Defi-IA 2022

Prévision des précipitations journalières cumulées

Salamata Ba, Adja Deguene Gueye, Younes Radi, Jiawen Wu

January 7, 2022

1 Introduction

Le défi IA est une compétition entre étudiants issus de différentes universités et écoles françaises. La première édition a été organisée en 2016 et depuis, chaque année, une nouvelle problématique autour de l'IA est proposée. La prévision météo est une tâche assez compliquée à cause de la grande quantité d'informations et de mesures que les ingénieurs météorologues doivent prendre en considération pour synthétiser et prédire avec précision la météo. Pour aider à améliorer cette précision, une première approche basée sur des techniques de machine learning (Random forest) a été mise en place et a donné de bons résultats. Pour continuer l'amélioration des prévisions météorologiques, Météo France a participé à l'organisation de la sixième édition du défi IA 2022 afin de donner l'occasion aux data scientists, actuels et en devenir, de tester de nouveaux algorithmes de séries temporelles et/ou de machine learning dans le but de prédire les précipitations journalières cumulées sur différentes stations d'observations.

2 Analyse exploratoire, Preprocessing et Feature Engineering

2.1 Exploration des données et création de variables

Nous avons à notre disposition différentes sources de données: mesures effectuées par heure au niveau des stations sol à savoir la direction du vent `dd`, la vitesse du vent `ff`, la température `t`, l'humidité `hu`, la température du point de rosée `td` et la précipitation `precip` ainsi que les prévisions météorologiques ou forecasts de la pluviométrie par jour pour ces stations.
A partir de ces données, nous avons étudié la distribution des variables, la corrélation entre elles ainsi que le pourcentage de données manquantes dans la base.

Comme on peut le voir sur la figure 1 ci-dessous, la part de données manquantes est assez élevée sur l'échantillon d'apprentissage. Il en est de même pour l'échantillon test. Il est alors nécessaire de trouver un moyen d'imputer ces valeurs manquantes sur les mesures des paramètres météorologiques mais aussi sur les forecasts (variable `Prediction`) effectués par les modèles déterministes de Météo France comme AROME ou ARPEGE.

D'autre part, en regardant la distribution de la variable précipitation à prédire (figure 2), on remarque que la majorité des valeurs prises sont proches de 0, ce qui nous suggère de tester lors de la modélisation une transformation de cette variable en prenant le logarithme de 1 plus sa valeur (on ajoute 1 pour n'avoir que des valeurs strictement positives).

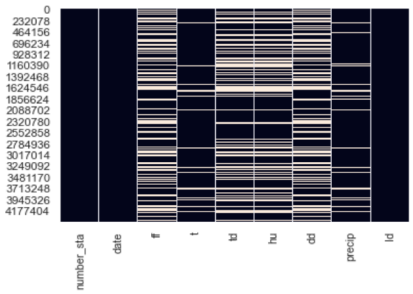

Figure 1: Part de Données manquantes

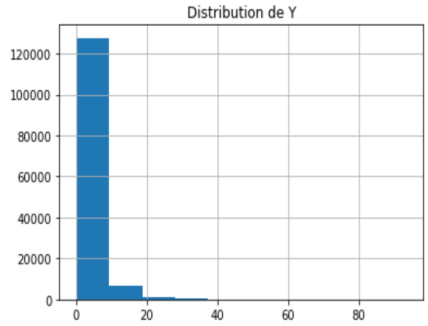

Figure 2: Distribution de la variable précipitation

Notre objectif est de prédire la quantité de pluie par jour, or nous disposons au départ pour certaines variables (t, td, dd, hu, ff et precip) de mesures par heure. Nous avons donc commencé par construire les moyennes de ces mesures par jour et par station excepté pour la variable precip où l'on calcule la somme par jour en mettant une valeur NaN si une des mesures par heure est manquante pour le jour considéré. À noter que pour le jeu de données test, la station et la date n'étaient pas explicitement disponible, mais on a utilisé la colonne Id pour retrouver la station et l'indice du jour avant de faire un regroupement par rapport à ces variables pour calculer les moyennes.

