INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

# Défi IA 2022

## Prédiction du cumul de pluie journalier dans le quart Nord-Ouest de la France


Célia Duluc
Thomas Nivelet
Elisa Escanez
Sébastien Castets

*AI Frameworks*

5e année - Mathématiques appliquées
Institut National des Sciences Appliquées
Université Fédérale Toulouse Midi-Pyrénées

13 janvier 2022

# Table des matières

# Introduction

Ce rapport traite des démarches suivies et des résultats que nous avons obtenus lors de notre participation au *Défi-IA 2022* sur Kaggle, un concours d'intelligence artificielle organisé chaque année entre étudiants. Nous avons eu l'occasion d'y participer dans le cadre du cours d'AIF (*Artificial Intelligence Frameworks*).

Cette année, l'objectif du défi était de prédire le cumul des précipitations journalières, observé au niveau de différentes stations du quart Nord-Ouest de la France.

Les données utilisées sont fournies par Météo-France. Deux types de données sont disponibles (cf. section 1). On s'intéresse aux années 2016 à 2019. Les données d'entraînement correspondent à la période de 2016 à 2017, les données de test aux années 2018 et 2019. On considère que deux années sont nécessaires pour assimiler la variabilité d'une année.

Remarque : Nous avons fait le choix de ne pas utiliser d'approche séquentielle pour ce projet. En effet, les données dans l'échantillon test pour lesquelles nous devons prédire sont mélangées aléatoirement. Il nous paraissait donc préférable de ne pas utiliser cette approche.

La phase de traitement des données a été réalisée en local en Python du fait du large volume que représentent les données (environ 80Go). Pour la phase de mise en place des modèles, nous avons travaillé avec Google Colaboratory.

Pour une reproductibilité de notre code, nous avons ensuite procédé à une migration sur un dépôt Git. Toutes les informations sur les procédures à réaliser pour reproduire nos résultats sont disponibles en suivant le lien suivant : `https://github.com/Thomas-Nivelet/Defi-IA-2022`.

# 1 Présentation des données

Nous disposons de données de deux natures différentes ainsi que de la variable objectif :

— $X_{station}$ (données historiques) : contient 6 mesures de paramètres météorologiques, observés par station.

— $X_{forecast}$ (données prévisions) : contient des prévisions de Météo France faites à partir de modèles physiques.

— $Y$ : correspond à la variable objectif. Elle contient le cumul des précipitations sur une journée pour chaque station. On ne dispose de cette variable que pour la partie *train* des données.

## 1.1 Données historiques

Le premier jeu de données, que nous appelons $X_{station}$, contient les données observées le jour même pour chaque station auxquelles nous avons ajouté les coordonnées GPS. Il contient les informations suivantes :

— `number_sta` : le numéro de la station météorologique,

— `lat` : la latitude de la station,

— `lon` : la longitude de la station,

— `height_sta` : la hauteur de la station,

— `dd` : la direction du vent en degrés,

— `ff` : la vitesse du vent en mètres par seconde,

— `precip` : la précipitation en $kg.m^{-2}$,

— `hu` : l'humidité en pourcentage ,

— `td` : la température de rosée en Kelvin,

— `t` : la température en Kelvin.

## 1.2 Données de prévision

Le deuxième type de données correspond aux prévisions effectuées par Météo-France par des modèles physiques. Nous utilisons les prédictions de deux modèles météorologiques différents :

— `AROME`, un modèle de prévision avec un maillage fin d'une résolution spatiale de 0.025,

— `ARPEGE`, un modèle ayant un maillage moins fin d'une résolution spatiale de 0.1.

Chaque jour, chacun de ces deux modèles tourne et les prévisions heure par heure de la journée sont enregistrées dans un fichier. Les différents paramètres météorologiques considérés sont les suivants :

— `t2m, d2m` : la température et la température de rosée en Kelvin à 2 mètres du sol,

— `r` : l'humidité relative en pourcentage à 2m du sol,

— `w, u10, v10` : la vitesse du vent et les composantes U et V du vent en $m.s^{-1}$ à 10 mètres du sol,

— `p3031` : la direction du vent en degré,

— `msl` : la pression moyenne en Pascal relevée au niveau de la mer,

— `tp` : la précipitation totale depuis le début de la journée en $kg.m^{-2}$ ou $mm$.

