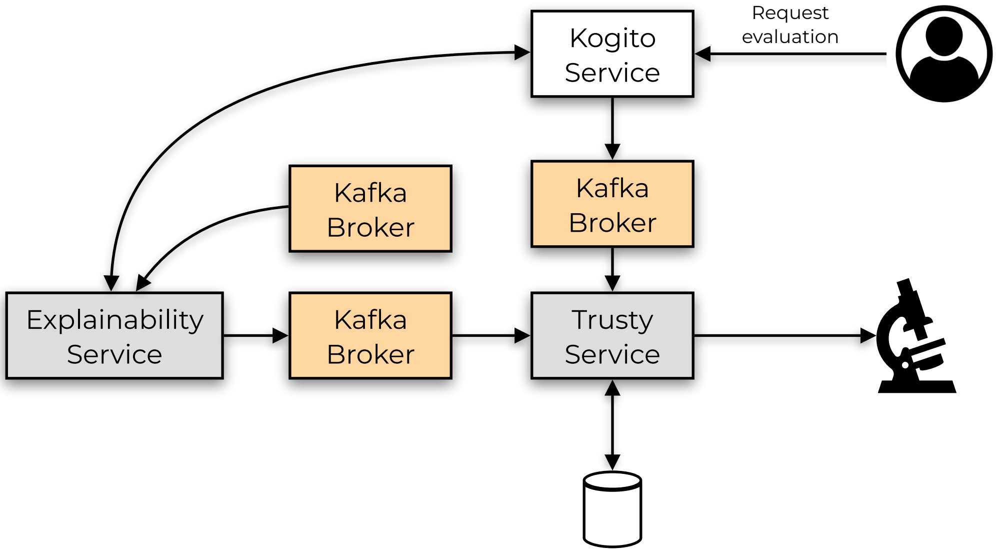
Local

- Model specific
- **Model agnostic**

TrustyAI’s explainability

TrustyAI

Kogito
https://kogito.kie.org/



Explainability algorithms

LIME

Gives you the feature importances.

“What are the most important inputs to our model?”

Counterfactuals

Counterfactuals explanations.

“What would I need to change in my inputs to get my desired outcome?”

SHAP

Feature contributions for the model’s outcome.

“How much does each input contribute to the outcome?”

LIME

Local Interpretable Model-Agnostic Explanations

LIME

Question

Which features are *more important*?
How do they *affect* the result?

LIME

Local Interpretable Model-Agnostic Explanations

“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 tech-

how much the human understands a model’s behaviour, as opposed to seeing it as a black box.

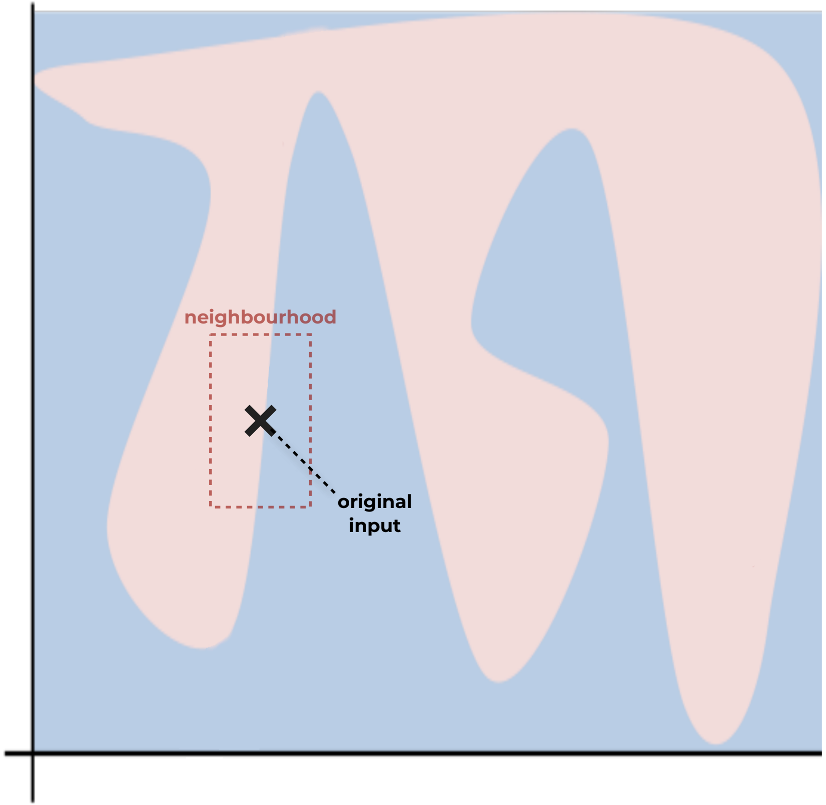
Determining trust in individual predictions is an important problem when the model is used for decision making. When using machine learning for medical diagnosis [6] or terrorism detection, for example, predictions cannot be acted upon on blind faith, as the consequences may be catastrophic.

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

LIME

Local explanation

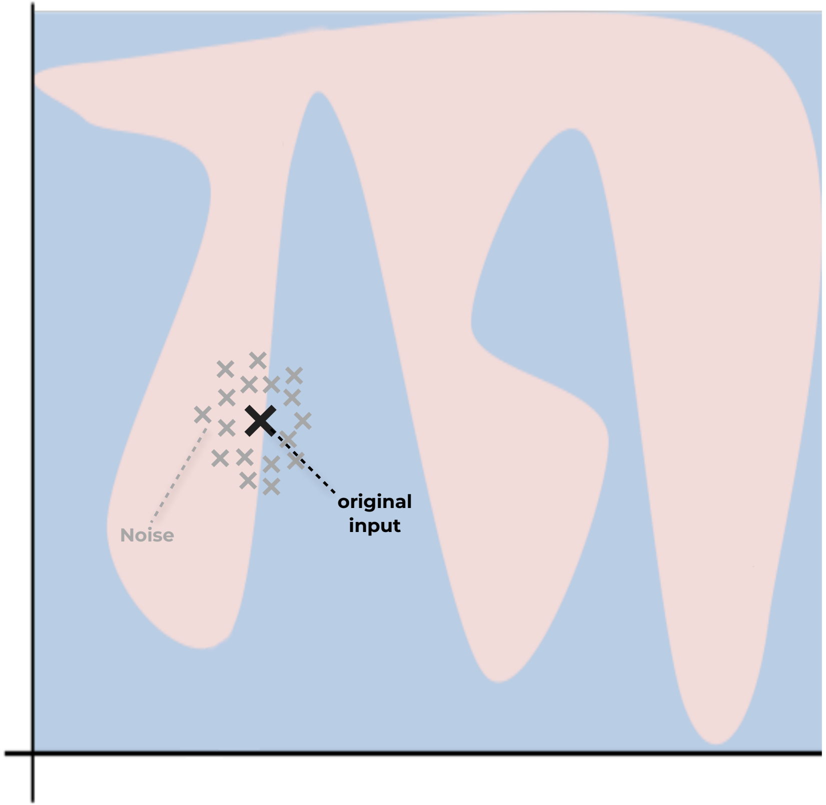
- Complex decision function
- Difficult to explain



