PyXAI : calculer en Python des explications pour des modèles d'apprentissage supervisé

Gilles Audemard, Steve Bellart, Louenas Bounia, Jean-Marie Lagniez, Pierre Marquis, Nicolas Szczepanski

PyXAI : calculer en Python des explications pour des modèles d'apprentissage supervisé

Gilles Audemard *, Steve Bellart *, Louenas Bounia *
Jean-Marie Lagniez *, Pierre Marquis *,**, Nicolas Szczepanski [1] *

*Univ. Artois, CNRS, CRIL, F-62300 Lens
**Institut universitaire de France
nom@cril.fr

Résumé. L'intelligence artificielle explicable est un sous-domaine de l'IA, qui connait un essor important depuis quelques années. Le but poursuivi est de développer des méthodes et outils pour expliquer les résultats produits par des algorithmes d'IA. Consacrée à cette tâche, PyXAI est une librairie Python permettant de calculer des explications pour des prédictions réalisées à partir de plusieurs modèles d'apprentissage supervisé bien connus à base d'arbres : les arbres de décision (*decision trees*), les forêts aléatoires (*random forests*) et les arbres optimisés (*boosted trees*). PyXAI prend en charge deux bibliothèques d'apprentissage automatique : Scikit-Learn et XGBoost. Plusieurs types d'explication peuvent être calculés : *abductive* (pourquoi cette prédiction ?) et *contrastive* (pourquoi pas une autre prédiction ?). Divers types d'explications abductives sont proposés permettant de réaliser différents compromis taille / temps de calcul.

1 Introduction

L'essor de l'intelligence artificielle et de l'apprentissage automatique (ML pour *machine learning*) à travers ses nombreuses applications (diagnostic médical, reconnaissance vocale, conduite autonome, ...) a conduit au développement rapide de l'intelligence artificielle explicable (XAI pour *eXplainable Artificial Intelligence*). En effet, l'apprentissage automatique produit souvent des modèles de type « boîtes noires » et se pose ainsi un problème de confiance envers les décisions prises. Pour pallier ce problème, il importe de développer de nouvelles approches pour garantir confiance et transparence envers les systèmes d'IA exploitant de tels modèles. Cela passe, en particulier, par la possibilité de produire des explications (aussi appelées raisons) pour les décisions obtenues. Dans cette mouvance, l'Union Européenne a introduit un droit à l'explication dans le droit général de la protection des données (RGPD).

Diverses approches de calcul d'explications, indépendantes du modèle d'apprentissage considéré, ont été proposées. Cependant, il a été montré dans Ignatiev (2020) que les approches les plus populaires basées sur les *explications agnostiques* au modèle, telles que LIME (Ribeiro et al., 2016), Anchors (Ribeiro et al., 2018) et SHAP (Lundberg et Lee, 2017), fournissent un

1. Ce travail a été réalisé dans le cadre de la chaire ANR d'enseignement et de recherche EXPEKCTATION (ANR-19-CHIA-0005-01).

très grand nombre d'explications incorrectes (plus de 99% d'explications invalides sur certains jeux de données). Dans un tel cadre, une explication du classement d'une instance est considérée comme incorrecte quand il existe au moins une autre instance pour laquelle l'explication s'applique également, mais pour laquelle le classeur utilisé prédit une classe différente.

Contrairement aux explications agnostiques au modèle, les *explications abductives* sont *spécifiques au modèle*. Une explication abductive pour une instance donnée par un classeur est un sous-ensemble des caractéristiques de l'instance (i.e., des couples attributs / valeurs permettant de décrire l'instance) qui est suffisant pour justifier la façon dont l'instance a été classée. De telles explications correspondent à des impliquants de la fonction associée à la prédiction de la classe de l'instance cible et sont par construction correctes en tout point de l'espace des caractéristiques pris en compte par le modèle. Une *explication contrastive* indique les ajustements des caractéristiques qu'il faut réaliser dans l'instance considérée pour changer la prédiction réalisée à son sujet. Ces explications sont, elles aussi, toujours correctes, elles sont dignes de confiance. Plusieurs travaux récents ont montré que des modèles d'apprentissage automatique de formes variées (incluant des modèles « boîtes noires ») peuvent être associés à des circuits booléens ayant les mêmes comportements en terme d'entrée-sortie. L'intérêt d'utiliser de tels circuits est que les mécanismes mis en oeuvre pour classer sont visibles et permettent donc de comprendre le fonctionnement (de tels circuits peuvent être vus comme des « boîtes blanches (ou transparentes) ») (Marques-Silva et Ignatiev, 2022; Darwiche et Hirth, 2020; Barceló et al., 2020).