Ces données sont disponibles en chaque point du maillage.

Nous avons aussi utilisé les données provenant du modèle `ARPEGE 3D height`. Elles contiennent les prévisions de la pression atmosphérique en Pascal à 20m, 100m, 500m, 875m, 1375m, 2000m et 3000m au dessus du sol en chaque point du maillage.

# 2 Traitement des données

## 2.1 Traitement des données historiques : $X_{station}$

Le jeu de données historiques fourni par Météo-France comporte un grand nombre de données manquantes. En effet, il y a 26% de données manquantes dans le jeu de données $Train$ et 28% dans le $Test$. Il est important

d'assurer la bonne gestion de ce problème afin d'assurer la qualité de la prédiction. Nous avons ainsi utilisé les méthodes `ffill` et `bfill` de la fonction Python `fillna` de Pandas. Il s'agit de remplir les données manquantes respectivement par la valeur qui la précède ou qui lui succède dans le jeu de données en les groupant par jour et par station au moyen de la fonction `groupby`. Cela nous a semblé cohérent puisque nous disposons des valeurs de chacune des variables heure par heure. On s'imagine donc que d'une heure à l'autre, les valeurs d'une même variable ne sont pas trop éloignées. Malgré ce premier traitement sur $X_{station}$, le pourcentage de données manquantes reste élevé de par les nombreuses absences de données sur des journées entières.

Pour chacune des variables, nous disposons des valeurs par station et par heure. Comme nous voulons prédire les précipitations accumulées sur une journée, nous avons décidé d'agréger ces informations sur une journée. Nous avons sommé les 24 valeurs de la journée pour la variable `precip` afin d'obtenir les précipitations cumulées et pour toutes les autres variables numériques nous avons moyenné pour obtenir une seule valeur sur la journée . Enfin, nous disposons également dans notre jeu de données de l'information sur le mois de l'année. Nous transformons les modalités de cette variable qualitative `month` en indicatrices (*dummy variables*).

## 2.2 Traitement des données de prévisions : $X_{forecast}$

Pour construire $X_{forecast}$, nous avons procédé comme suit :

- Pour chaque station, nous avons calculé les distances euclidiennes aux $K$ points du maillage `AROME` (ou `ARPEGE`) les plus proches. Pour `AROME` on prend 5 points et pour`ARPEGE` 3 points car son maillage est moins fin.
- Pour trouver la mesure au point de chaque station, nous avons interpolé grâce aux mesures des $K$ points des maillages trouvés précédemment. Nous avons procédé à une interpolation par pondération inverse à la distance. Elle consiste à calculer les interpolations aux points $x$ comme suit :

$$u(x) = \frac{\sum_{k=0}^{K} w_k(x)^p u_k}{\sum_{k=0}^{K} w_k(x)^p},$$

où

- $w_k$ est une fonction de pondération défini par $w_k(x) = \frac{1}{d(x,x_k)}$.
- $d$ est la distance euclidienne.
- $x_k$ est un point connu (un des $K$ points du maillage).
- $u_k = u(x_k)$.
- $p$ est un nombre positif réel, appelé paramètre de puissance. Sa valeur est choisie selon le degré de lissage désiré pour l'interpolation. Nous avons choisi de poser $p = 1$ afin de donner plus d'importance aux mesures des points les plus proches des stations.

Il est important de noter qu'un pourcentage conséquent de données manquantes existe dans $X_{forecast}$.

## 2.3 Construction d'un unique jeu de données

Nous avons concaténé les jeux de données $X_{station}$ et $X_{forecast}$ en un jeu de données nommé $X$. Une ligne correspond aux données historiques du jour $J-1$ et aux données prévisions du jour $J$.