LIME

Local explanation

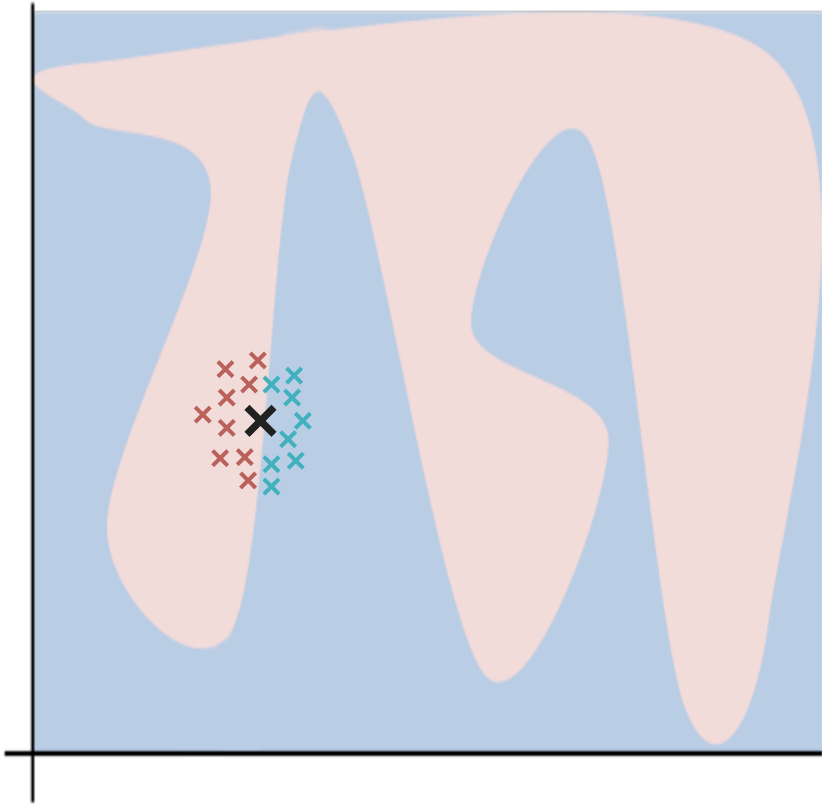
- Permute data
- Calculate distance between permutations and original data



LIME

Local explanation

- Permute data
- Calculate distance between permutations and original data
- Make predictions on data using complex model
- Pick m features best describing the complex model from the permuted data

