**EXPLAINABLE AI**

*Overview & Techniques*

# **INTRODUCTION**

Advances in artificial intelligence (AI) and machine learning (ML) techniques have produced headline-worthy results, causing an upsurge in the use of these algorithms across various application spaces and systems. There is clear promise for AI to provide rapid, intelligent aid in high-stakes environments like military engagements, financial transactions, and medical operations. Indeed, research and deployment of AI modeling into these fields is already well underway. However, as these models become more prolific, practitioners and stakeholders see increasing need to understand these models on a deeper level.

Explainable AI – frequently shortened to XAI, a term coined by the 2016 DARPA research initiative led by Matt Turek [1] – refers to the post-hoc analysis of a trained AI (typically ML) model, with the goal of providing explanation or insight into the predictions generated by that model. Often, the assumption made for these generated explanations is that they are *human-interpretable* (i.e. understandable to a human who might operate or interface with the model). What exactly encompasses human interpretability, however, is still an active debate in the research. Regardless, the goal of XAI efforts is to make the complex “black-box” models we deploy understandable beyond raw calculations and algorithms.

The motivations behind generating these XAI methods is diverse, and as this field develops it is helpful to outline these key motivations (and their differences) in order to select an appropriate XAI approach.

# **APPROACHES & TECHNIQUES**

There are a wide variety of XAI techniques that have been developed in the last few years, and can broadly be put into one of three core approaches –

1) building an inherently-interpretable (white-box) model or white-box surrogate of an existing model

2) performing a post-hoc analysis of a trained model to gather information about the underpinning structures and assumptions of that model, or

3) applying techniques that manipulate the input (and subsequent performance) of a trained model to understand (or alter) performance.

For all these approaches, the *scope* of their generated interpretations needs to be considered. Current XAI techniques (and their associated “explanations”) are often classified as either *local* or *global*. A *local* approach provides insight on a single prediction generated by a model. A *global* approach is applied to an entire model (i.e. all predictions/behavior of a model).

All these approaches each have competing advantages and drawbacks, and thus their utility and appropriateness depends on each unique use-case. Understanding the difference between techniques can shed light on when to use each “flavor,” and can also provide insight into where XAI efforts may go in the future.

### *APPROACH #1* | Inherently-Interpretable Models

This approach is the process of building – from the ground up – an interpretable (i.e. white-box) model for a given application. The logic behind this approach is simple: ensure from the start of a model’s development that we can interpret the model’s decisions and predictions. Initially, the white-box approach might seem like an obvious first-choice for any AI application. After all – doesn’t a white-box model wholly solve the problem of explainability?

#### **White-Box Modeling**

While it is true that a fully “interpretable” model may provide benefits, it is not a complete XAI solution. First, what actually constitutes “interpretable” is a surprisingly messy conversation (more on that later). Second, today’s state-of-the-art white-box models and algorithms are often “interpretable” because they are simply *less complex* – and therefore, usually *less accurate* than other black-box ML approaches. While these considerations do not negate white-box modeling as a promising area of ongoing XAI research, they show that no “magic bullet” XAI solution exists. Further discussion of these considerations is provided below, to provide guidance on how to approach using a white-box methodology.

As stated above, the word ‘interpretable’ is messy – meaning different people (and fields) have different (sometimes opposing) definitions. While this may seem like an irrelevant semantic quibble, the difference in definition of ‘interpretable’ is crucial to acknowledge at the start of any successful XAI effort. For example, imagine a hypothetical AI initiative funded by the US Department of Defense for building and deploying an automatic threat classification model that is ‘interpretable’. There is a constellation of key stakeholders in this project:

1. the engineers and technical staff charged with building the model,
2. the engineers or subject matter experts who will validate the model before deployment,
3. the high-level project managers and owners overseeing the project, and
4. the actual military officers and operators who will interact with and use this model once deployed.

It is clear that high-level (often non-technical) stakeholders might not deem certain models ‘interpretable’ in a way that an engineer or data practitioner might. While an engineer might identify a linear regression model as ‘interpretable’, military officers who need to make quick decisions based on the model might have a different opinion. Likewise, those tasked with evaluating and validating the model may find some ‘interpretable’ practices more or less convincing – which will risk a successful deployment. Because of this disconnect, it is critical that stakeholders for an XAI effort ensure from project inception that all relevant parties are in agreement on their application-specific criteria for ‘interpretability.’ This means, however, that a white-box model can be challenging to select and implement in a real-world project.

