Documentation sur les comparateurs de librairies d'explicabilité des réseaux de neurones

Marina CHAZAL
Adam WANG

Table des matières

Introduction

Nous étudions ici la comparaison de plusieurs librairies d'explicabilité de prédiction de modèles dans le cas des classifications.

Qu'est ce qu'un algorithme d'explicabilité ?

Une manière très répandue pour prédire la valeur d'un individu est d'utiliser un échantillon d'apprentissage pour entraîner un modèle (arbre, régression logistique, réseaux de neurones, etc...) puis d'utiliser ce modèle pour prédire la valeur de l'individu. Cependant, de nombreux types de modèles (réseaux de neurones, random forest, etc...) ne nous permettent pas de savoir pourquoi ils ont prédit telle ou telle valeur pour un individu donné et quelles ont été les variables les plus déterminantes à l'obtention de la prédiction. Les algorithmes d'explicabilité nous permettent donc de savoir pourquoi le modèle à prédit telle ou telle variable.

Des tutoriels de ces librairies sont disponibles sur internet.

Il faut aussi faire attention à n'entrer dans les librairies d'explicabilité et par la suite dans le comparateur, seulement des données numériques. Ainsi, s'il y a des données non numériques, il faut les encoder.

Rapide présentation des 4 librairies d'explicabilité étudiées

Les librairies BreakDown, SHAP et LIME effectuent leurs explications en prenant comme hypothèse que le modèle pris en paramètre est un modèle additif (additif = variables à expliquer indépendantes les unes des autres) contrairement à la librairie iBreakDown qui va supposer que des interactions entre les variables à expliquer sont possibles et les prendre en compte dans son algorithme.

L'algorithme BreakDown va utiliser une approche greedy. Ainsi, il va parcourir toutes les variables afin de trouver la variable pour laquelle la prédiction varie le moins possible lorsqu'on retire cette variable du modèle puis retirer cette variable du modèle. Il refait ensuite de même avec les variables restantes et ainsi de suite.

L'algorithme iBreakDown est une amélioration de BreakDown sauf qu'il regarde aussi les groupes de variables.

L'algorithme SHAP se base sur la théorie des jeux pour trouver les variables les plus explicatives pour un individu.

L'algorithme LIME génère des instances autour d'un invidu puis fait tourner un modèle simple à comprendre, par exemple une régression linéaire, sur ces nouvelles observations afin d'exploiter ses paramètres.

Le comparateur

Ce comparateur permet de déterminer quelle librairie d'explicabilité est la plus efficace. Dans le code fourni, il s'utilise avec les librairies BreakDown, iBreakDown, SHAP et LIME. L'algorithme fait appel à des fonctions nommées « évaluateur » qui sont propres à chaque librairie d'explicabilité mais qui ont toutes pour but de déterminer un taux d'erreur d'explicabilité de leur librairie. Nous avons donc dans notre cas 4 librairies évaluateur nommées « eval_bd» (BreakDown), «eval_ibd » (iBreakDown), « eval_lime » (LIME) et « eval_shap » (SHAP). Si l'on souhaitait changer les librairies étudiées, cela nécessiterait de créer les fonctions « évaluateur » pour les nouvelles librairies et de changer le nom des fonctions appelées au sein du code.

Présentation des évaluateurs

Un évaluateur permet de déterminer le taux d'erreur d'explicabilité de sa librairie pour un jeu de données et un modèle donnés. Il prendra donc en paramètres un vecteur à 2 dimensions nommé « data » contenant toutes les colonnes du jeux de données des variables explicatives, un vecteur à 1 dimension nommé « y » contenant la colonne du dataset de la variable à expliquer, un vecteur nommé « feature_names » contenant le nom des variables à expliquer, le nombre de variables « trust_worthy » désiré (cette notion sera expliquée plus loin) ainsi que le nom du modèle entre guillemets (attention, tous les modèles n'existent pas dans ce code, il faut se référer à la liste des modèles jointe en annexe).

Pseudo-code de l'évaluateur

Fonction eval (data, y, feature_names, nb_tw, modele) :

Création à partir de data et y d'un dataset d'entraînement composé de train_data (variables explicatives) et train_labels (variable à expliquer) et d'un dataset de test composé de test_data et test_labels.

Entraînement du modèle (modèle1) à l'aide du dataset d'entraînement.

Création d'un explicateur (qui servira tout au long du code) à l'aide des données d'entraînement.

nb_diff = 0 (nombre de mauvaises prédictions)

Pour i allant de 0 au nombre d'individus test-1 :

Création de l'explication du i ème individu de test_data.