L'objectif principal de PyXAI est de donner à la communauté IA un accès simple à diverses méthodes de calcul d'explications.

Ainsi, PyXAI (pour *Python eXplainable AI*) est une librairie en Python (version 3.6 ou plus) permettant de produire des explications de formes variées à partir des modèles produits par les approches de *machines learning*, souvent utilisées pour construire des classeurs à partir de données tabulaires. Plus précisément, PyXAI est le fruit de recherches engagées sur le calcul d'explications lorsque le prédicteur utilisé est un arbre de décision (DT pour *Decision Tree*), une forêt aléatoire (RF pour *Random Forest*) ou encore un arbre optimisé (BT pour *Boosted Tree*) (Audemard et al., 2022b,c,d). Il est important de garder en tête que les requêtes XAI sont, en général, calculatoirement difficiles pour ces modèles (Audemard et al., 2021).

Comme l'illustre la figure 1, l'utilisateur doit être familier avec la notion de modèle : les modèles sont les objets résultant d'un protocole expérimental de ML par le biais d'une méthode de validation simple ou croisée choisie (par exemple, le résultat d'une phase d'entraînement puis de test ayant conduit à produire un arbre de décision). Il est important de noter que dans PyXAI, il y a une séparation complète entre les phases d'apprentissage et celles d'explication via deux modules distincts. Vous produisez/chargez/sauvegardez des modèles avec le module `Learning`, et vous récupérez des explications à partir de ces modèles en utilisant le module `Explainer`. Plusieurs types d'explications pour le classement d'une instance x donnée peuvent être calculés :

- les explications abductives de x sont destinées à expliquer pourquoi x a été classé de la manière dont il a été classé par le modèle ML (répondant à la question « Pourquoi ? »).
- les explications contrastives de x consistent à expliquer pourquoi x n'a pas été classé par le modèle ML comme espéré (répondant à la question « Pourquoi pas ? »).

Travaux connexes. Plusieurs méthodes de calcul d'explications ont été envisagées dans la perspective d'extraire les explications les plus compactes possibles, mais aussi de les calcu-

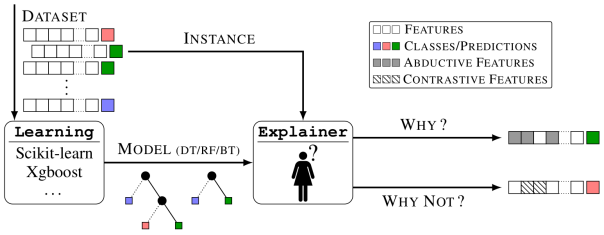

FIG. 1 – *Explication des prédictions individuelles à un utilisateur humain avec PyXAI.*

ler le plus rapidement possible (car le passage à l'échelle des algorithmes reste encore un enjeu, vu la complexité algorithmique des tâches correspondantes). Ainsi, (Choi et al., 2020; Izza et Marques-Silva, 2021; Audemard et al., 2022c) montrent comment dériver des explications abductives pour les forêts aléatoires. Afin d'éviter la présence de caractéristiques inutiles dans les explications, les explications abductives minimales pour l'inclusion (i.e., non redondantes) (alias les raisons suffisantes (Darwiche et Hirth, 2020)) sont souvent ciblées. En ce qui concerne les arbres optimisés, Ignatiev et al. (2019) fournit un schéma d'encodage SMT (*satisfiability modulo theory*) pour calculer des raisons suffisantes. L'outil XAI correspondant est appelé XPlainer (`https://github.com/alexeyignatiev/xplainer`). Ignatiev et al. (2022) présente un autre schéma, basé cette fois sur MaxSAT (*maximum satisfiability*) pour calculer des raisons suffisantes. L'outil associé est appelé XReason (`https://github.com/alexeyignatiev/xreason`).