As stated, white-box modeling also has another commonly cited drawback: performance. Currently, white-box modeling techniques largely rely on being less complex in order to be human-interpretable. For example, a linear regression model or a decision tree might be considered ‘interpretable’ – but their performance might pale in comparison to a more complex neural network. In short, a lack of complexity can lead to less accurate models.

Because of these drawbacks, white-box modeling might not always be the most advantageous choice for an XAI project – particularly if the application requires modeling very complex functions or real-world phenomena. Rather, other techniques that can be applied to more highly-complex models may be a better fit (no pun intended). However, XAI researchers have also proposed an alternative solution to these drawbacks: surrogate models.

#### **Surrogate Modeling**

To combat the problem of accuracy loss with white-box models, XAI researchers have proposed methods for building interpretable (white-box) *surrogate* models for more complex models. In this case, a highly-complex black-box model is deployed, and then one (or many) white-box surrogate models are trained to accurately mimic the prediction of the black-box model. In turn, a black-box model becomes more human-interpretable.

Of course, this approach still suffers from the same “what is *interpretable*” difficulty that white-box modeling faces. However, this does allow engineers to deploy more complex black-box models while still providing some degree of interpretability.

Surrogate modeling might seem to better fit the ‘post-hoc analysis’ section below, considering it uses black-box modeling. However, surrogate modeling is not in the service of providing *wholly-accurate* information about a model’s internal structure. Rather, surrogate models provide an *approximate* model that behaves similarly to its parent.

This distinction significantly impacts the method’s appropriateness for different use-cases. If the motivation behind applying XAI techniques for a given effort is to meet compliance standards for a review board, this approach would not be appropriate. Similarly, if a wholly-accurate blueprint of a highly-complex model is desired, other XAI methods should be used.

However, white-box surrogate models *do* still have utility. While they may provide nothing beyond a “back-of-the-napkin” approximation of a complex model, a suite of various white-box surrogates that all provide similar results can provide reassurance to both engineers and stakeholders that a model is behaving as expected (or not). Furthermore, surrogate models can be a helpful tool in debugging a model during development, and to investigate potential bias or over-fitting in a model.

The surrogate model may then be fruitful if all that is required of ‘interpretability’ is a rough understanding of the inner-workings of a model. Many XAI projects could find benefit from white-box surrogate modeling, and this approach may be sufficient for the needs of such an effort. It can be noted that any of the existing white-box methodologies described below would present a reasonable surrogate model candidate. However, there are obvious drawbacks of this approach that might necessitate using different XAI methods.

If white-box or surrogate modeling would be an appropriate place to start for a given XAI project, a number of techniques could be explored: Decision Tree Surrogates [2], Explainable Boosting Machines [3], Scalable Bayesian Rule Lists [4], XNNs and ANNs [5], other rule-based models [6, 7], Supersparse Linear Integer Models (SLIMs) [8], and various regression techniques (including linear and logistic regression, elastic net, quantile regression, and generalized additive models [GAMs]).

### *APPROACH #2* | Post-Hoc Analysis

Post-hoc analysis refers to any XAI technique which interacts with an existing, trained model. The goal of this analysis is to glean information about internal model structure and predictions. Post-hoc analysis provides a method for making previously un-interpretable (i.e. highly complex) models interpretable. (Or, at least, to ascertain some information about a previously “black-box” model.)

It should be stated that modern ML modeling techniques are not “black-box” models in the traditional sense. Indeed, these models are not unknown to an engineer – but rather, too complex to be meaningfully understood. Lundberg and Lee [9] concisely summarize this idea, stating “the best explanation of a simple model is the model itself; it perfectly represents itself and is easy to understand… [however, for] complex models… we cannot use the original model as its own best explanation because it is not easy to understand.” As a result, XAI practitioners have been proposing methods for shining a light inside these unruly and complex “black-box” models.

The motivations for post-hoc methods are varied. Some of the most common use-cases are model debugging, establishing trust and assurance of a model before/during deployment, and post-deployment analysis of a model. In all of these cases, a more nuanced understanding of the deep complexity of a model can aid in the production and deployment of a successful AI model.