Il est important de noter que la $i$-ème ligne de $X$ correspondant au $i$-ième jour est utilisée pour prédire la précipitation cumulée du $i$-ème jour ( $i$-ème ligne de $Y$).

## 2.4 Traitement des données manquantes

Pour traiter les données manquantes de la variable cible $Y$, nous supprimons toutes les lignes du jeu de données d'entraînement dont la valeur de la variable cible correspondante était manquante. Le but est de faire apprendre nos modèles sur des données dites « pures »,

Pour les données manquantes dans $X$, ayant énormément de données, il nous aurait été possible de simplement supprimer toute ligne comportant au moins une donnée manquante. La quantité d'information disponible aurait grandement diminué, mais serait certainement restée suffisante pour notre objectif. Cependant, l'échantillon de test

sur lequel nous devons prédire comporte également des données manquantes. Il nous a donc fallu trouver un moyen de gérer ce problème pour imputer les données manquantes et non simplement les supprimer afin d'appliquer le même traitement aux jeux de données $Train$ et $Test$. Nous avons choisi l'algorithme *MissForest* pour remplir toutes les données manquantes. Il faut noter que nous avons éxécuter deux $Missforest$ différentes : une sur le $Train$ et une sur le $Test$. Au total, cette opération a duré 7 heures environ.

*MissForest* est une approche itérative d'imputation de données manquantes qui utilise des forêts aléatoires.

On commence par combler les données manquantes en utilisant une méthode simple de la moyenne ou de la médiane par exemple. Ensuite, les données manquantes comblées sont marquées comme des données à prédire. Toutes les autres lignes du jeu de données sont les données d'entraînement. Sur ces lignes d'entraînement, on ajuste un modèle de forêts aléatoires pour générer des prédictions sur les données manquantes. On répète cette procédure plusieurs fois, en considérant à chaque fois une partie différente des données manquantes comme données d'entraînement. Ainsi, à chaque itération, les prédictions des données manquantes s'améliorent.

Plusieurs raisons nous ont menées à sélectionner cette méthode d'imputation des données manquantes. C'est une méthode qui ne nécessite aucun pré-traitement des données, aucune optimisation de ses paramètres, et qui accepte les données mixtes (quantitatives et qualitatives). Elle est généralement assez efficace, même si le coût peut être assez élevé pour des gros jeux de données. De plus, comme un algorithme de forêt aléatoire classique, on peut parvenir à déterminer les variables importantes.

## 2.5 Séparation des données d'entraînement

Afin de pouvoir tester notre modèle avant de le soumettre, nous séparons les données d'entraînement dont nous disposons en deux échantillons : un échantillon d'apprentissage et un échantillon de validation. L'échantillon d'apprentissage nous permet d'apprendre le modèle sur les données. L'échantillon de validation permet de valider les capacités de généralisation de notre modèle, et donc sa performance sur de nouveaux jeux de données.

Afin de répartir au mieux ce partage, nous avons séparé l'échantillon d'entraînement selon la distribution de la variable réponse. Ainsi, les distributions de la variable cible sera ressemblante pour les deux échantillons.

## 2.6 Traitement des valeurs extrêmes

Afin de réduire l'effet de valeurs potentiellement aberrantes dans notre jeu de données, nous limitons les valeurs de la variable réponse à celles contenues dans l'intervalle de confiance à 95% correspondant. On calcule la borne supérieure de cet intervalle. Puis, si la valeur de `Ground_truth` est plus grande que la borne supérieure, on la remplace par cette dernière. On ne change pas la borne inférieure. On seuille uniquement sur l'échantillon d'apprentissage.

## 2.7 Normalisation des données

Une fois les échantillons séparés, nous avons normalisé les données, selon la moyenne et la variance de l'échantillon d'apprentissage $X_{train}$. Cela permet de pouvoir comparer les variables explicatives et leur importance sur la même échelle, et réduit également le temps d'apprentissage.