Comme XPlainer et XReason sont des scripts réalisant chacun une tâche précise, PyXAI est, à notre connaissance, la première librairie calculant des explications abductives et contrastives pour les arbres de décision, les forêts aléatoires et les arbres optimisés. PyXAI possède les caractéristiques suivantes :

- une documentation complète avec plus de 20 exemples Jupyter à télécharger [2] ;
- un Github public [3] et une installation rapide via le gestionnaire de paquets PyPi [4] ;
- la génération (protocole de validation simples ou croisée), construction (via des nœuds et feuilles), sauvegarde/chargement et importation des modèles des librairies XGBoost et Scikit-learn ;
- le calcul d'explications abductives et contrastives ;
- la visualisation des explications pour certains jeux de données graphiques.

2 Préliminaires

On considère un ensemble fini $\{A_1, \ldots, A_n\}$ d'attributs. Un arbre de décision sur $\{A_1, \ldots, A_n\}$ est un arbre binaire T, chacun de ses nœuds internes est étiqueté avec une condition booléenne sur un attribut de $\{A_1, \ldots, A_n\}$, et les feuilles sont étiquetées par des nombres représentant des classes. Les conditions sont généralement de la forme "<id_feature> <operator>

2. `http://www.cril.univ-artois.fr/pyxai/`
3. `https://github.com/crillab/pyxai`
4. `https://pypi.org/project/pyxai/`

<threshold> ?" (comme "$x_4 \geq 0.5$?"). La prédiction $T(x)$ de T pour une instance d'entrée x est donnée par la feuille atteinte à partir de la racine comme suit : à chaque nœud, on va vers l'enfant de gauche ou de droite selon que la condition étiquetée par x est satisfaite ou non.

Soit une fonction booléenne représentée par une forêt aléatoire RF et soit x une instance. RF est un ensemble d'arbres $T_1, \cdots, T_m$, où chaque T_i est un arbre de décision, et tel que la prédiction $RF(x)$ est donnée par : $RF(x) = 1$ si $\frac{1}{m}\sum_{i=1}^{m} T_i(x) > \frac{1}{2}$, 0 sinon.

Contrairement à un arbre de décision ou à une forêt aléatoire (où les feuilles représentent des classes), un arbre de régression est un arbre binaire T où les feuilles représentent des valeurs réelles $w_i \in \mathbb{R}$. Une forêt F associée à une classe j est un ensemble d'arbres $T_1^j, \ldots T_m^j$ où chaque T_i^j ($i \in [m]$) est un arbre de régression tel que le poids $W(F, x) \in \mathbb{R}$ pour une instance d'entrée x est donné par $W(F, x) = \sum_{i=1}^{m} w(T_i^j, x)$ et où le poids $w(T_i^j, x) \in \mathbb{R}$ d'un arbre T_i^j pour une instance x est donné par la feuille atteinte à partir de la racine comme pour les arbres de décision. Le calcul de prédiction d'un arbre optimisé BT est réalisé différemment selon le nombre de classes du jeu de données (*dataset*) :

- dans le cas d'une classification binaire, un arbre optimisé BT est constitué d'une seule forêt $F = \{T_1, \cdots, T_m\}$ et une instance x est considérée comme une instance positive lorsque $W(F, x) > 0$ et une instance négative sinon. Nous notons $BT(x) = 1$ dans le premier cas et $BT(x) = 0$ dans le second ;
- dans un contexte multi-classes à p classes ($p > 2$), un arbre optimisé BT est un ensemble de p forêts $F^1, \ldots, F^p$ où la forêt F^j ($j \in [p]$) est associée à la classe j et une instance x est classée comme un élément de la classe j, noté $BT(x) = j$, si et seulement si $W(F^j, x) > W(F^i, x)$ pour chaque classe i telle $i \neq j$.

En interne, le module `Explainer` fonctionne avec les conditions des nœuds des arbres, qui sont traitées comme des variables booléennes. La représentation binaire d'une instance est un ensemble de variables booléennes correspondant à ces conditions. Chaque variable booléenne représente une condition "<id_feature> <operator> <threshold> ?" du modèle. La représentation binaire se trouve dans la variable `binary_representation` de l'objet `Explainer`. La méthode `to_features` convertit une représentation binaire (ou une raison) en un tuple de conditions "<id_feature> <operator> <threshold> ?".