Broadly speaking, most post-hoc analysis methods fall into two categories: variable importance, and rule (or explanation) generation. Often, when deploying a model, common questions surface –

1. *how do different input values contribute to the output of this model?*
2. *how significant (or not) is this input feature to this model’s output?*
3. *why did this model make this prediction?* or
4. *how does this model ‘think’ or ‘understand’ our problem environment?*

Feature importance attempts to answer the first two questions, while rule (or explanation) generation attempts to answer the latter. These questions, on their face, might seem quite different. However, all of them are in service of a basic desire to render the mapping of a model’s input and output clear to a human interpreter.

Unlike a surrogate model, however, the post-hoc analysis methods detailed below attempt to interpret a black-box model *directly*. These methods can be useful when an XAI method requires precision and/or clear, traceable mathematical justification.

#### **Feature Importance**

These techniques rely on the observation of the input values (and corresponding output) of a model. The underlying assumption of this category is that many modern machine learning techniques are essentially different methodologies for constructing a “good-enough” function which maps intended input to desired output. Thus, understanding how the input affects a model’s output will aid in achieving an explainable model.

Below is a condensed list of the most popular (or frequently cited) techniques where feature importance is a foundational element of the methodology.

##### **Leave-One-Covariate-Out (LOCO)**

*also known as: Leave-One-Column-Out*

There are a number of currently-used algorithms that follow a similar principle: quantify the effect of a single input value (or feature) on a given model’s output. This section will focus on one particular methodology: LOCO. Many popular methods are variants of LOCO, and follow a similar structure.

The LOCO method is as follows: determine the impact of each input variable (feature) on a given model by iteratively removing features from the dataset and quantifying the impact on the model output (i.e. prediction).

This method is *model-agnostic* – meaning that it can work for any model that has an input/output mapping (regardless of internal model structure). This method can also be applied both locally (i.e. for a given prediction), and globally (i.e. for a given model or dataset). The motivation for this approach is “to focus on predictive quantities… [and] to measure variable importance directly in terms of prediction” [10].

LOCO expands upon the idea of linear regression, where coefficients in a linear model can be interpreted as the relative importance (or effect) of a given feature value. However, rather than looking at various coefficients – or, in the case of complex neural structures, potentially thousands of weights and bias values – LOCO iteratively removes a single feature from the model’s training data and assesses the resulting impact on prediction.

The algorithm [10, 11] is as follows –

For a trained model , feature matrix , target vector , initial feature values, and error measure (i.e. loss function)

1. Estimate the original model error (e.g. mean-squared error)
2. For each do:
   1. Generate feature matrix by removing feature j in the data
   2. Refit the model with data
   3. Estimate error based on the predictions of the reduced data
   4. Calculate feature importance
      1. Alternatively, the difference can be used:
   5. Sort feature by descending feature importance (FI)

The LOCO approach is similar to the concept of permutation feature importance (PFI) originally proposed by Breiman in 2001 [12] and expanded upon by Fisher, Rudin and Dominici in 2019 [13]. LOCO (and related methods) can provide a single value that describes the relationship between a feature value and its impact on a given model. As shown above, a ranked feature importance list can be produced – which can be beneficial in understanding which features most-contributed to model output.

##### **SHAP (and related methods)**

LOCO and other variable importance methodologies bear similarity to a currently (very) popular XAI technique: SHAP. SHAP values were proposed by Lundberg and Lee [9], with the motivation of unifying several other existing variable importance methods (Locally Interpretable Model-Agnostic Explanations [LIME], DeepLIFT, and Layer-Wise Relevance Propagation) using a game-theory concept originally proposed by Lloyd Shapley in 1953.

Shapley proposed a method for assigning a unique distribution (among the players in a cooperative game) of a total surplus generated by the coalition of all players. This idea of quantifying player impact in a cooperative system was then applied by Lundberg and Lee to “[assign] each feature an importance value for a particular prediction” [9]. In their original paper, Lundberg and Lee proposed both model-agnostic (Shapley sampling values, Kernel SHAP) and model-specific (Max SHAP, Deep SHAP, TreeSHAP) methods. These methods all draw from the same motivation as LOCO and its variants: quantifying individual feature contribution to model output.