D'autre part, la saison pouvant être une information importante, on décide de créer une variable season.

2.2 Imputation des données manquantes

La problématique des données manquantes est assez conséquente sur ce projet. On décide de faire une interpolation pour remplir les NaNs. Le module `geopy` de python permet de calculer la distance entre les stations en utilisant leurs coordonnées géographiques (longitudes et latitudes) qui sont aussi procurées sur le site Kaggle du défi. À partir de cette métrique, on construit deux grandes matrices contenant les distances par paires entre les stations présentes dans le jeu de données train d'une part et dans le dataset test de l'autre.

Le remplissage a été fait différemment dans le train et le test. Ceci est dû au fait que dans le jeu de données test, on n'admet pas de valeurs manquantes qui subsistent, on est obligé de toutes les remplir, alors que pour le dataset train, on a pensé qu'il était important de laisser les données d'apprentissage assez propres et de ne pas risquer d'introduire beaucoup de biais en essayant de remplacer coûte que coûte les NaNs.

Imputation des NaNs dans le train set

À partir de la matrice des distances construire, on peut aisément trouver une liste ordonnée des cinq stations les plus proches d'une station S. Si une donnée est manquante pour la station S le jour J, on la remplace par la valeur de la station la plus proche de S le même jour J. Si la valeur n'est également pas disponible, on regarde pour la deuxième station la proche le jour J et ainsi de suite. On laisse la valeur à NaN si on ne trouve pas de valeur disponible sur un rayon de cinq stations pour le jour considéré.

Avec cette méthode, on arrive à imputer une large partie des données manquantes sur le train set. La figure 2.2 montre les pourcentages de NaNs avant et après l'imputation.

	column_name	percent_missing_before	percent_missing_after
0	number_sta	0.00	0.00
1	date	0.00	0.00
2	td	32.13	18.12
3	hu	32.07	10.52
4	ff	39.48	18.12
5	t	4.92	1.28
6	dd	39.50	10.54

Imputation des NaNs dans le test set:

Ici pour remplir toutes les valeurs manquantes, on effectue un clustering sur nos stations. On utilise une classification ascendante hierarchique pour définir des clusters de stations proches. L'algorithme se sert de la matrice de distance calculée au préalable pour déterminer la distance entre deux stations et la métrique de Ward pour définir la distance entre des clusters de stations (critère permettant de minimiser la variance interclasse). Le dendrogramme obtenu (figure 3) est coupé en $h = 1500$ pour former au total une trentaine de clusters.

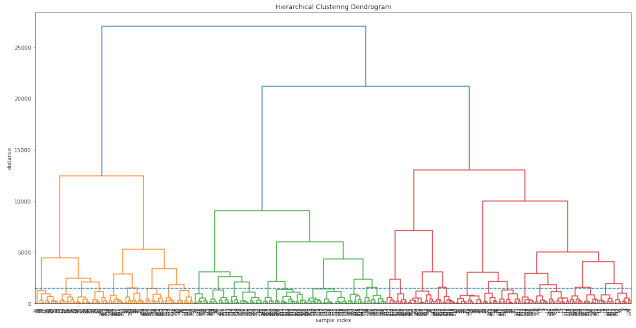

Figure 3: Clustering sur les stations du test set

Une fois les clusters formés, on calcule, pour chacune des variables dd, ff, t, td, hu et Prediction, la moyenne sur chaque cluster pour chaque indice de jour présent dans la colonne Id.

On procède à l'imputation de la manière suivante: si une valeur est manquante pour une station S le jour J, on la remplace par la moyenne de la variable le jour J sur le cluster auquel appartient S. Cependant, il peut arriver que la moyenne d'une variable sur un cluster soit égale à NaN pour un jour J notamment si les mesures de ce jour ne sont présentes sur toutes les stations de ce cluster. Dans ce cas, on regarde la moyenne du cluster de S pour la date la plus récente à partir de ce jour (Jours J-1, J+1, J-2, J+2, ect) jusqu'à trouver une valeur. On aurait pu également chercher la moyenne sur les clusters les plus proches pour le jour J ou combiner les deux méthodes.

