Alibi Explain: Algorithms for Explaining Machine Learning Models

Janis Klaise  
jk@seldon.io  
Arnaud Van Looveren  
avl@seldon.io  
Giovanni Vacanti  
gv@seldon.io  
Seldon Technologies Limited  
Alexandru Coca∗  
ac2123@cam.ac.uk  
University of Cambridge

Abstract

We introduce Alibi Explain, an open-source Python library for explaining predictions of machine learning models (https://github.com/SeldonIO/alibi). The library features state-of-the-art explainability algorithms for classification and regression models. The algorithms cover both the model-agnostic (black-box) and model-specific (white-box) setting, cater for multiple data types (tabular, text, images) and explanation scope (local and global explanations). The library exposes a unified API enabling users to work with explanations in a consistent way. Alibi adheres to best development practices featuring extensive testing of code correctness and algorithm convergence in a continuous integration environment. The library comes with extensive documentation of both usage and theoretical background of methods, and a suite of worked end-to-end use cases. Alibi aims to be a production-ready toolkit with integrations into machine learning deployment platforms such as Seldon Core and KFServing, and distributed explanation capabilities using Ray.

Keywords: Explainability, Open Source, Python

1. Introduction

Explainable AI, also known as model explainability, refers to techniques for elucidating the reasons behind predictions made by complex, opaque machine learning models in a format that is understandable to human observers (Molnar, 2019). The ability to explain predictions helps to build trust in the model’s decision making process and is therefore an integral part of a robust machine learning system (Bhatt et al., 2020; Klaise et al., 2020).

The desired insights provided by explanations differ strongly dependent on the consumer of the explanations, ranging from data scientists debugging models to regulators auditing them. As a result, multiple methods are needed to cater to the needs of the target audience (ICO, 2019; Bhatt et al., 2020). Moreover, standalone explanation methods can generate non-informative or even misleading explanations (Heo et al., 2019). This means that a holistic approach to model explanations is required.

We present Alibi which aims to bridge the gap between the fast growing area of explainability research and industry. The goal of Alibi is to host reference implementations

∗. Work done at Seldon.
of a broad range of production-ready model explanation algorithms. Alibi contains local, global, black- and white-box post-hoc explanation methods to cover a wide variety of use cases. Whilst there are a few contemporaneous explainability libraries (see Table 1), Alibi is uniquely focused on providing production level explanation methods with deployment platform integrations and a distributed backend.

2. Project Focus

Scope of applications. Model explainability often requires a holistic approach as there is no one-size-fits-all solution. This is reflected in the breadth of algorithms currently supported (Section 2.1) and the guidance of their applicability (Table 2).

Build robustness. Extensive testing of code correctness and algorithm convergence is done using pytest under various Python versions. Tests are executed on every pull request via a continuous integration setup using Github Actions.

Documentation and examples. The library features comprehensive documentation and extensive in-depth examples of use cases\(^1\). The documentation includes usage and theoretical background of each method. Furthermore, the scope and applicability of all methods is clearly documented to help practitioners quickly identify relevant algorithms (Table 2).

Industry relevance. Alibi is integrated into deployment platforms Seldon Core (Cox et al., 2018) and KFServing (KFServing, 2019) to enable deploying explanations into production. Alibi also features a distributed backend using Ray (Moritz et al., 2018) to enable large-scale parallel computation of batch explanations.

We also provide a more detailed feature comparison with other actively developed explanation libraries, see Table 1.

<table>
<thead>
<tr>
<th>Library</th>
<th>Local post-hoc</th>
<th>Global post-hoc</th>
<th>Feature attributions</th>
<th>Anchors</th>
<th>Counterfactuals</th>
<th>Multiple data types</th>
<th>Deployment options</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alibi Explain</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>AIX360</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interpret</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Captum</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>iNNvestigate</td>
<td>✓</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Comparison with related explanation libraries AIX360 (Arya et al., 2020), Interpret (Nori et al., 2019), Captum (Kokhlikyan et al., 2020), iNNvestigate (Alber et al., 2019). Libraries are selected and compared on the basis of providing post-hoc, black-box or white-box, local or global explanation techniques implemented in Python which have had some development activity in the past 12 months.

2.1 Algorithms

The current version of the library includes the following explanation algorithms (c.f. Table 2 for detailed capabilities): 1. Accumulated Local Effects (ALE), Apley and Zhu (2016): calculate global feature effects on the model predictions. 2. Anchor explanations, Ribeiro