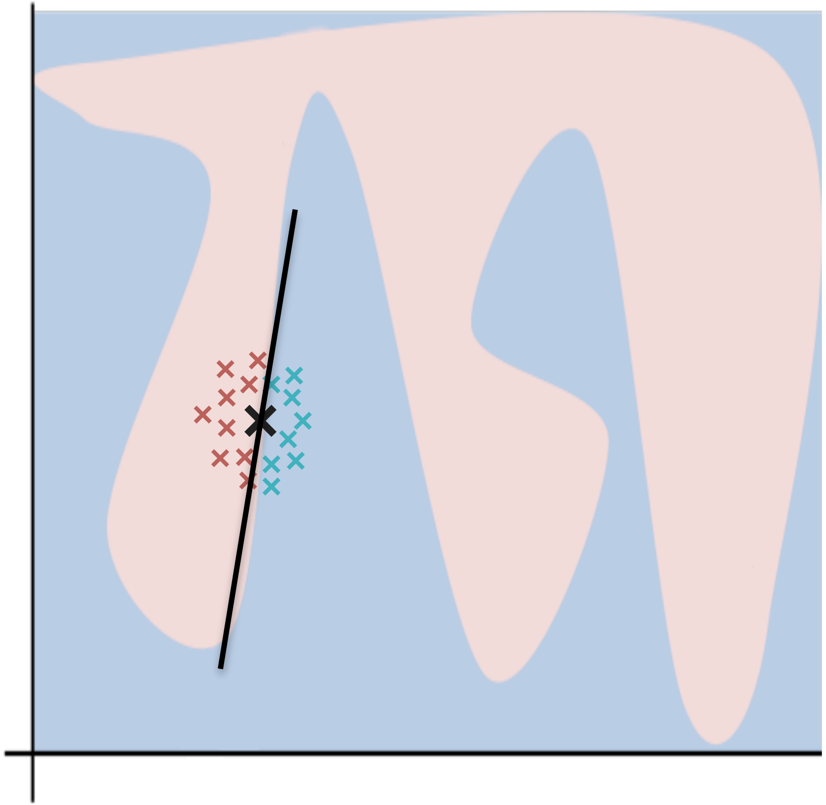


-

LIME

Local explanation

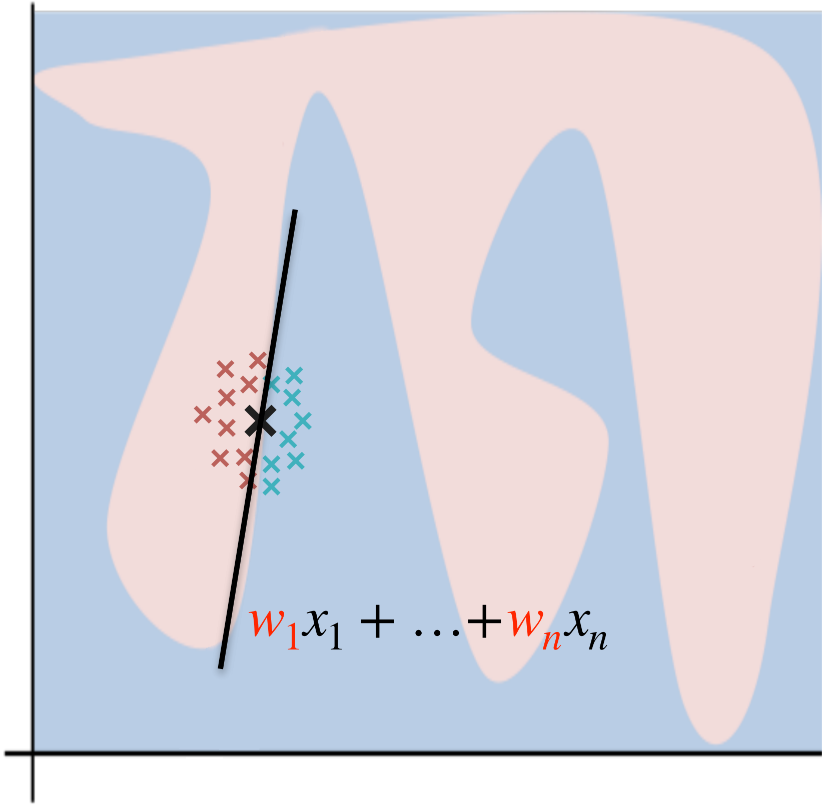
- Permute data
- Calculate distance between permutations and original data
- Make predictions on data using complex model
- Pick m features best describing the complex model from the permuted data
- Fit a simple model to the permuted data with m features and and similarity score as weights



LIME

Local explanation

- Permute data
- Calculate distance between permutations and original data
- Make predictions on data using complex model
- Pick m features best describing the complex model from the permuted data
- Fit a simple model to the permuted data with m features and and similarity score as weights
- Feature **weights** from the simple model make explanations for the complex model local behaviour

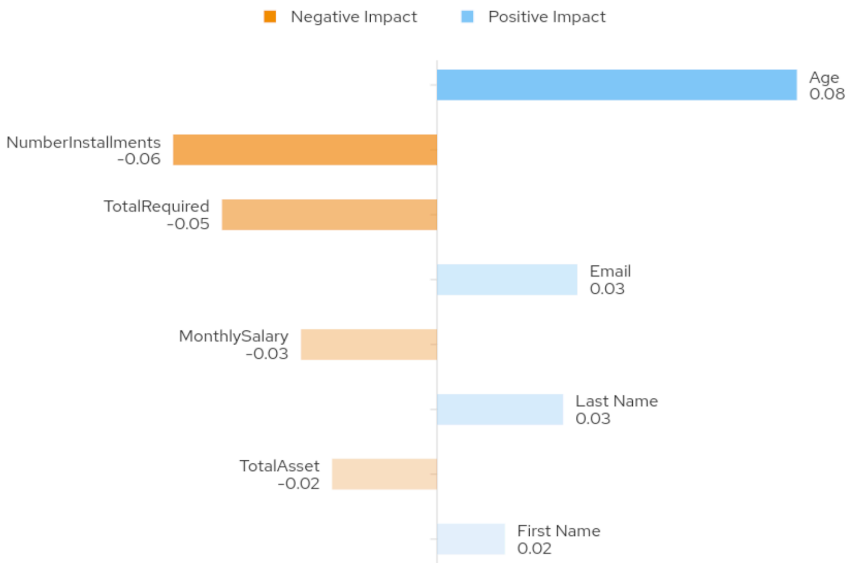


LIME

Example

```
int noOfSamples = 100;
int noOfPerturbations = 1;
LimeExplainer limeExplainer =
new LimeExplainer(noOfSamples, noOfPerturbations);

List<Feature> features = new ArrayList<>();
//…
PredictionInput input = new PredictionInput(features);
PredictionProvider predictionProvider = ...
PredictionOutput output =
predictionProvider.predictAsync(List.of(input)).get().get(0);
Prediction prediction = new Prediction(input, output);
Map<String, Saliency> saliencyMap =
limeExplainer.explainAsync(prediction, model).get();
```



Counterfactuals

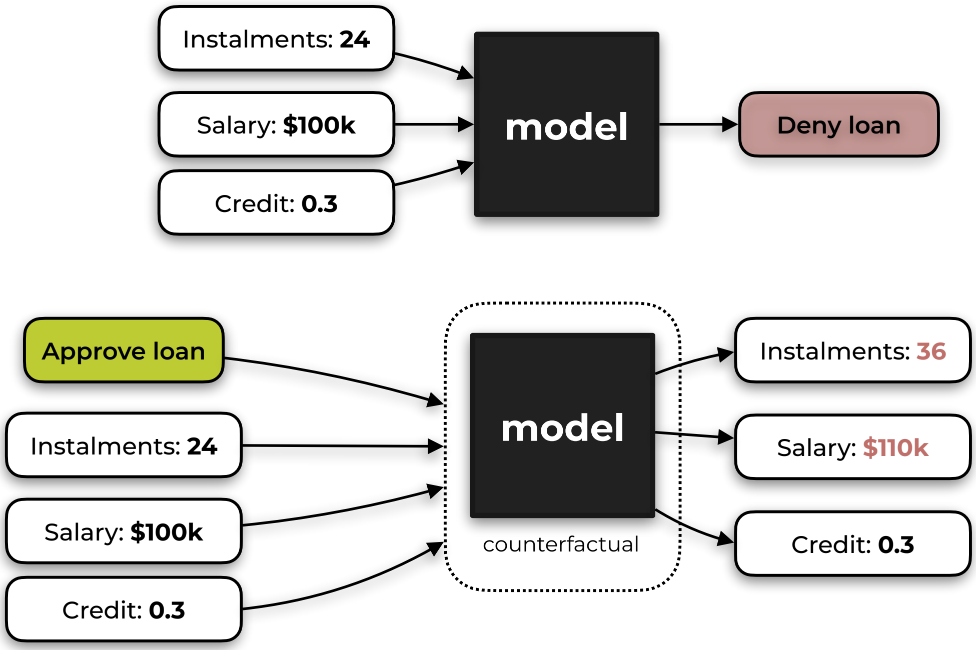
Counterfactuals

Question

In order to get *this* outcome, what *should* my inputs be?