3 Le module `Learning`

L'exemple suivant montre l'utilisation de PyXAI pour générer des modèles :

```
from PyXAI import Learning

learner = Learning.Xgboost("../dataset/iris.csv")
models = learner.evaluate(method=Learning.K_FOLDS, output=Learning.BT)
for model in models:
    instances_with_predictions =
        learner.get_instances(model, n=10, indexes=Learning.TEST)
    for instance, prediction in instances_with_prediction:
        print("instance:", instance)
        print("prediction", prediction)
```

Le constructeur de classe `Learning.Xgboost` retourne un objet d'apprentissage `Learner` construit à partir du jeu de données passé en paramètre. Ensuite, la méthode

`learner.evaluate` applique un protocole expérimental d'apprentissage automatique (en utilisant Scikit-learn) afin de générer des modèles. Cette méthode (utilisée avec le paramètre nommé `method=Learning.K_FOLDS`) réalise une validation croisée à k blocs ($k = 10$ par défaut) et convertit ensuite les 10 modèles générés dans des formats de données adaptés aux calculs des explications. La méthode `learner.get_instances` permet de sélectionner des instances vérifiant certaines propriétés. Dans l'exemple, 10 instances de l'ensemble test de chaque modèle ont été récupérées. Bien sûr, ces méthodes possèdent de nombreuses autres fonctionnalités à travers l'utilisation de paramètres, comme la possibilité de choisir le nombre de blocs pour la validation croisée ou encore le fait de sélectionner uniquement des instances qui sont correctement classées par les modèles. Nous vous invitons à explorer la documentation pour prendre connaissance des nombreuses autres possibilités du module `Learning` qui sont la construction, la sauvegarde, le chargement et l'importation des modèles.

4 Le module `Explainer`

Le module `Explainer` de PyXAI fournit différentes méthodes pour expliquer les décisions prises par les modèles. Actuellement, il prend en charge les modèles d'arbres de décision (DT), de forêts aléatoires (RF) et d'arbres optimisés (BT).

Raison directe. Soit $F = T_1, \ldots, T_m$ une forêt et x une instance, la raison directe pour x est le terme de la représentation binaire correspondant à l'union des termes correspondant aux chemins de racine à feuille de tous les arbres T_i qui sont compatibles avec x. En raison de sa simplicité, c'est l'une des plus faciles à calculer, mais elle est souvent largement redondante.

```
from PyXAI import Explainer

explainer = Explainer.initialize(model, instance)
direct_reason = explainer.direct_reason()
print("is_reason:", explainer.is_reason(direct_reason))
print("to_features:", explainer.to_features(direct_reason))
```

```
direct: (1, 2, 3, 4)
to_features: ('Petal.Width < 0.75', 'Petal.Length < 4.950000047683716')
```

Le module `Explainer` permet de construire un objet `explainer` capable de calculer des explications pour un modèle et une instance donnée. Ensuite, il suffit d'appeler la méthode `direct_reason` afin de calculer la raison directe pour cette instance. Les raisons calculées par le module `Explainer` sont toujours sous la forme de variables binaires représentant les conditions du modèle. Pour les traduire dans l'espace des attributs considérés initialement, on peut utiliser la méthode `to_features`. Notons que cette méthode élimine les conditions redondantes.

Raison suffisante. Formellement, une explication abductive t pour une instance x donnée par un classeur f (qui est binaire ou non) est un sous-ensemble t des caractéristiques de x tel que t couvre x (noté $t \subseteq t_x$) et toute instance x' partageant cet ensemble t de caractéristiques est classée par f comme l'est x. Une raison suffisante t pour x étant donné f est une explication abductive t pour x étant donné f tel qu'aucun sous-ensemble propre t' de t n'est une explication abductive pour x étant donné f (t est donc minimal pour l'inclusion ensembliste). Enfin,

une raison suffisante minimale pour x est une raison suffisante pour x qui contient un nombre minimal de caractéristiques.

```
sufficient_reason = explainer.sufficient_reason(n=1)
print("to_features:", explainer.to_features(sufficient_reason))
```

```
to_features: ('Petal.Width < 0.75',)
```

PyXAI est capable de calculer des raisons suffisantes pour les arbres de décisions (DT) et les forêts aléatoires (RF) en utilisant des solveurs SAT, MaxSAT et d'extraction de MUS (des travaux sont en cours pour pouvoir les calculer pour les arbres optimisés). Il faut noter que sur certains jeux de données et pour certains modèles ML (en particulier, les forêts aléatoires et les arbres optimisés), calculer une raison suffisante peut se révéler trop difficile en termes de calcul (Audemard et al., 2022b,c,d). Pour pallier ce problème, nous avons introduit les notions d'explication majoritaire et d'explication locale. Ces explications sont, elles aussi, abductives et donc logiquement correctes. Contrairement aux raisons suffisantes, elles peuvent contenir des informations redondantes.