# 3 Méthodologie et algorithmes de machine learning

## 3.1 Classification ou régression ?

Nous cherchons à prédire les précipitations accumulées sur une journée au niveau de stations météorologiques du Nord-Ouest de la France. Il s'agit d'une variable quantitative continue, le problème à traiter est donc un problème de régression. La méthode la plus intuitive est donc d'appliquer une méthode de régression directe. On entraîne un régresseur sur l'échantillon d'apprentissage, qui pourra être performant sur tout échantillon de test (capacités de généralisation).

Une deuxième approche peut consister à procéder successivement à une classification puis à une régression. En effet, nous avons remarqué que le jeu de données fourni comportait beaucoup de valeurs nulles pour la variable à prédire. Ainsi, commencer le travail de prédiction par un problème de classification (une classe correspondant aux précipitations nulles et une classe correspondant aux précipitations non nulles), permettrait d'avoir une erreur exactement égale à 0 pour les prédictions bien classées ! Il suffirait par la suite d'appliquer un régresseur sur les données prédites dans la classe des précipitations non nulles. Ce régresseur est au préalable entraîné uniquement sur les données pour lesquelles la valeur de la précipitation à prédire est strictement supérieure à 0. On concatène pour finir les résultats de classification et de régression. Si notre classifieur est assez performant, cela permet de diminuer l'erreur de régression.

## 3.2 Prédiction directe ou indirecte ?

Comme nous l'avons précisé auparavant, nous disposons de données historiques, réellement observées au jour $J$. Nous disposons également de prévisions données par les modèles de Météo-France. Ainsi, deux approches sont envisageables :

- On prédit directement la valeur de la variable `Ground_truth`, correspondant à la précipitation accumulée journalière.
- On utilise la valeur `arome_tp` (ou `arpege_tp`) qui correspond à la valeur de la précipitation accumulée journalière prédite par le modèle `AROME` de Météo-France (respectivement par le modèle `ARPEGE`). On prédit l'erreur commise par le modèle (`arome_tp` − `Ground_truth`) (respectivement `arpege_tp` - `Ground_truth`), et on ajoute cette erreur prédite à la valeur prédite par Météo-France pour obtenir notre prédiction finale de la précipitation.

## 3.3 Quel modèle ?

Nous avons pu au cours de ces derniers mois tester plusieurs modèles de Machine Learning, et plus particulièrement les réseaux de neurones et l'algorithme Light GBM (Light Gradient Boosting Method).

### 3.3.1 Réseaux de neurones

Les réseaux de neurones constituent les briques élémentaires du deep learning. Ils sont beaucoup utilisés dès lors que l'on cherche à modéliser des données avec des architectures complexes, qui combinent plusieurs transformations non-linéaires. C'est pourquoi l'on a choisi de tester l'utilisation de tels modèles.

Un réseau de neurones est constitué de plusieurs couches cachées de neurones. La sortie d'un neurone devient l'entrée d'un autre neurone, et ainsi de suite. Un neurone artificiel est une application $f$ non linéaire par rapport à ses paramètres. Elle associe à une entrée $x$ une sortie $y$ qui peut s'exprimer de la manière suivante : $y_j = f_j(x) = \phi(\langle \omega_j, x \rangle + b_j)$. On applique à l'entrée un poids $\omega_j$, puis on rajoute un biais $b_j$. Pour obtenir la sortie $y_j$, on applique finalement une fonction d'activation $\phi$. La procédure précédente est répétée pour chaque neurone, avec des valeurs différentes des paramètres. On estime ces derniers par la minimisation d'une fonction de perte, à l'aide d'un algorithme de descente de gradient.

Les paramètres d'une telle architecture sont le nombre de couches cachées du réseau, ainsi que le nombre de neurones pour chaque couche. La force du réseau réside cependant dans sa profondeur, c'est-à-dire son nombre de couches cachées. Un autre paramètre important est le nombre d'époques (`nb_epochs`), c'est-à-dire le nombre d'itérations opérées sur toutes les données d'apprentissage.