Prédiction (prédiction1) de la valeur de la variable à expliquer de l'individu à l'aide du modèle modèle1 entraîné.

Création d'un dataset possédant les nb_tw variables à expliquer les plus explicatives pour la prédiction de cet individu (les variables en question sont récupérées dans l'explication créée pour l'individu).

Création d'un dataset d'entraînement et d'un dataset de test à partir de ce nouveau dataset (train_data2, test_data2, train_labels, test_labels).

Entraînement d’un nouveau modèle (modèle2) à partir du dataset d’entraînement venant d’être créé (train_data2 et train_labels).

Prédiction (prédiction2) de la valeur de la variable à expliquer de l’individu à l’aide du modèle modèle2.

Si prédiction1 et prédiction2 sont différentes incrémenter de 1 la valeur de nb_diff.

Fin Pour

Retourner nb_diff / nombre d’individu test (cela permet d’obtenir un taux d’erreur).

Fin Fonction

Illustration du pseudo-code de l’évaluateur

Dataset :

	Var1	Var2	Var3	Y
Ind1	1	13	1.5	1
Ind2	2.5	46	1.5	0
Ind3	15	29	3	0
Ind4	3	35	2.5	1
Ind5	6	10	0.5	0

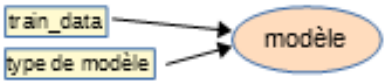
train_data

test_data

data

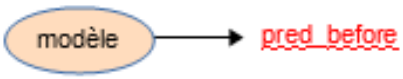
y

Création du premier modèle à partir de train_data :

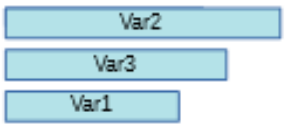


Pour chaque individu de test_data :

Prédiction de l'individu à partir du modèle :



Explication de l'individu :



Dans notre exemple, on suppose que le nombre de variables trust-worthy entré en paramètre est 2. On ne souhaite donc garder que les 2 variables les plus contributives qui sont ici Var2 et Var3

Donc dataset2 :

	Var2	Var3	Y
Ind1	13	1.5	1
Ind2	46	1.5	0
Ind3	29	3	0
Ind4	35	2.5	1
Ind5	10	0.5	0

train_data2

test_data2

Création du deuxième modèle à partir de train_data2 puis prédiction de l'individu à partir de ce modèle :

Si pred_before est différent de pred_after, on augmente de 1 le nombre de mauvaises prédictions.

Fin Pour

On divise le nombre le nombre de mauvaises prédictions par le nombre d'individus test afin d'obtenir un taux d'erreur qu'on retourne.

Différences entre les différents évaluateurs

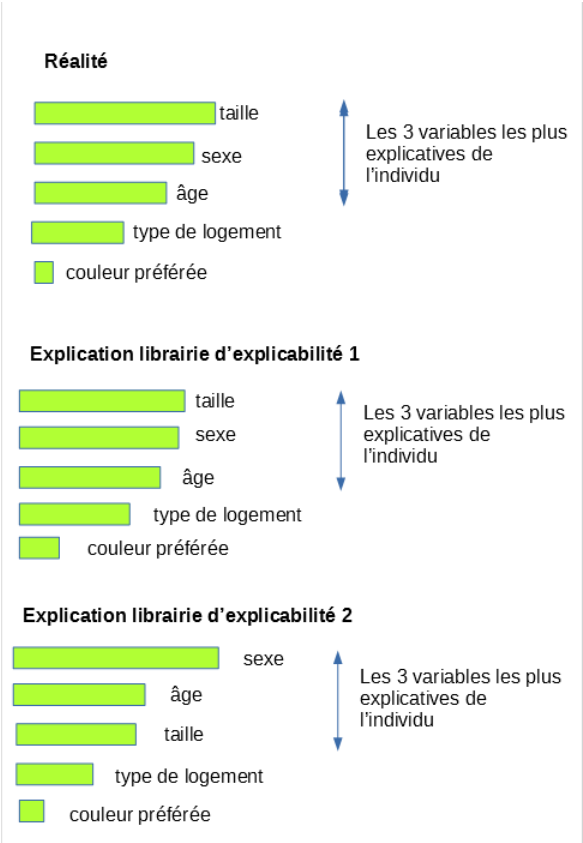
Les évaluateurs des différentes librairies d'explicabilité ont tous des codes qui correspondent au pseudo-code ci-dessus. Cependant, leurs codes diffèrent au niveau de la récupération des variables « trust-worthy » car chaque librairie a ses propres attributs et ils ne se récupèrent pas de la même manière selon les librairies. De plus, la librairie iBreakDown, contrairement aux autres librairies, ne considère pas les modèles comme additifs. Elle peut donc renvoyer comme variables explicatives des tuples de variables (contrairement aux autres librairies qui renvoient les variables explicatives sous forme de singleton), ce qui complique encore la récupération des variables les plus explicatives pour un individu.