Raison majoritaire. Étant donné une forêt aléatoire, une raison majoritaire pour x est un terme t couvrant x, tel que t est un implicant d'au moins une majorité stricte d'arbres de décision de la forêt et qui est minimal par rapport à l'inclusion ensembliste.

```
majoritary = explainer.majoritary_reason()
minimal_majoritaries = explainer.minimal_majoritary_reason(n=100,
    time_limit=200)
```

PyXAI utilise un algorithme glouton pour calculer une raison majoritaire et un solveur de type MaxXSAT quand il s'agit d'en calculer plusieurs. Comme le montre ce code, vous pouvez demander plusieurs explications avec un temps limite donné en paramètre.

Raison locale. On considère $BT = \{F\}$ un arbre (optimisé) défini sur un ensemble d'attributs $\{A_1, \ldots, A_n\}$ et x une instance. Soit $t \subseteq t_x$.

— Une *pire instance* étendant t étant donné F est une instance x' telle que $t \subseteq t_{x'}$ et $x' = argmin_{x'':t\subseteq t_{x''}}(\{w(F, x'')\})$.

— Une *meilleure instance* étendant t étant donné Fest une instance x' telle que $t \subseteq t_{x'}$ et $x' = argmax_{x'':t\subseteq t_{x''}}(\{w(F, x'')\})$.

$W(t, F)$ (resp. $B(t, F)$) désigne l'ensemble des pires (resp. meilleures) instances couvertes par t étant donné F, et $w_\downarrow(t, F)$ (resp. $w_\uparrow(t, F)$) désigne le poids de la pire (resp. meilleure) instance couverte par t étant donné F. Dans le cas multi-classe, soit $BT = \{F^1, \cdots, F^p\}$ où chaque F^i ($i \in [p]$) contient p_i arbres et soit x une instance telle que $BT(x) = j$. On dit que t est une explication locale pour x étant donné BT si et seulement si t couvre x, pour chaque $i \in [p] \setminus \{j\}$, on a $\sum_{k=1}^{p_j} w_\downarrow(t, T_k^j) > \sum_{k=1}^{p_i} w_\uparrow(t, T_k^i)$, et enfin aucun sous-ensemble propre de t ne satisfait cette dernière condition.

Comme il existe un algorithme simple, en temps linéaire, pour calculer chaque $w_\downarrow(t, T_k^j)$ et chaque $w_\uparrow(t, T_k^i)$, les raisons locales pour x sont beaucoup plus faciles à engendrer que les raisons suffisantes et restent abductives.

```
tree_specific = explainer.tree_specific_reason()
```

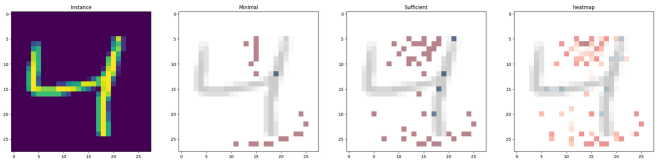

FIG. 2 – *Visualisation d'explications avec PyXAI.*

Raison contrastive. Formellement, une raison contrastive pour x est un sous-ensemble t des caractéristiques de x qui est minimal pour l'inclusion ensembliste parmi ceux qui sont tels qu'au moins une instance x' qui coïncide avec x, sauf sur les caractéristiques de t, n'est pas classée par le modèle dans la même classe que x. Plus simplement, une raison contrastive représente les ajustements des caractéristiques qu'il faut réaliser pour changer la prédiction d'une instance. Une raison contrastive minimale pour x est une raison contrastive pour x qui contient un nombre minimal de littéraux. En d'autres termes, une raison contrastive minimale a une taille minimale.

```
contrastive_reason = explainer.contrastive_reason()
print("contrastive reason:", contrastive_reason)
print("to_features:", explainer.to_features(contrastive_reason))
```

```
contrastive reason: (1,)
to_features: ('Petal.Width >= 0.75',)
```

Nous invitons les lecteurs intéressés à se diriger vers la documentation afin d'obtenir plus d'informations sur PyXAI. La figure 2 fournit un exemple obtenu via l'outil de visualisation de PyXAI.