Counterfactuals

Counterfactual explanations



Counterfactuals

Desiderata

- Validity
- Actionability
- Sparsity
- Data manifold closeness
- Causality
- Amortised inference

Counterfactuals

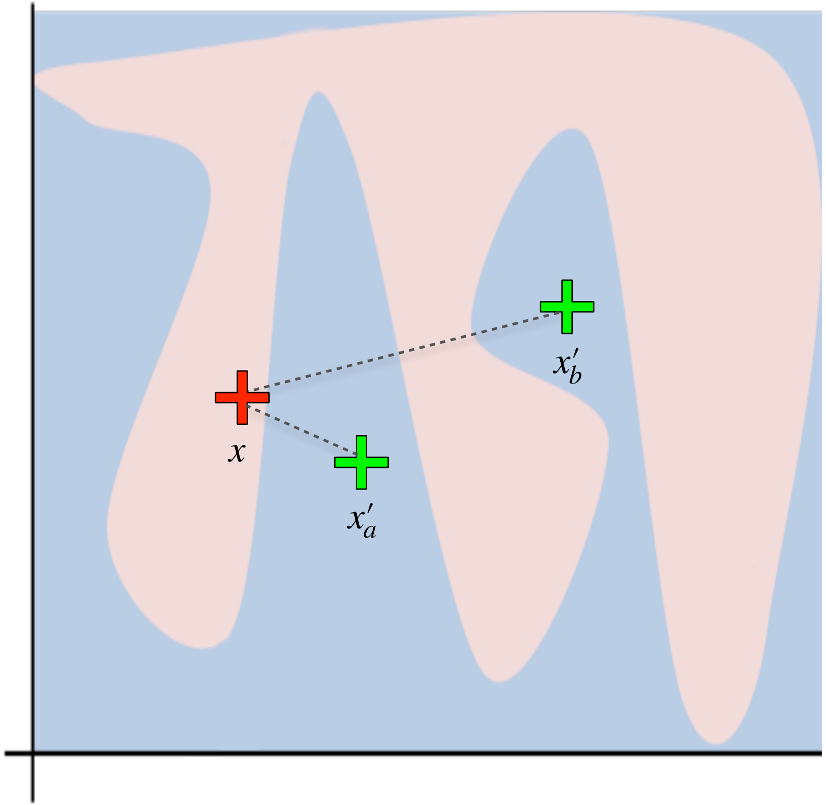
Desiderata

- **Validity**
- **Actionability**
- **Sparsity**
- Data manifold closeness
- Causality
- Amortised inference

Counterfactuals

Validity

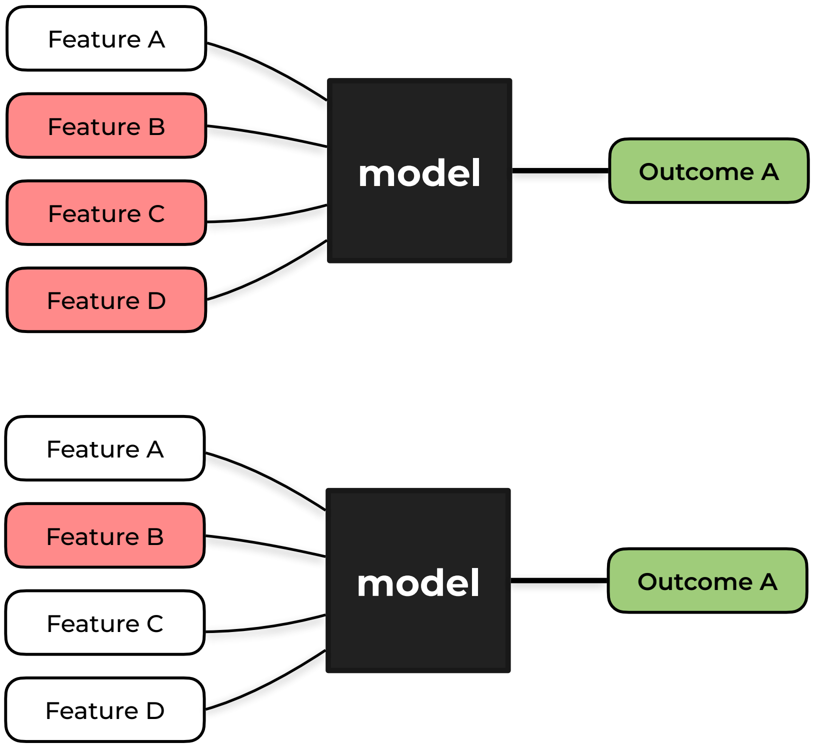
- A counterfactual has the minimum distance between its outcome and the goal and its features and the original ones
- $d(x, x'_a) < d(x, x'_b)$



Counterfactuals

Sparsity

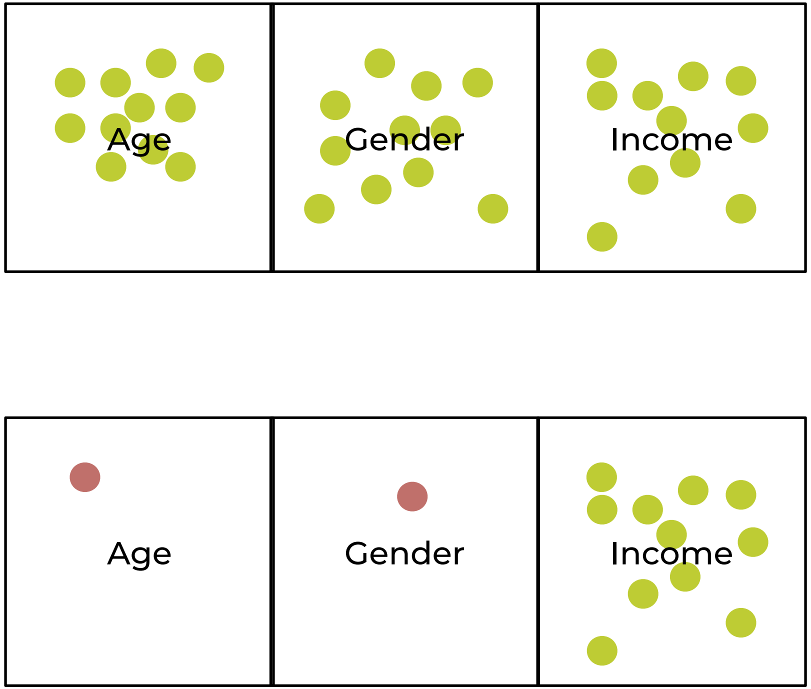
- An effective counterfactual implementation should change the least amount of features as possible
- $g(x' - x)$
- $\arg\min_{\lambda} \max \lambda \cdot (\hat{f}(x') - y')^2 + d(x, x') + g(x' - x)$