\(^1\) https://docs.seldon.io/projects/alibi/en/latest/
Alibi Explain

<table>
<thead>
<tr>
<th>Method</th>
<th>Models</th>
<th>Explanations</th>
<th>Tasks</th>
<th>Data types</th>
<th>Train set req.</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALE Ancestral</td>
<td>BB</td>
<td>global</td>
<td>C, R</td>
<td>tab*</td>
<td>✓</td>
</tr>
<tr>
<td>Anchors</td>
<td>BB</td>
<td>local</td>
<td>C</td>
<td>tab, text, img</td>
<td>For tabular</td>
</tr>
<tr>
<td>CEM</td>
<td>BB*, WB</td>
<td>local</td>
<td>C</td>
<td>tab*, img</td>
<td>Optional</td>
</tr>
<tr>
<td>Counterfactuals</td>
<td>BB*, WB</td>
<td>local</td>
<td>C</td>
<td>tab*, img</td>
<td>No</td>
</tr>
<tr>
<td>Prototype</td>
<td>BB*, WB</td>
<td>local</td>
<td>C</td>
<td>tab, img</td>
<td>Optional</td>
</tr>
<tr>
<td>Counterfactuals</td>
<td>BB*, WB</td>
<td>local</td>
<td>C</td>
<td>tab, img</td>
<td>Optional</td>
</tr>
<tr>
<td>Integrated</td>
<td>WB</td>
<td>local</td>
<td>C, R</td>
<td>tab, text, img</td>
<td>Optional</td>
</tr>
<tr>
<td>Gradients</td>
<td>WB</td>
<td>local</td>
<td>C, R</td>
<td>tab, text, img</td>
<td>Optional</td>
</tr>
<tr>
<td>Kernel SHAP</td>
<td>BB</td>
<td>local, global</td>
<td>C, R</td>
<td>tab</td>
<td>✓</td>
</tr>
<tr>
<td>Tree SHAP</td>
<td>BB</td>
<td>local, global</td>
<td>C, R</td>
<td>tab</td>
<td>Optional</td>
</tr>
</tbody>
</table>

Table 2: Comparison of explanation methods in Alibi. **Models.** Type of model expected. BB: black-box (can call a model), BB*: differentiable black-box, WB: white-box (access to model internals). **Explanations.** local: explain single predictions, global: explain the model overall. **Tasks.** Type of tasks supported. C: classification, R: regression. **Data types.** Type of data supported. tab: tabular, tab*: tabular without categorical variable support, img: images. **Train set req.** Whether a training set is required.

et al. (2018): find a minimal subset of features to guarantee (with high probability) the same prediction regardless of other features. 3. **Contrastive Explanation Method (CEM),** Dhurandhar et al. (2018): find features which should be minimally and sufficiently present as well as features which should be necessarily absent to justify a prediction for a specific instance. 4. **Counterfactual explanations,** Wachter et al. (2018): find synthetic instances close to the original but resulting in a different prediction. 5. **Counterfactual explanations with prototypes,** Van Looveren and Klaise (2019): improves the counterfactual explanation method to result in more interpretable, in-distribution instances. 6. **Integrated Gradients,** Sundararajan et al. (2017): calculate feature attributions to the prediction by accumulating gradients along a path from a baseline instance to the instance of interest. 7. **Kernel Shapley Additive Values,** Lundberg and Lee (2017): calculate feature attributions to the prediction via a game theoretic approach by considering groups of features to be “uninformative”. 8. **Tree Shapley Additive Values,** Lundberg et al. (2020): algorithmic improvement of Shapley additive values to tree ensemble models. Figure 1 shows outputs for a selection of supported explanation algorithms.

3. Library Design

The user facing API of Alibi is designed to be consistent across algorithms and easy to use (Code Snippet 1). An explanation algorithm is initialized by passing either a prediction function (a Python Callable taking and returning numpy arrays) in the black-box case or a pre-trained model (e.g. xgboost for TreeSHAP or TensorFlow) in the white-box case. As detailed in Table 2, for methods where training data is required, the fit method must be called. Finally, an explain method is called to calculate an explanation on an instance or a set of instances. This returns an Explanation object containing dictionaries meta and data with the explanation metadata (e.g. hyperparameter settings, names) and the expla-
Figure 1: A selection of supported explanation algorithms. **Top left:** Anchor explanation on image classification explaining the prediction “Persian cat”. **Top right:** Integrated Gradients attributions on a sentiment prediction task explaining the prediction “positive”. **Bottom left:** Counterfactual explanations of (a) MNIST digit classification and (b) Income classification. **Bottom right:** ALE feature effects for a logistic regression model on the Iris dataset.

The structure of the `Explanation` object enables easy serialization in production systems for further processing (e.g. logging, visualization). The metadata captures settings used to obtain each explanation and acts as an audit trail.

```python
>>> from alibi.explainers import AnchorTabular
>>> explainer = AnchorTabular(predict_fn, feature_names)
>>> explainer.fit(X_train)
>>> explanation = explainer.explain(x)
>>> explanation.meta
{'name': 'AnchorTabular', 'type': ['blackbox'], 'explanations': ['local'], 'params': {'seed': None, 'disc_perc': (25, 50, 75), 'threshold': 0.95}}
>>> explanation.data
{'anchor': ['petal width (cm) > 1.80', 'sepal width (cm) <= 2.80'], 'precision': 0.98, 'coverage': 0.32}
```

Code Snippet 1: Demo of the Alibi API with the AnchorTabular explanation algorithm.

## 4. Outlook

The first phase of the development of Alibi has focused on creating a curated set of reference explanation algorithms with comprehensive guidance on typical use cases. While the work on white-box gradient based methods focused on supporting TensorFlow models (Abadi et al., 2016), achieving feature parity with PyTorch models (Paszke et al., 2019) in the near future is a key goal. Further, we plan to extend the use of the Ray project (Moritz...
et al., 2018) to enable parallelization for all explanation algorithms. The choice of Ray also enables the scaling of explanations beyond a single multi-core computation node (Coca, 2020).

References