While these methodologies (and their variants) present a model-agnostic, adaptable method for quantifying single-input effect on output, XAI researchers have also pointed out the limitations of this approach – namely, variable correlation or dependence. The approaches above do not, on their own, consider the impact of feature value dependence (i.e. subgroup correlation). Feature values that produce different output only when in combination with correlated feature values cannot be observed because this methodology is based on isolating the effect of a single variable on model output. As summarized by [11], “correlation makes an isolated analysis of the explanatory power of a feature complicated…which results in an erroneous ranking in feature importance and hence, in incorrect conclusions.”

Methods such as Partial Dependence Plots (PDPs) and Individual Conditional Expectation (ICE) plots can be used in combination with the methods above to gain more understanding of variable correlation. Other methods such as SHAP dependence and clustering plots, as well as calculating SHAP interaction values, have also been proposed. However, in these instances, the benefit of a single numeric value describing impact is then lost (while the benefit of quantifying variable interdependence is gained).

#### **Rule/Explanation Generation**

All the previous approaches were based on creating numeric output in order to provide more human interpretability. However, as seen, these numbers can quickly overwhelm a human interpreter as more numbers are needed to achieve greater specificity. The rule/explanation generation approach attempts to solve this problem.

This approach has a similar underlying motivation as variable importance: describe the effects of input on output of a model. However, generating explanations is about building a different descriptive artifact than a numeric value or complex mathematical model. These methods attempt to build “rules” for how a give model behaves under different circumstances (i.e. for different combinations of features).

A key motivation for generating rules, rather than additional models or numerical values, is that these rules (or explanations) can be constructed in a way that is “human-interpretable.” Rules are typically in an IF 🡪 THEN format, and may also provide coverage bounds or confidence values for a given rule. While there are several different XAI methods that adhere to this approach, the anchors method [7] has been chosen as a useful illustrative example.

##### **Anchors**

*Summary:* Generate candidate rulesets for a model, then evaluate their precision by nearest-neighbor perturbations. This method results in an optimized set of plain language rules to describe a model, where each rule can be accompanied by a precision and coverage score.

This approach was introduced by the same group of researchers that developed the popular LIME methodology [7]. The method rests on the concept of “anchor” explanations, which are described by Ribeiro, Singh, and Guestrin as “…a rule that sufficiently ‘anchors’ the prediction locally – such that changes to the rest of the feature values of the instance do not matter” [7]. In essence, the motivation of this method is to find a set of ‘anchor’ points (or rules) where a model’s prediction is almost always the same. For example (from [7]), anchors generated for a model that predicts income were as follows:

**if**  🡪 **predict**

**if** 🡪 **predict**

Anchors, unlike the LIME method, can capture non-linear behavior. It is also (often) easier for humans to engage with plain-English rules as opposed to linear equations. According to the original authors, “…the anchor approach combines the benefits of local model-agnostic explanations with the interpretability of rules…constructed in a way to best support human understanding” [7].

Formally, [7] denotes the following –

The goal of this approach is to explain the behavior of [i.e. the behavior of a given model] using plain-English rules, where is the *individual* prediction of instance . ­­Let anchor be defined as a rule (i.e. set of feature predicates) such that if all feature predicates in are true for an instance .

The method is two-fold:

1. Generate a set of candidate anchors, then
2. Iteratively evaluate these candidates until a set of high-precision rules are identified.

Additional details and formal definitions for each step are provided below.

**STEP 1 | Generate Candidate Anchors**

First, we define how to calculate the *coverage* and *precision* of an anchor candidate, given a distribution and sample from .

For a given anchor , let represent the probability that applies to samples from :

(1) .

Then, define precision as follows – a complete (but computationally intractable) method for a given black-box model and an arbitrary :

(2)

Since this cannot be computed directly, [7] propose a probabilistic definition. Let represent a desired level of precision, then do:

(3)

So, to construct a candidate anchor ,

1. Initialize to an empty rule (i.e. one which applies to every instance ), and then
2. Iteratively extend by an additional feature predicate .

Thus, a set of candidates can be iteratively constructed so that in each iteration, Now, we can formally define a method for generating a set of candidates.