Counterfactuals

Actionability

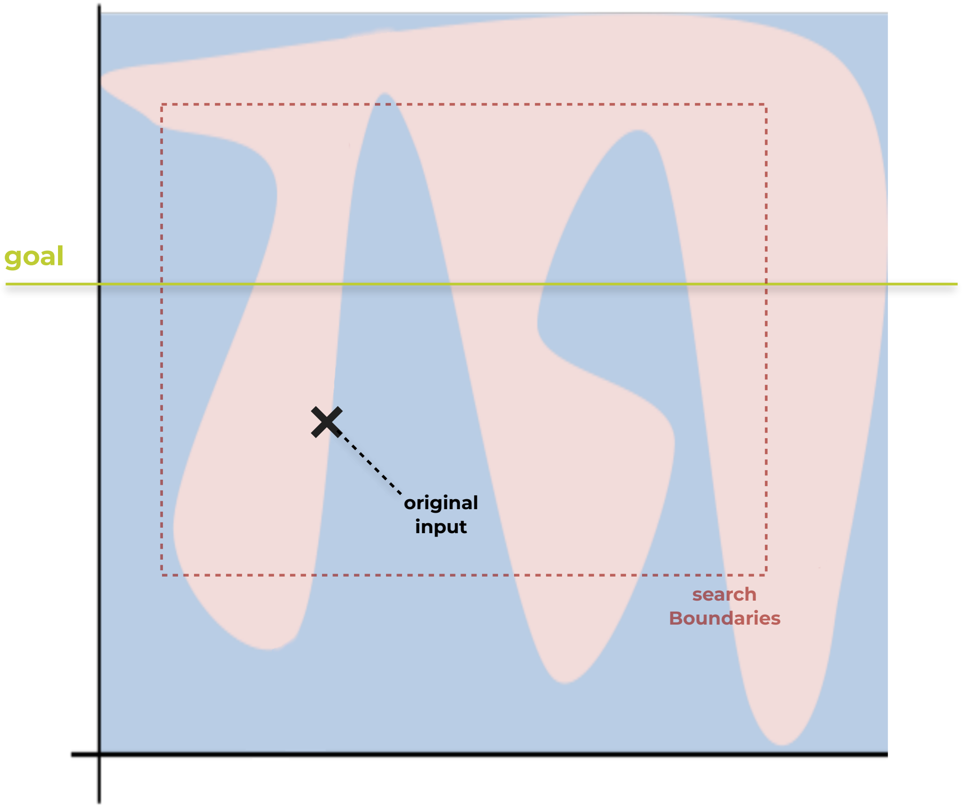
- The ability of a counterfactual method to separate between mutable and immutable features
- $\arg\min_{x' \in \mathcal{A}} \max_{\lambda} \lambda \cdot (\hat{f}(x') - y')^2 + d(x, x') + g(x' - x)$



Counterfactuals

Implementation

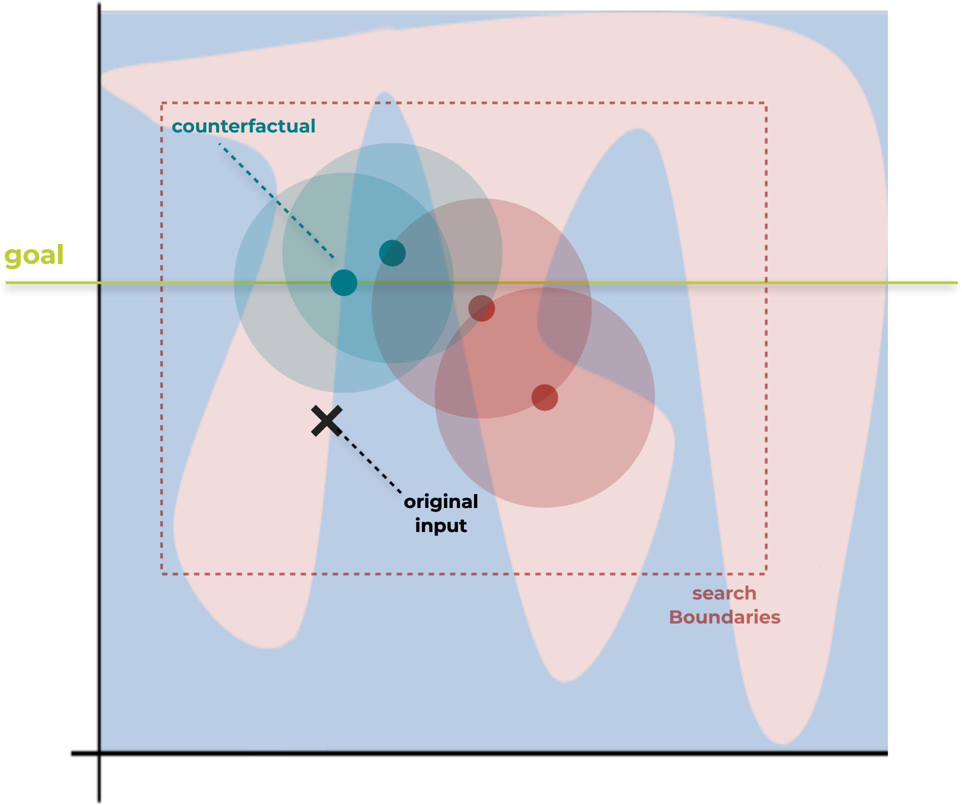
- Define search boundaries



Counterfactuals

Implementation

- Define search boundaries
- Explore feature space
- Minimising with validity, sparsity and actionability