Pour finir, PyXAI offre aussi la possibilité de prendre en compte des préférences utilisateurs pour pouvoir calculer des explications préférées. Différents types de préférences sont gérés ; par exemple, l'utilisateur peut choisir d'exclure certaines caractéristiques (Audemard et al., 2022a).

5 Perspectives

Des nouvelles fonctionnalités vont voir le jour dans PyXAI. Nous travaillons sur le calcul de raisons suffisantes pour les arbres optimisés. Nous souhaitons inclure de nouvelles bibliothèques générant des modèles comme CatBoost. Nous travaillons également sur d'autres types de modèles d'apprentissage.

Références

Audemard, G., S. Bellart, L. Bounia, F. Koriche, J. Lagniez, et P. Marquis (2021). On the computational intelligibility of Boolean classifiers. In *Proc. of KR'21*, pp. 74–86.

Audemard, G., S. Bellart, L. Bounia, F. Koriche, J.-M. Lagniez, et P. Marquis (2022a). On preferred abductive explanations for decision trees and random forests. In *IJCAI'22*, pp. 643–650.

Audemard, G., S. Bellart, L. Bounia, F. Koriche, J.-M. Lagniez, et P. Marquis (2022b). On the explanatory power of Boolean decision trees. *Data & Knowledge Engineering*.

Audemard, G., S. Bellart, L. Bounia, F. Koriche, J.-M. Lagniez, et P. Marquis (2022c). Trading complexity for sparsity in random forest explanations. In *Proc. of AAAI'22*, pp. 5461–5469.

Audemard, G., J.-M. Lagniez, P. Marquis, et N. Szczepanski (2022d). Computing abductive explanations for boosted trees. *CoRR abs/2209.07740*.

Barceló, P., M. Monet, J. Pérez, et B. Subercaseaux (2020). Model interpretability through the lens of computational complexity. In *Proc. of NeurIPS'20*.

Choi, A., A. Shih, A. Goyanka, et A. Darwiche (2020). On symbolically encoding the behavior of random forests. In *Proc. of FoMLAS'20, Workshop at CAV'20*.

Darwiche, A. et A. Hirth (2020). On the reasons behind decisions. In *Proc. of ECAI'20*, pp. 712–720.

Ignatiev, A. (2020). Towards trustable explainable AI. In *Proc. of IJCAI'20*, pp. 5154–5158.

Ignatiev, A., Y. Izza, P. Stuckey, et J. Marques-Silva (2022). Using MaxSAT for efficient explanations of tree ensembles. In *Proc. of AAAI'22*, pp. 3776–3785.

Ignatiev, A., N. Narodytska, et J. Marques-Silva (2019). On validating, repairing and refining heuristic ML explanations. *CoRR abs/1907.02509*.

Izza, Y. et J. Marques-Silva (2021). On explaining random forests with SAT. In *Proc. of IJCAI'21*, pp. 2584–2591.

Lundberg, S. et S.-I. Lee (2017). A unified approach to interpreting model predictions. In *Proc. of NIPS'17*, pp. 4765–4774.

Marques-Silva, J. et A. Ignatiev (2022). Delivering trustworthy AI through formal XAI. In *Proc. of AAAI'22*, pp. 12342–12350.

Ribeiro, M. T., S. Singh, et C. Guestrin (2016). "Why should I trust you ?" : Explaining the predictions of any classifier. In *Proc. of KDD'16*, pp. 1135–1144.

Ribeiro, M. T., S. Singh, et C. Guestrin (2018). Anchors : High-precision model-agnostic explanations. In *Proc. of AAAI'18*, pp. 1527–1535.

Summary

EXplainable Artificial Intelligence is a subfield of AI, which has experienced significant growth in recent years. The aim is to develop methods and tools to explain the results produced by AI algorithms, in particular predictors built from data using machine learning approaches. Dedicated to this task, PyXAI is a Python library allowing to compute explanations for predictions made from several well-known tree-based supervised learning models: decision trees, random forests, and boosted trees). PyXAI supports wo popular machine learning libraries: Scikit-Learn and XGBoost. Several types of explanation can be calculated: *abductive* (why this prediction?) and *contrastive* (why not another prediction?). Various classes of abductive explanations are proposed to the user allowing to achieve different compromises in terms of size / computation time.