Explication du code du comparateur

Le comparateur permet de renvoyer une liste contenant quatre taux d'erreurs (un pour chaque librairie). Il prend en paramètres un vecteur à 2 dimensions nommé « data » contenant toutes les colonnes du jeux de données des variables explicatives, un vecteur à 1 dimension nommé « y » contenant la colonne du dataset de la variable à expliquer, un vecteur nommé « feature_names » contenant le nom des variables à expliquer et le nom du modèle désiré entre guillemets.

Le but de cet algorithme est de s'appuyer sur les fonctions évaluateurs pour pouvoir calculer un taux d'erreur, associé à chaque librairie, plus proche de la réalité que le taux d'erreur calculé pour l'évaluateur. En effet, pour l'évaluateur, on calcule le taux d'erreur à partir d'un nombre nb_tw de variables trust-worthy. Ainsi, deux librairies donnant les mêmes nb_tw premières variables explicatives mais pas dans le même ordre auront le même taux d'erreur alors qu'une des explications sera plus proche de la réalité que l'autre.

Par exemple, si l'on cherche à expliquer la variable poids d'un individu en fonction des variables sexe, âge, taille, type de logement et couleur préférée, les résultats de 2 librairies différentes pourraient être :



Les évaluateurs des librairies 1 et 2 auraient donc pour cette individu tous les 2 la même prédiction finale car dans leur algorithme, ils construiraient tous les 2 leur deuxième modèle avec les variable taille, sexe et âge. Or, on remarque ici que l'explication de la librairie 1 correspond beaucoup mieux à la réalité que l'explication de la librairie 2 et que ces deux explications sont quand même très différentes. Elles ne devraient donc pas forcément avoir la même prédiction.

Pour pallier ce problème, nous avons construit un comparateur qui ne calcule pas juste un score mais qui en calcule plusieurs. En effet, le comparateur va calculer les scores de chaque librairie à l'aide des fonctions évaluateur pour un nombre de variables trust-worthy allant de 1 au nombre de variables explicatives-1 (des poids pourront peut-être être mis par la suite pour rendre le comparateur plus précis) puis faire la moyenne de ces scores. Ainsi, l'ordre des variables sera pris en compte. En effet, lors du calcul du taux d'erreur avec une seule variable trust-worthy, l'ordre de la première variable aura de l'importance. Ensuite, lors du calcul du taux d'erreur avec deux variables trust-worthy, l'ordre des 2 premières variables aura de l'importance, etc …

Pseudo-code du comparateur

```
def comparateur (data, y, feature_names, modele) :
        mauvaises_pred_bd = 0
        mauvaises_pred_ibd = 0
        mauvaises_pred_lime = 0
        mauvaises_pred_shap = 0
         #Faire tourner les algos pour un nombre de variables trustworthy alors de 1 à n_f-1 (n_f nombre
        #total de variables à expliquer) et sommer les taux retournés par les algos
        Pour i allant de 1 au nombre de variables-1 :
                mauvaises_pred_bd = mauvaises_pred_bd + résultat taux d'erreur évaluateur BreakDown avec les
                paramètres (data, y, feature_names, i, modele)
                mauvaises_pred_ibd = mauvaises_pred_ibd + résultat taux d'erreur évaluateur iBreakDown avec les
                paramètres (data, y, feature_names, i, modele)
                mauvaises_pred_lime = mauvaises_pred_lime + résultat taux d'erreur évaluateur SHAP avec les
                paramètres (data, y, feature_names, i, modele)
                mauvaises_pred_shap = mauvaises_pred_shap + résultat taux d'erreur évaluateur LIME avec les
                paramètres (data, y, feature_names, i, modele)
        Fin Pour

        #diviser la somme pour obtenir un taux moyen pour chaque algo pour ces données et ce modèle
        mauvaises_pred_bd = mauvaises_pred_bd/(n_f-1)
        mauvaises_pred_ibd = mauvaises_pred_ibd/(n_f-1)
        mauvaises_pred_lime = mauvaises_pred_lime/(n_f-1)
        mauvaises_pred_shap = mauvaises_pred_shap/(n_f-1)

        Retourner sous forme de liste les 4 mauvaises_pred
```

Annexes

Liste des modèles pouvant être appelés dans un évaluateur :

« Arbre », « SVM », « Random Forest », « KNN », « Logistic Regression », « NN » (bientôt).