2.3 Choix des variables pour l'entraînement

Au vu de l'analyse exploratoire, nous avons décidé de tester et de comparer nos modèles sur quatre datasets différents selon les variables qu'on considère. De plus, on applique la transformation logarithmique sur le Ytrain à chaque fois pour voir si on note une amélioration du score.

Dataset 1: Pour prédire la quantité de pluie, on ne considère dans un premier temps que les variables météorologiques t, td, hu, dd et ff.
Dataset 2: On rajoute les forecasts de Météo France à ces variables t, td, hu, dd et ff.
Dataset 3: En plus des variables citées précédemment, on utilise la saison ainsi que les coordonnées géographiques des stations à savoir longitude, latituge et altitude.
Dataset 4: En affichant le graphe de corrélation des variables, on a trouvé que les variables température t et température point de rosée td sont fortement corrélées. On décide alors de faire un modèle où l'on supprime l'un des deux (td).

3 Modèles

3.1 Evaluation des modèles: score choisi

Pour évaluer les performances du modèle, on calcule le "Mean Absolute Percentage Error" défini comme suit:

$$\text{MAPE} = \frac{100\%}{n} \sum_{t=1}^{n} \left| \frac{A_t - F_t}{A_t} \right|$$

A_t représente les vraies valeurs de précitations et F_t les prévisions obtenues par le modèle. Comme la quantité de pluie peut valoir 0, on ajoute 1 à A_t et à F_t lors du calcul.

3.2 Premiers modèles classiques de Machine Learning

Dans un premier temps, on a testé nos modèles sur les datasets 1,2,3 et 4 décrits à la section 2.3. Le tableau 4 récapitule les résultats obtenus. On voit bien que pour ces modèles, la transformation logarithmique du Ytrain permet d'améliorer très fortement le score. On a utilisé la validation croisée pour trouver les paramètres optimaux (max_depth ou nombre de couches cachées par exemple).

Modèles	Dataset1	Dataset1 avec log	Dataset2	Dataset2 avec log	Dataset3	Dataset3 avec log	Dataset4	Dataset4 avec log
Régression linéaire	103.16	51.72	102.29	51.77	104.90	52.498	105.41	52.06
Arbre de décision (max_depth=9)	78.00	49.64	74.48	48.30	74.84	49.06	72.89	47.55
Forêts aléatoires (nb_estimators=10)	86.16	55.19	83.11	52.84	84.07	52.84	83.57	53.66
Perceptrons multicouches (5 couches cachées)	82.35	48.08	74.46	48.20	83.65	47.47	82.86	47.65

Figure 4: Tableau comparatif MAPE de quelques modèles

3.3 ARIMA

Auto Regressive Integrated Moving Average (ARIMA) est un modèle de machine learning utilisé pour faire la prévision de séries temporelles. Une des approches que l'on considère pour prédire les précipitations est celle série temporelle. On considère comme variable d'entrée uniquement les précipitations journalières observées par station. Le modèle est de la forme suivante:

$$X_t = \sum_{i=0}^{p}(\alpha_i X_{t-i}) + \sum_{i=0}^{q}(\beta_i \varepsilon_{t-i})$$

X_t: les précipations d'une station le jour t
ε_i: les termes d'erreurs
α_i, β_i: les paramètres du modèle

Pour pouvoir appliquer ARIMA, il faut d'abord vérifier la stationnarité des données. On applique le test de Dickey-Fuller qui permet de tester la stationnarité d'une série temporelle. On a une p-valeur=0.0005 qui est inférieur à 0.05. On rejette alors l'hypothèse nulle qui suppose que la série comporte une racine unitaire et on garde l'hypothèse alternative où on suppose que les données sont stationnaires.