Counterfactuals

Example

```
final List<Output> goal =
    List.of(new Output("approved", Type.BOOLEAN, new Value(true), 0.0d));
List<Feature> features = // … original features
List<FeatureDomain> featureBoundaries = new LinkedList<>();
featureBoundaries.add(NumericalFeatureDomain.create(0.0, 1000.0));
List<Boolean> constraints = new LinkedList<>();
constraints.add(false);
final CounterfactualExplainer counterfactualExplainer =
        new CounterfactualExplainer();
PredictionProvider model = // … model
PredictionInput input = new PredictionInput(features);
PredictionOutput output = new PredictionOutput(goal);
Prediction prediction =
        new CounterfactualPrediction(input,
                output,
                new PredictionFeatureDomain(featureBoundaries),
                constraints);
CounterfactualResult counterfactualResult =
    counterfactualExplainer.explainAsync(prediction, model)
        .get();
```

State our desired outcome

Counterfactuals

Example

```
final List<Output> goal =
    List.of(new Output("approved", Type.BOOLEAN, new Value(true), 0.0d));
List<Feature> features = // … original features
List<FeatureDomain> featureBoundaries = new LinkedList<>();
featureBoundaries.add(NumericalFeatureDomain.create(0.0, 1000.0));
List<Boolean> constraints = new LinkedList<>();
constraints.add(false);
final CounterfactualExplainer counterfactualExplainer =
        new CounterfactualExplainer();
PredictionProvider model = // … model
PredictionInput input = new PredictionInput(features);
PredictionOutput output = new PredictionOutput(goal);
Prediction prediction =
        new CounterfactualPrediction(input,
                output,
                new PredictionFeatureDomain(featureBoundaries),
                constraints);
CounterfactualResult counterfactualResult =
    counterfactualExplainer.explainAsync(prediction, model)
        .get();
```

State our desired outcome

Original inputs

Counterfactuals

Example

```
final List<Output> goal =
    List.of(new Output("approved", Type.BOOLEAN, new Value(true), 0.0d));
List<Feature> features = // … original features
List<FeatureDomain> featureBoundaries = new LinkedList<>();
featureBoundaries.add(NumericalFeatureDomain.create(0.0, 1000.0));
List<Boolean> constraints = new LinkedList<>();
constraints.add(false);
final CounterfactualExplainer counterfactualExplainer =
        new CounterfactualExplainer();
PredictionProvider model = // … model
PredictionInput input = new PredictionInput(features);
PredictionOutput output = new PredictionOutput(goal);
Prediction prediction =
        new CounterfactualPrediction(input,
                output,
                new PredictionFeatureDomain(featureBoundaries),
                constraints);
CounterfactualResult counterfactualResult =
    counterfactualExplainer.explainAsync(prediction, model)
        .get();
```

State our desired outcome

Original inputs

Define search boundaries and constraints

Counterfactuals

Example

```
final List<Output> goal =
    List.of(new Output("approved", Type.BOOLEAN, new Value(true), 0.0d));
List<Feature> features = // … original features
List<FeatureDomain> featureBoundaries = new LinkedList<>();
featureBoundaries.add(NumericalFeatureDomain.create(0.0, 1000.0));
List<Boolean> constraints = new LinkedList<>();
constraints.add(false);
final CounterfactualExplainer counterfactualExplainer =
        new CounterfactualExplainer();
PredictionProvider model = // … model
PredictionInput input = new PredictionInput(features);
PredictionOutput output = new PredictionOutput(goal);
Prediction prediction =
        new CounterfactualPrediction(input,
                output,
                new PredictionFeatureDomain(featureBoundaries),
                constraints);
CounterfactualResult counterfactualResult =
    counterfactualExplainer.explainAsync(prediction, model)
        .get();
```

State our desired outcome

Original inputs

Define search boundaries and constraints

Initialise explainer and build context

Counterfactuals

Example

```
final List<Output> goal =
    List.of(new Output("approved", Type.BOOLEAN, new Value(true), 0.0d));
List<Feature> features = // … original features
List<FeatureDomain> featureBoundaries = new LinkedList<>();
featureBoundaries.add(NumericalFeatureDomain.create(0.0, 1000.0));
List<Boolean> constraints = new LinkedList<>();
constraints.add(false);
final CounterfactualExplainer counterfactualExplainer =
        new CounterfactualExplainer();
PredictionProvider model = // … model
PredictionInput input = new PredictionInput(features);
PredictionOutput output = new PredictionOutput(goal);
Prediction prediction =
        new CounterfactualPrediction(input,
                output,
                new PredictionFeatureDomain(featureBoundaries),
                constraints);
CounterfactualResult counterfactualResult =
    counterfactualExplainer.explainAsync(prediction, model)
        .get();
```