**function**

**for all** , **do**

**if**  **then**

**return**

**STEP 2 | Evaluate Candidate Anchors**

Once an initial candidate set has been generated, both a greedy and a beam-search approach to find an optimized rule set were outlined by the original authors. The greedy approach is outlined below. For a deeper explanation of the beam-search approach, see [7].

**Greedy Approach**

This approach begins by generating an initial candidate set of anchors, then evaluates the *estimated precision* of each candidate to determine a “current best” candidate. Then, the “current best” is used to generate a new candidate rule set (by appending the “current best” rule with one additional predicate). This rule set is then evaluated using the same process. This continues until the current “best candidate” in a given candidate set is less precise then the anchor used to generate the current candidate set (i.e. greedy search).

Since *true precision* for a given anchor cannot be calculated, precision is estimated by taking samples from . However, a fixed sample size could often be too small (or too large) to calculate a good-enough estimation. Thus, [7] formulate the problem of precision estimation as an instance of the *pure-exploration multi-armed bandit* problem. The goal, in this case, is to minimize the number of calls to (i.e. taking the fewest samples from ) to estimate which candidate has the highest true precision. [7] frame the problem as the following:

1. Each candidate anchor is an arm
2. The true precision of on is the latent reward
3. Each pull of the arm is an evaluation of on a sample from

They propose the use of the KL-LUCB [14] algorithm to perform this precision estimation. This greedy approach will find the shortest anchors and can guarantee that, at each step, the chosen candidate was near-optimal with high probability.

However, the greedy approach only ever allows for single rule to be propagated to the next round of evaluation (and any choice at a given iteration, even if sub-optimal, is irreversible). Additionally, the greedy approach does not take coverage of a rule into account – opting instead for assuming that the shortest rules should provide the highest possible coverage. Thus, the beam-search approach was introduced by the authors to account for these drawbacks. As stated, details about this approach can be found in the original paper [7].

### *APPROACH #3* | Input & Performance Modification

All the techniques described thus far have been about either building an inherently interpretable model or analyzing existing models or datasets. This third approach, however, involves actually manipulating or changing a model (or its input). This might seem less related to the ‘explainable’ part of XAI, but the methods below can all be used in service of gaining a clearer understanding of how a model behaves – and in some cases, even improving a model’s performance in the process.

##### **Input Manipulation**

The process of input manipulation is most often used when referring to *re-weighing.* This is a method usually utilized for *disparate impact testing* (i.e. quantifying if certain features have an outsized effect on model output). Much of the research in this field is in service of adhering to necessary anti-discrimination laws (particularly in the financial sector, such as with automated lending). As such, most targeted input manipulation techniques are focused on determining how a given pre-defined feature (such as race) does (or does not) affect a model’s prediction.

The process of *re-weighing* is perturbing the value (i.e. the “weight”) of a given feature(s) to observe how those perturbations affect model output. These features are usually different demographic groups (such as race or gender), and this XAI approach can be useful for determining potential model bias for specific features. The process of perturbing input in order to test model flexibility (and to assess possible overfitting) is also well-established.

##### **Adversarial Techniques**

These XAI techniques all build upon the practice of adversarial learning and examples from existing machine learning research. Two in particular – adversarial de-biasing and reject option-based classification – will be briefly described below.

**Adversarial De-biasing**

This is a technique in which a classifier is trained on data from an existing model to try and determine if bias exists in the model – even if these features were not explicitly given to a model during training. In this approach, a classifier is trained with the following learning task: given a subset (or even complete set) of feature values for a given sample, and the output (i.e. prediction of the model) for that sample -- predict the demographic group where the corresponding sample resides.

For example, an ML model in the financial sector has been trained to determine loan eligibility given demographic information about an individual. Initially, race and gender were provided as a part of this demographic information. To apply adversarial de-biasing, an adversary model would be trained by providing, for each sample, all the demographic information (excluding race and gender) and the corresponding model prediction. This approach can be used to determine if bias is present. It can also be used to then re-evaluate a model after being re-trained without explicit protected-class demographics – or even in conjunction with the training of the original model to produce a less-biased model from inception.

**Reject Option-Based Classification**

This approach has a similar motivation to adversarial de-biasing. This process, however, is most often used during model training. To perform reject option-based classification: for each sample provided to a model during training, swap positive and negative labels (i.e. ) for individuals from unprivileged groups close to the decision boundary. The intention of this approach is to limit real-world bias or discrimination in the existing dataset, in order to build a less-biased model.