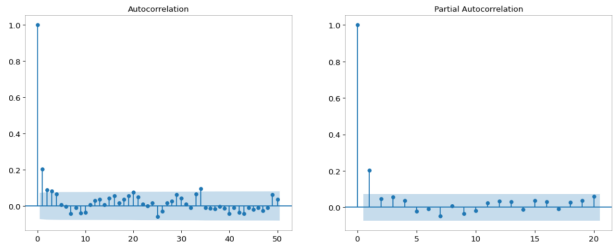

Figure 5: Graphe d'autocorrélation et d'autocorrélation partielle

Le graphe d'autocorrélation partielle permet de décider du nombre de variables passées p (lags) à considérer et le graphe d'autocorrélation permet de choisir le nombre d'erreurs passées q à considérer dans le modèle. Sur les deux graphes, l'intervalle bleu représente les variables non pertinentes, donc on considère que les variables en dehors de cet intervalle. On voit que seulement la variable et l'erreur à l'instant précédant Y_{t-1} et ε_{t-1} sont pertinentes. Comme les données sont stationnaires, on ne va pas les différentier. On va alors choisir un modèle ARIMA (1,0,1). Une deuxième approche pour décider les paramètres du modèle est de minimiser le critere AIC. En utilisant la fonction auto-plot, on a également un modèle ARIMA (1,0,1) avec un AIC= 4334.04 .

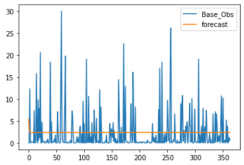

Figure 6: Courbe des valeurs prédites par ARIMA

Après avoir prédit les précipitations pour une durée de 363 jours, on a un MAPE= 154.04 qui n'est pas du tout satisfaisant.

3.4 Deep learning pour la prédiction de séries temporelles

Les modèles de Deep learning sont souvent assez robustes face aux données manquantes et permettent de trouver des variables cachées afin de résoudre au mieux le problème de prédiction. Cependant, cela fait qu'ils restent moins interprétables que d'autres modèles. Ces modèles offrent aussi une solution pour ignorer les données manquantes avec la couche "Masking" de Keras. Nous travaillons ici avec les données d'apprentissage pour lesquelles quelques données manquantes ont déjà été imputées. Nous appliquons ensuite une régression avec soit des réseaux de neurones récurrents (RNN), soit des réseaux de neurones convolutionnels (CNN).

Méthodologie

Nous utilisons ici une méthode permettant de prédire des séries multiples en parallèle. L'idée est de regrouper l'ensemble des données de pluviométrie dans une grande matrice. Comme nous avons au total 325 stations, 730 jours de données dans notre échantillon d'apprentissage et 363 jours de données dans notre échantillon de test, nous construisons une matrice 1093x325. Chaque colonne i correspond donc à l'ensemble des données de pluviométrie pour la i-ème station dans l'ordre chronologique. Nous effectuons ensuite la transformation logarithmique de nos données afin d'accorder plus d'importance aux petites valeurs comme expliqué dans la partie 2.1.

Nous voulons entrainer notre modèle à prédire une valeur pour chacune de nos colonnes. Pour cela, il nous faut définir des échantillons d'entrée d'une certaines taille et des échantillons de sortie. Voici un exemple afin de mieux comprendre le procédé: Si nous avons 2 stations en entrée avec les valeurs $[1.3, 0.1, 0.4, 0.9, 1.12, 0, 1.1, 0.7, 1.1, 0.7]$ et $[1.5, 0., 0.8, 1.1, 1.2, 0.4, 1.1, 0.9, 0.8, 0.7]$, nous allons par exemple prédire la sixième valeur avec les cinq premières puis la septième avec les cinq suivantes. On aura donc:

```
X: [[1.3  1.5 ]   X: [[0.1  0.  ]   X: [[0.4  0.8 ]
 [0.1  0.  ]       [0.4  0.8 ]       [0.9  1.1 ]
 [0.4  0.8 ]       [0.9  1.1 ]       [1.12 1.2 ]
 [0.9  1.1 ]       [1.12 1.2 ]       [0.   0.4 ]
 [1.12 1.2 ]]      [0.   0.4 ]]      [1.1  1.1 ]]
y: [[0.  0.4]]    y: [[1.1 1.1]]    y: [[0.7 0.9]]
```

Figure 7: Les 3 premiers échantillons obtenus avec notre exemple

Dans notre cas, nous avons remarqué que prédire la valeur suivante avec les 60 valeurs précédentes donnait de bons résultats. A partir de ces échantillons-là, nous constituons les échantillons de train et de test: X_{train} et y_{train} seront composés des 730 premiers échantillons de X et y tandis que X_{test} et y_{test}. seront composés des 363 derniers.