State our desired outcome

Original inputs

Define search boundaries and constraints

Initialise explainer and build context

Request counterfactual

SHAP

SHapley Additive exPlanations

SHAP

Question

How much **did each individual input** ***contribute*** **to the result?**

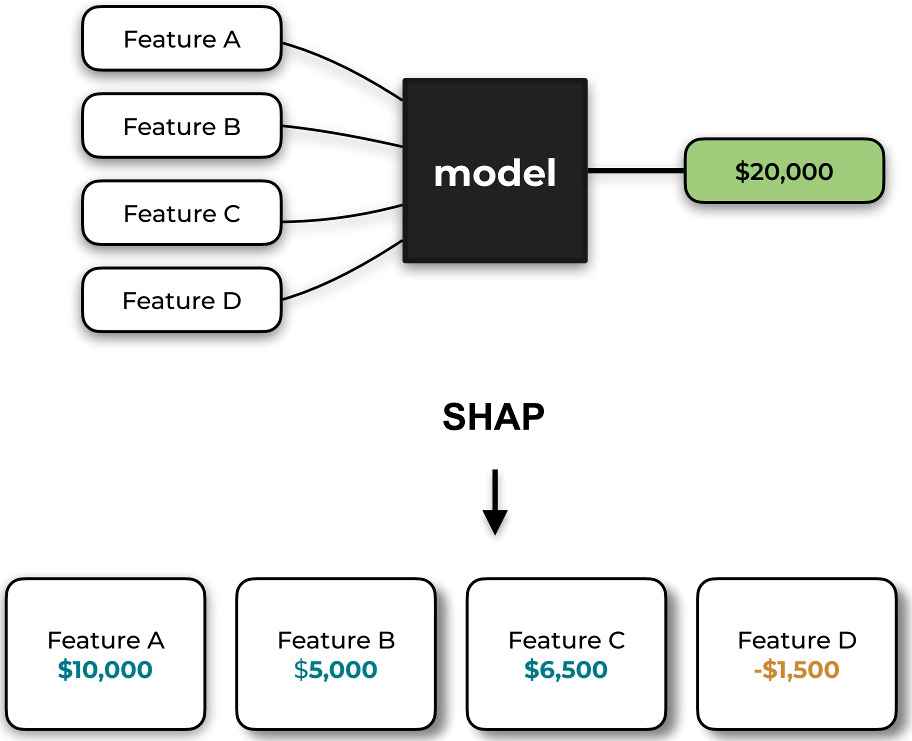
SHAP

Shapley values

- **“A Unified Approach to Interpreting Model Predictions”**
    - Lundberg, Lee, 2017

- Based on Shapley values

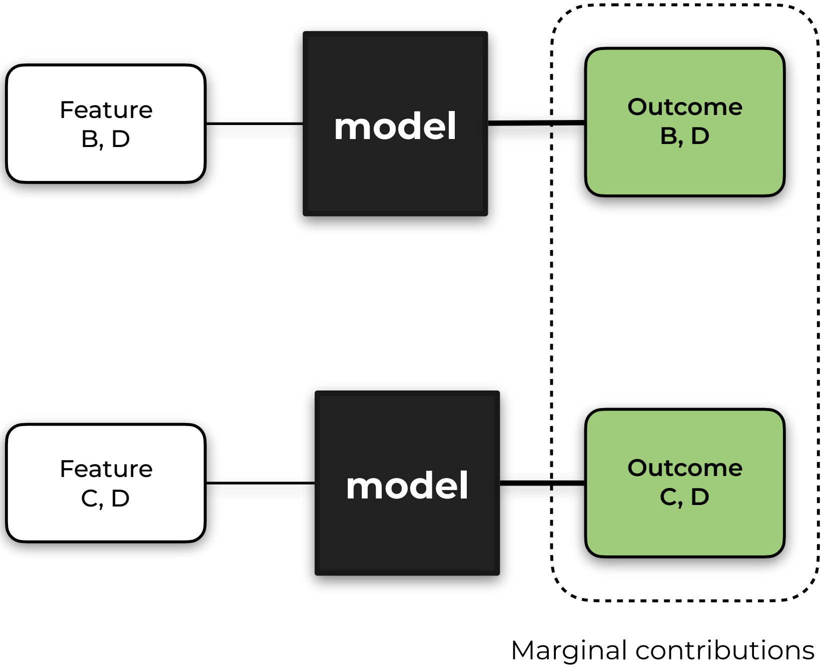
- A coalition of cooperating values each contributing to a final outcome



SHAP

Marginal contributions

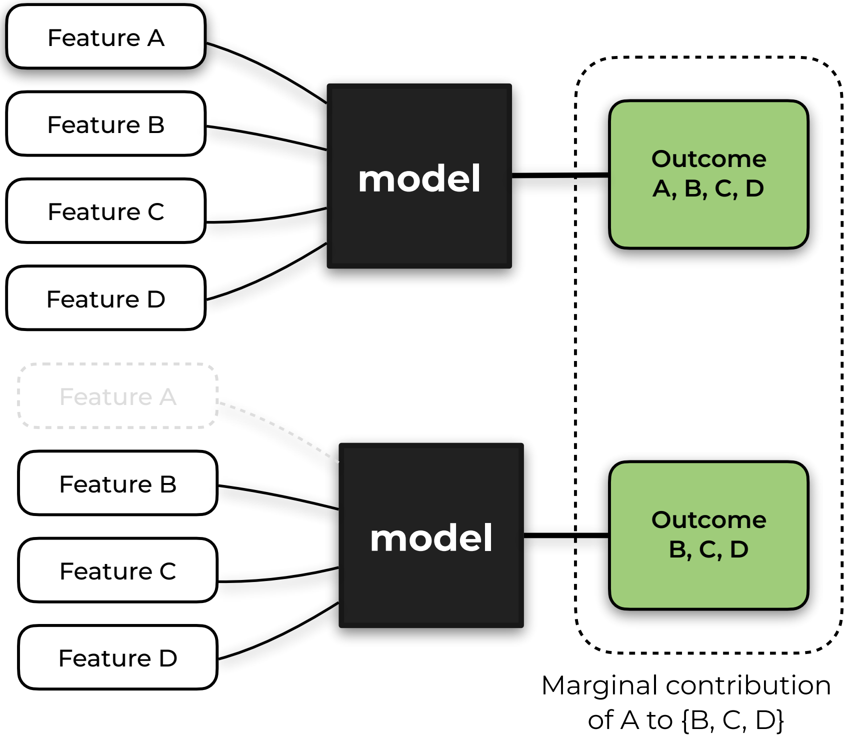
- Select a feature
- Calculate marginal contribution of A to the outcome of B, C, D



SHAP

Marginal contributions

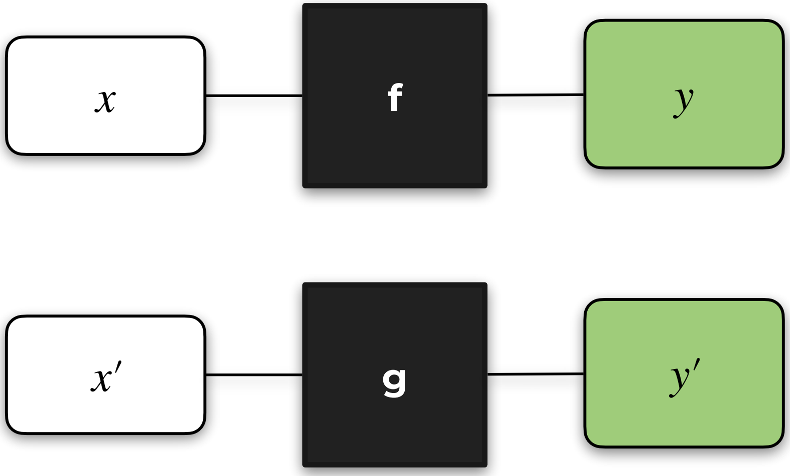
- Select a feature
- Calculate marginal contribution of A to the outcome of B, C, D
- We do this for all coalition not including A
- Mean marginal contributions is the A's Shapley value
- $$\phi_i = \frac{1}{N} \sum_{\forall C i \notin C} \frac{M_C(i)}{|C|}$$