Both of these methods (and others like them), can be useful when performing model debugging. They can also be useful for providing quantitative metrics about a model (either during training, validation, or even after implementation), or even when building trust between model engineers and stakeholders.

# **FUTURE WORK**

The above methods constitute a broad picture of the current state-of-the-art in explainable AI techniques. Current thinking around XAI proposes performing a suite of these techniques to gain a clearer picture of existing (and future) ML models. However, as described for each of the three categories of approach, there are limitations and drawbacks.

Thematically, there is a shift in the research from model-dependent to model-agnostic approaches. There is also a current focus on local explanations, since these explanations are often more tractable to produce (and more actionable once created). However, there is also clear demand for more global XAI techniques that can achieve a level of precision currently reserved for local methods, while still providing an “overall picture” of a model of interest.

The field of XAI is a nascent, growing field and new methods are being explored and proposed every year. These methods will hopefully result in the construction of better, more understandable ML models – as well as provide further insight into the models that already govern much of our world today.

# ***APPENDIX A* Open-Source Repositories**

*Below is a list of various XAI techniques, short descriptions of the technique, and a (partial) list of open-source implementations of the technique.*

**Partial Dependence Plot**

Visualize the relationship between one (or two) input variables (features) and the output of a model (prediction), on average. Combines well with ICE plots.

**Open-Source Implementations**

|  |  |  |
| --- | --- | --- |
| **Name** | **Link** |  |
| sklearn | <https://scikit-learn.org/stable/modules/partial_dependence.html> | Python |

**Further Reading**

[https://christophm.github.io/interpretable-ml-book/ice.html](https://christophm.github.io/interpretable-ml-book/pdp.html#pdp)

**Accumulated Local Effect (ALE) Plot**

Visualize the shape of the relationship between model input (features) and model output (prediction) for complex, non-linear models.

**Open-Source Implementations**

|  |  |  |
| --- | --- | --- |
| **Name** | **Link** |  |
| ALEPython | <https://github.com/blent-ai/ALEPython> | Python |
| Alibi EXPLAIN | <https://docs.seldon.io/projects/alibi/en/stable/methods/ALE.html> | Python, Larger Package Suite |
| CRAN | <https://cran.r-project.org/web/packages/ALEPlot/index.html> | R, author implementation |

**Individual Conditional Expectation (ICE) Plot**

Local implementation of the (global) partial dependence plot (PDP) which depicts how a model behaves per observation (i.e. one row of data). Can be used in conjunction with PDPs to identify strong interactions between input variables by observing where PDP and ICE plots diverge.

**Open-Source Implementations**

|  |  |  |
| --- | --- | --- |
| **Name** | **Link** |  |
| PyCEBox | <https://github.com/AustinRochford/PyCEbox> | Python, part of PyPI library |
| sklearn | <https://scikit-learn.org/stable/modules/partial_dependence.html> | Python |

**Further Reading**

[https://christophm.github.io/interpretable-ml-book/ice.html](https://christophm.github.io/interpretable-ml-book/pdp.html#pdp)

**Residual Plot**

Plot residuals for a complex, black-box model to determine how the model responds to different input values (features).

**Open-Source Implementations**

|  |  |  |
| --- | --- | --- |
| **Name** | **Link** |  |
| YellowBrick | <https://www.scikit-yb.org/en/latest/api/regressor/residuals.html> | Python, for regression models |
| SeaBorn | [https://seaborn.pydata.org/examples/residplot.html](https://seaborn.pydata.org/generated/seaborn.residplot.html) | Python, larger package suite |

**Decision Tree Surrogate**

Train a decision tree by using the original inputs (features) and outputs (predictions) of a more complex, black-box model. No theoretical guarantee the surrogate will be highly representative of original model).

**Open-Source Implementations**

|  |  |  |
| --- | --- | --- |
| **Name** | **Link** |  |
| Skater | <https://github.com/oracle/Skater/blob/main/skater/core/global_interpretation/tree_surrogate.py> | Python |
| sklearn | <https://scikit-learn.org/stable/modules/tree.html> | Python |

**LORE**

Build surrogate of local network structures using a genetic algorithm, then derive decision rules from those surrogates.