Les RNN-LSTM

Comme réseau de neurones récurrent, nous utilisons les LSTM qui permettent de garder une mémoire artificielle de nos précédentes entrées. Ce réseau permet de prévoir la sortie dans un con-

texte particulier . Les LSTM bidirectionnels permettent en plus de garder une mémoire dans les deux sens: du passé vers le futur et du futur vers le passé. Leur temps de calcul en est donc rendu plus long. Voici le réseau que nous avons implémenté:

Layer (type)	Output Shape	Param #
masking (Masking)	(None, 60, 325)	0
bidirectional (Bidirectional)	(None, 400)	841600
repeat_vector (RepeatVector)	(None, 1, 400)	0
bidirectional_1 (Bidirectional)	(None, 1, 200)	400800
dropout (Dropout)	(None, 1, 200)	0
time_distributed (TimeDistributed)	(None, 1, 325)	65325

Total params: 1,307,725
Trainable params: 1,307,725
Non-trainable params: 0

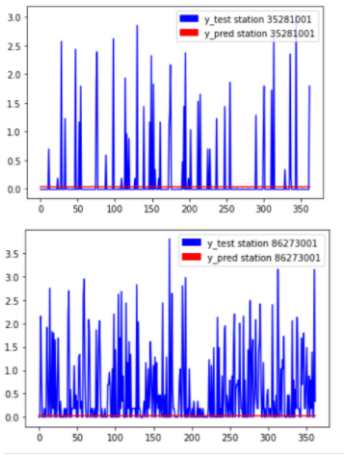

Figure 8: Bi-LSTM: Réseau implémenté et pluviométrie prédite sur 2 stations (en rouge)

Ce réseau donne d'assez bons résultats en terme de score:. Cependant, comme nous pouvons le voir, les valeurs obtenues sont quasiment constantes et ne suivent pas les fluctuations de la série. Le score obtenu n'est pas stable non plus: il varie entre 30 et 60 %.

Les CNN

Les CNN permettent de réduire le temps de calcul mais aussi d'apprendre la structure des données d'entrée afin d'avoir de meilleurs résultats visuellement. Nous pouvons remarquer dans la figure 9 qu'il y a moins de paramètre que dans le réseau précédent.

Layer (type)	Output Shape	Param #
masking_1 (Masking)	(None, 60, 325)	0
conv1d_2 (Conv1D)	(None, 59, 64)	41664
conv1d_3 (Conv1D)	(None, 58, 32)	4128
max_pooling1d_1 (MaxPooling1D)	(None, 29, 32)	0
flatten_1 (Flatten)	(None, 928)	0
dense_2 (Dense)	(None, 600)	557400
dropout_1 (Dropout)	(None, 600)	0
dense_3 (Dense)	(None, 325)	195325

Total params: 798,517
Trainable params: 798,517
Non-trainable params: 0

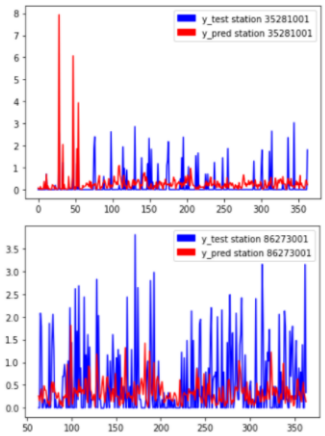

Figure 9: CNN: Réseau implémenté et pluviométrie prédite sur 2 stations (en rouge)

Nous pouvons voir que malgré des prédictions assez élevées pour les 5 premiers jours, le réseaux arrive à capter la structure des données. Nous obtenons ici un score aux alentours de 40 %.