SHAP

SHapley Additive exPlanations

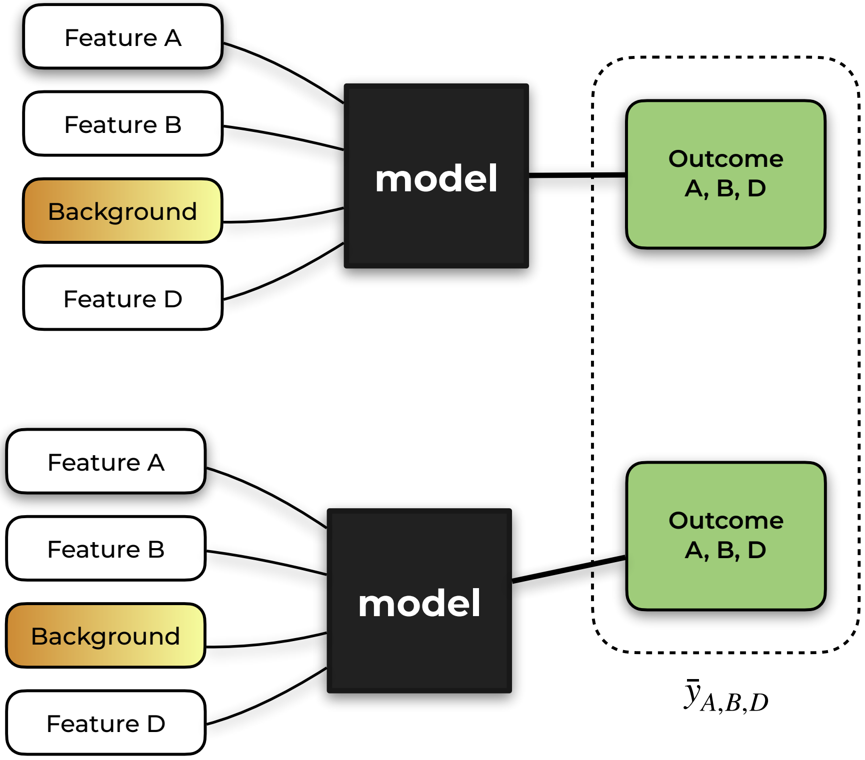
- Define a model g, which takes a simplified set of inputs x'
- If $x' \approx x$, then $y' \approx y$
- $g(x') = \phi_0 + \sum_{i}^{N} \phi_i x'_i$
- ϕ_0 is the average output of the model



SHAP

SHapley Additive exPlanations

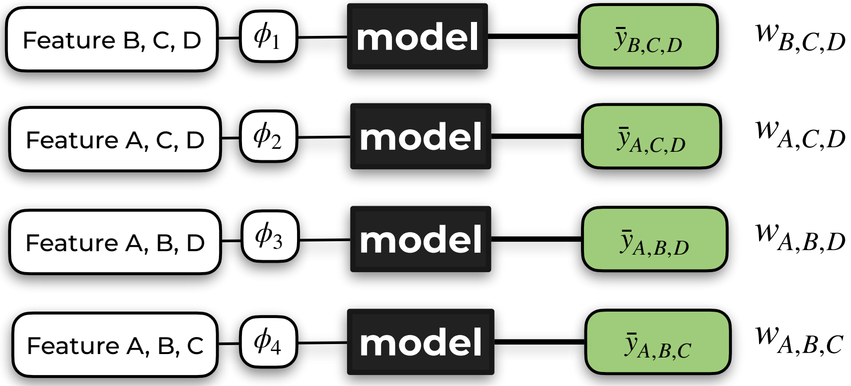
- Combinatorial explosion with number of features
- Shapley Kernel
- Fill the missing features with background data
- Calculate the average outcome for the background synthetic data



SHAP

SHapley Additive exPlanations

- Solver the linear system
- Coefficients are the Shapley values



SHAP

Example

```
List<Feature> features = // … original features
List<PredictionInput> input = List.of(new PredictionInput(features));
Prediction prediction = new SimplePrediction(input, output);
PredictionProvider model = // … model
ShapKernelExplainer shap = new ShapKernelExplainer();
Saliency[] explanation = shap.explainAsync(prediction, model)
```

Input of prediction to explain

SHAP

Example

```
List<Feature> features = // … original features
List<PredictionInput> input = List.of(new PredictionInput(features));
Prediction prediction = new SimplePrediction(input, output);
PredictionProvider model = // … model
ShapKernelExplainer shap = new ShapKernelExplainer();
Saliency[] explanation = shap.explainAsync(prediction, model)
```

Input of prediction to explain

Original prediction context

SHAP

Example

```
List<Feature> features = // … original features
List<PredictionInput> input = List.of(new PredictionInput(features));
Prediction prediction = new SimplePrediction(input, output);
PredictionProvider model = // … model
ShapKernelExplainer shap = new ShapKernelExplainer();
Saliency[] explanation = shap.explainAsync(prediction, model)
```

Input of prediction to explain

Original prediction context

Get explanation

Summary

Summary

- Explainability/Interpretability is a critical concern
- Trust on AI/ML outcomes is essential
- Active research area
- Open Source tooling and implementations

Resources

TrustyAI https://kogito.kie.org/trustyai/

TrustyAI pre-print https://arxiv.org/abs/2104.12717

TrustyAI chat https://kie.zulipchat.com/#narrow/stream/232681-trusty-ai

TrustyAI code https://github.com/kiegroup/kogito-apps/tree/main/explainability

Acknowledgements

Tommaso Teofili tteofili@redhat.com

Rob Geada rgeada@redhat.com

Rebecca Whitworth rsimmond@redhat.com

Daniele Zonca dzonca@redhat.com

Thank you