**Open-Source Implementations**

|  |  |  |
| --- | --- | --- |
| **Name** | **Link** |  |
| lore | <https://github.com/riccotti/LORE> | Python, author implementation |

**Anchors**

Generate high-precision sets of plain language rules to describe prediction in terms of input variable values. Currently most-applicable to classification applications.

**Open-Source Implementations**

|  |  |  |
| --- | --- | --- |
| **Name** | **Link** |  |
| anchor | <https://github.com/marcotcr/anchor> | Python, part of PyPI |
| Alibi EXPLAIN | <https://docs.seldon.io/projects/alibi/en/stable/methods/Anchors.html> | Python, larger package suite |

**Further Reading**

https://christophm.github.io/interpretable-ml-book/anchors.html

**Counterfactual Explanations**

Generate human-readable rules for how to change an input (feature) value of an instance (row of data) in order to “flip” (i.e. change) the prediction of a model. A counterfactual explanation of a prediction describes the smallest change to the feature values that changes the prediction to a pre-defined output [15].

**Open-Source Implementations**

|  |  |  |
| --- | --- | --- |
| **Name** | **Link** |  |
| counterfactuals | <https://github.com/dandls/moc/tree/master/counterfactuals> | R, author implementation (Dandl) |
| Alibi EXPLAIN | <https://docs.seldon.io/projects/alibi/en/stable/methods/CF.html> | Python, larger package suite (simple) |
| Alibi EXPLAIN | <https://docs.seldon.io/projects/alibi/en/stable/methods/CFProto.html> | Python, larger package suite (extended) |
| mace | <https://github.com/amirhk/mace> | Python, author implementation (MACE, MINT) |
| interpretml | <https://github.com/interpretml/DiCE> | Python (DiCE) |

**Rule-Based Modeling Techniques (Various)**

Learn decision rules (IF→THEN) for predicting a specific output. Can be used as a stand- alone model, or applied to an existing black-box model. Rule lists for complex models might become impractically long in some cases.

**Open-Source Implementations**

|  |  |  |
| --- | --- | --- |
| **Name** | **Link** |  |
| weka | <https://weka.sourceforge.io/doc.dev/weka/classifiers/rules/package-summary.html> | Java (implementations for RIPPER, M5Rules, OneR, PART, and more) |
| imodels | <https://github.com/csinva/imodels> | Python, larger package suite (implementations for SLIPPER, Bayesian rule list, CORELS, CART, and more) |
| skater | <https://github.com/oracle/Skater> | Python, larger package suite, implements Scalable Bayesian Rule List (SBRL) |

**Further Reading**

https://christophm.github.io/interpretable-ml-book/rules.html

**Leave-One-Covariate-Out (LOCO)**

Determine the impact of each input variable (feature) on a given model by iteratively removing features from the dataset and quantifying the impact on the model output (prediction).

**Open-Source Implementations**

|  |  |  |
| --- | --- | --- |
| **Name** | **Link** |  |
| conformal | <https://github.com/ryantibs/conformal> | R, author implementation |

**Local Interpretable Model-Agnostic Explanations (LIME)**

From “An Introduction to Machine Learning Interpretability” –

“*… [enforce] a uniform splitting strategy in constituent decision trees, where binary splits of a variable in one direction always increase the average value of the dependent variable in the resultant child node, and binary splits of the variable in the other direction always decrease the average value of the dependent variable in the other resultant child node.”* [

**Open-Source Implementations**

|  |  |  |
| --- | --- | --- |
| **Name** | **Link** |  |
| XGBoost | <https://xgboost.readthedocs.io/en/stable/python/index.html> | Python, larger package suite |

**Local Interpretable Model-Agnostic Explanations (LIME)**

Use local linear surrogate models to generate sparse, simplified explanations of local variables that are human-interpretable.

**Open-Source Implementations**

|  |  |  |
| --- | --- | --- |
| **Name** | **Link** |  |
| sklearn | <https://scikit-learn.org/stable/modules/partial_dependence.html> | Python |

**Further Reading**

https://christophm.github.io/interpretable-ml-book/lime.html

**TreeInterpreter**

Decomposes tree algorithms into a list of individual variable (feature) contributions - both globally, and for each example (row of data). Used with decision trees, random forests, and GBMs).