3.5 XGBoost

Afin de personnaliser la fonction de perte de `XGBoost` dans `sklearn`, nous devons donner son gradient et sa hessienne.
Pour éviter d'avoir des 0 au dénominateur du MAPE quand l'algorithme prédit qu'il ne pleut pas, on prendra le dénominateur égal à la précipitation +1. Il n'est cependant toujours pas possible de calculer sa dérivée. Pour plus de rigueur, on choisit donc d'approximer le MAPE par des fonctions dérivables. On introduit la Huber loss qui est définie comme suit :

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{si} |y - f(x)| \leq \delta, \\ \delta\left(|y - f(x)| - \frac{1}{2}\delta\right) & \text{sinon.} \end{cases}$$

Lorsque nous choisissons $\delta = 1$, la trajectoire de Huber est représentée par la ligne rose de la figure 10, et nous sommes en fait proche de la courbe de la valeur absolue. La pente dans la partie non nulle est également relativement similaire. Sur cette base, si l'on veut convertir cette approche de MAE en MAPE, il suffit de diviser sa valeur par A_t(+1 dans notre cas).

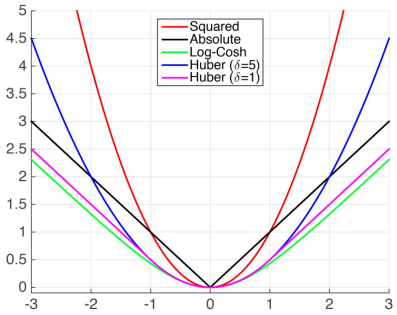

Figure 10: L'approche par Huber loss

Si nous voulons être plus méticuleux dans les mathématiques, nous prenons en fait la dérivée de la fonction de perte Pseudo-Huber. Il est également possible de choisir d'approximer la fonction de perte en utilisant ln(cosh). Nous pouvons voir dans la figure 10 que c'est aussi une approximation proche.

Avec XGBoost, nous avons obtenu un bon score de 31.417. Mais nous obtenons une prédiction de $0,0212$ pour tous les 85140 lignes.

3.6 Réseau de neurones

Avec les ajustements appropriés, nous avons obtenu un réseau neuronal assez performant avec une structure relativement simple comme ci-dessus (figure 11). Avec de multiples tests, un nombre plus raisonnable d'epochs est d'environ 15, et nous avons été en mesure d'atteindre une performance optimale sur l'ensemble du dataset de test. Le réseau ne surapprend pas non plus.