**Open-Source Implementations**

|  |  |  |
| --- | --- | --- |
| **Name** | **Link** |  |
| treeinterpreter | <https://blog.datadive.net/random-forest-interpretation-with-scikit-learn/> | Python, author implementation |
| eli5 | [https://eli5.readthedocs.io/en/latest/libraries/sklearn.html? highlight=treeinterpreter#decision-trees-ensembles](https://eli5.readthedocs.io/en/latest/libraries/sklearn.html?highlight=treeinterpreter&decision-trees-ensembles) | Python, larger package suite |

**Super-sparse Linear Integer Model (SLIM)**

A type of predictive model where input variables are constrained to only be subject to basic mathematical functions (+, -, \*, /) when generating predictions. This generates human- interpretable functions which are useful for high-stakes decision-making.

**Open-Source Implementations**

|  |  |  |
| --- | --- | --- |
| **Name** | **Link** |  |
| slim-python | <https://github.com/ustunb/slim-python> | Python (6+ year-old repository) |

# ***REFERENCES***

1. Turek, M. (2016). *Explainable AI (XAI).* Defense Advanced Research Project Agency. <https://www.darpa.mil/program/explainable-artificial-intelligence>
2. Bastani, O., Kim, C., & Bastani, H. (2017). *Interpreting Blackbox Models via Model Extraction.* doi:10.48550/ARXIV.1705.08504
3. Lou, Y., Caruana, R., Gehrke, J., & Hooker, G. (2013). Accurate Intelligible Models with Pairwise Interactions. In Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (pp. 623–631). Association for Computing Machinery.
4. Yang, H., Rudin, C., & Seltzer, M. (2016). *Scalable Bayesian Rule Lists*. doi:10.48550/ARXIV.1602.08610
5. Vaughan, J., Sudjianto, A., Brahimi, E., Chen, J., & Nair, V. N. (2018). *Explainable Neural Networks based on Additive Index Models.* doi:10.48550/ARXIV.1806.01933
6. Guidotti, R., Monreale, A., Ruggieri, S., Pedreschi, D., Turini, F., & Giannotti, F.. (2018). Local Rule-Based Explanations of Black Box Decision Systems.
7. Marco Tulio Ribeiro, Sameer Singh, & Carlos Guestrin (2018). Anchors: High-Precision Model-Agnostic Explanations. In *AAAI*.
8. Ustun, B., Tracà, S., & Rudin, C.. (2013). Supersparse Linear Integer Models for Interpretable Classification.
9. Lundberg, S., & Lee, S.I.. (2017). A Unified Approach to Interpreting Model Predictions.
10. Lei, J., G'Sell, M., Rinaldo, A., Tibshirani, R., & Wasserman, L.. (2016). Distribution-Free Predictive Inference For Regression.
11. Altmann, T., Bodensteiner, J., Dankers, C., Dassen, T., Fritz, N., & Gruber, S. et al. (2020). Limitations of Interpretable Machine Learning Methods [Ebook]. Retrieved from <https://slds-lmu.github.io/iml_methods_limitations/>
12. Breiman, L. (2001). Random Forests. Machine Learning, 45 (1), 5-32. <https://doi.org/https://doi.org/10.1023/A:1010933404324>
13. Aaron Fisher, Cynthia Rudin, & Francesca Dominici (2019). All Models are Wrong, but Many are Useful: Learning a Variable's Importance by Studying an Entire Class of Prediction Models Simultaneously. *Journal of Machine Learning Research*, 20(177), 1–81.
14. Kaufmann, E., & Kalyanakrishnan, S.. (2013). Information Complexity in Bandit Subset Selection. *Proceedings of the 26th Annual Conference on Learning Theory*, in *Proceedings of Machine Learning Research* 30:228-251 Available from https://proceedings.mlr.press/v30/Kaufmann13.html.
15. Christoph Molnar (2022). Interpretable Machine Learning. Available from https://christophm.github.io/interpretable-ml-book.
16. Hall, P., & Gill, N. (2018). An introduction to machine learning interpretability: an applied perspective on fairness, accountability, transparency, and explainable AI. First edition. Sebastopol, CA: O'Reilly Media.