Il s'agit de notre meilleur modèle, qui obtient parfois un score MAPE de 31 ou moins sur l'ensemble du dataset de test. Lorsque nous examinons les résultats des prédictions, nous constatons que le réseau donne généralement des valeurs de prédiction très faibles. Nous supposons que c'est pour obtenir un meilleur score MAPE. En même temps, il donne parfois des résultats négatifs, et cela ne doit pas être la solution la plus adaptée. Après avoir corrigé tous les résultats pour qu'ils soient positifs, nous avons obtenu nos **résultats optimaux**.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1           [-1, 136000, 32]             224
              Tanh-2           [-1, 136000, 32]               0
            Linear-3           [-1, 136000, 16]             528
              Tanh-4           [-1, 136000, 16]               0
            Linear-5            [-1, 136000, 1]              17
================================================================
Total params: 769
Trainable params: 769
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 3.11
Forward/backward pass size (MB): 100.65
Params size (MB): 0.00
Estimated Total Size (MB): 103.76
----------------------------------------------------------------
```

Figure 11: Résumé de réseau

4 Ouverture

Comme nous disposons d'une très grande quantité de données avec 0 comme précipitation, nous avons eu l'idée d'utiliser un classifieur pour prédire d'abord s'il allait pleuvoir ou non. Nous avons essayé les 7 classifieurs et les deux jeux de données suivants. Globalement, nous avons obtenu de meilleures performances sur le dataset 2. Le dataset 3 contient des informations géographiques sur les stations ainsi que la saison ce qui peut induire le classifieur en erreur.

Classifier	`accuracy_score(onDataset2)`	`accuracy_score(onDataset3)`
ExtraTreesClassifier	0.789	0.741
RandomForestClassifier	0.818	0.725
AdaBoostClassifier	**0.834**	0.727
BaggingClassifier	0.715	0.715
DecisionTreeClassifier	0.720	0.666
GradientBoostingClassifier	0.806	0.729
XGBClassifier	0.812	0.724

Table 1: Précision obtenue après classification dans différents jeux de données

Tout d'abord nous classons par le `AdaBoostClassifier`, si le classifieur prédit qu'il ne pleuvra pas, alors nous mettons la valeur de prédiction à 0. S'il prédit qu'il pleuvra et que la probabilité de pluie dépasse un certain seuil, nous prenons la valeur de prédiction à sa valeur de prédiction `XGBoost`. La raison pour laquelle nous fixons ici un seuil supérieur à 0,5 est que nous voulons minimiser le MAPE, qui amplifie les erreurs de prédiction lorsque la quantité de pluie est faible, nous aurions donc tendance à prédire du côté des faibles.

En ajoutant un classifieur préexistant à `XGBoost` pur, nous avons obtenu une amélioration de 0,15 sur MAPE. Mais lorsque nous regardons à nouveau les valeurs prédites, nous obtenons des zéros partout, car les probabilités prédites par le classifieur sont essentiellement toutes concentrées entre 49,9% et 50,1%. Il ressort de l'étape d'observation des données que la distribution de pluie ou non sur ces variables que nous avons utilisées est fondamentalement similaire. Les réponses données par les classifieurs sont donc également assez ambiguës.

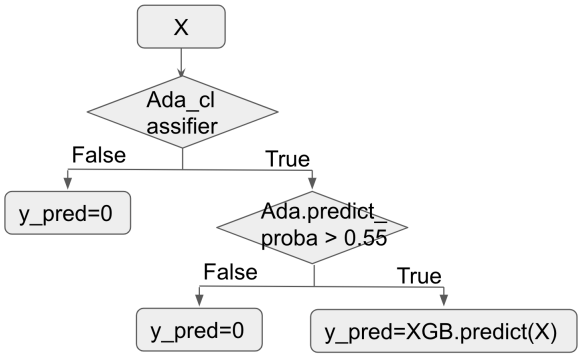

Figure 12: Ordinogramme de la classification et de la régression

Nous pouvons donc nous demander si, même si ce modèle retourne un assez bon score, est-il le modèle que nous devrions adopter ? Une prédiction qui obtient une assez bonne note, mais qui ne reflète pas du tout les caractéristiques des données, est-elle une bonne prédiction ?

5 Conclusion

Ce défi nous a permis d'expérimenter le cheminement effectué par un data-scientist "dans la vrai vie" sur un projet, de l'imputation des données manquantes à la comparaison et la critique de nos différents modèles. Afin de pouvoir prédire au mieux les quantités journalières de pluie, nous analysons les caractéristiques des données, complétons les données météorologiques puis utilisons différents ensembles de données pour comparer les résultats de tous les algorithmes d'apprentissage automatique et de séries temporelles que nous avons testés. Comme il s'agit d'un concours ouvert, la compétition a été intense et nous avons eu envie d'améliorer nos algorithmes afin d'obtenir le meilleur score possible.

Revenons au sujet lui-même pour parler de la fonction de perte. L'algorithme de XGBoost renvoie une prédiction constante quelle que soit la valeur d'entrée. Alors que cela nous donne un bon score, la solution n'est pas satisfaisante car l'algorithme est "paresseux": il trouve simplement une constante qui minimise la fonction de perte. Tout comme lorsque nous récupérons une image avec une partie masquée. L'algorithme nous renvoie une pièce plutôt floue et vague : "Je ne suis pas précis, mais je ne suis pas terriblement faux non plus." Sur le TP d'Image, nous avions résolu ce problème en optimisant la fonction de perte. Ainsi, dans ce défi, nous aurions pu envisager d'utiliser une fonction de perte plus complexe, ou de pénaliser les prédictions avec une petite variance pour obtenir des données de prédiction plus réalistes.

Finalement, nous avons choisi les réseaux de neurones convolutionnels pour prédire la quantité de pluie. En effet, ce réseau offrait un bon compromis entre minimisation du score et respect de la structure des données d'entrée comme le montre les résultats.