### 3.3.2 Light Gradient Boosting Method

Le modèle LGBM est une méthode de *gradient boosting*. Elle consiste en l'agrégation de plusieurs modèles simples pour obtenir de meilleures prédictions. Les prédicteurs élémentaires utilisés ici sont les arbres de décision. Le principe est d'optimiser une fonction de perte différentiable, au moyen d'un algorithme itératif de descente de gradient.

Dans cette méthode, l'optimisation du découpage de l'arbre se fait du point de vue des feuilles (et non de la profondeur de l'arbre comme dans la plupart des méthodes). Pour la construction de chaque arbre, l'algorithme choisit la feuille qui maximise la diminution de la fonction de perte si on décide de la partager en deux noeuds fils. De plus, beaucoup de méthodes cherchent le meilleur découpage possible d'un arbre à partir des valeurs triées des variables. Ce n'est pas le cas de ce modèle qui implémente un algorithme d'apprentissage basé sur un histogramme optimisé.

La méthode LGBM a donc plusieurs avantages, qui nous ont incité à la choisir. Le premier avantage est une meilleure précision, car la fonction de perte est plus réduite grâce à la méthode d'apprentissage "leaf-wise". Cependant il faut être vigilant sur ce point. On obtient une meilleure précision car les arbres construits sont plus complexes. Il faut donc faire attention au sur-apprentissage. Le paramètre `max_depth` permet d'avoir un levier d'action sur ce problème.

Comme cette méthode transforme les valeurs continues en quelques valeurs discrètes, elle garantit une plus grande rapidité à l'apprentissage, et réduit les besoins mémoires. Sur des jeux de données assez grands, on observe généralement de nettes améliorations des temps de calcul, pour des performances équivalentes à d'autres méthodes de *gradient boosting*.

Nous introduisons ici quelques paramètres importants de cette méthode, que nous avons choisi d'optimiser lors de l'apprentissage du modèle sur les données d'entraînement.

- `max_depth` : profondeur de l'arbre. Ce paramètre permet notamment de contrôler le problème de sur-apprentissage.
- `num_leaves` : nombre de feuilles dans un arbre.
- `min_child_samples` : nombre de données minimum que doit contenir une feuille.
- `learning_rate` : Ce paramètre contrôle la vitesse à laquelle le modèle apprend. Sa valeur est comprise entre 0 et 1. Plus elle est basse, plus le modèle est lent à apprendre, mais plus le modèle est robuste.
- `reg_alpha` : terme de régularisation L1.
- `n_estimators` : nombre d'arbres à ajuster (nombre de prédicteurs élémentaires).

## 3.4 Approche finale retenue

Après avoir testé plusieurs algorithmes et plusieurs stratégies, les résultats obtenus après entraînement de réseaux de neurones étaient semblables à ceux obtenus par l'algorithme LGBM (cf. section 3.3.2). D'autre part, nous n'avons pas retenu l'approche combinant une classification et une régression (cf. section 3.1). En effet, la performance offerte par cette stratégie était moins bonne que par une régression directe étant donné que l'on a obtenu un nombre non négligeable de prédictions nulles pour des valeurs observées non nulles. Enfin, nous n'avons pas adopté l'approche par prédiction indirecte (cf. section 3.2), qui consistait à prédire l'erreur commise par les modèles physiques (`AROME` et `ARPEGE`), puis d'ajouter cette erreur à la valeur prédite par Météo-France. Par le biais de cette technique, nous avons obtenu certaines valeurs prédites négatives, et donc une performance moins bonne qu'en prédisant directement la variable cible `Ground_truth`.

En ce sens, d'une manière assez arbitraire, nous avons opté pour un réseau de neurones comportant 20 couches de 32 neurones implémenté avec la librairie `Keras` de Python, pour prédire la variable `Ground_truth` directement par régression. Les données en entrée sont l'ensemble des données historiques ($X_{station}$) et les données de prévision ($X_{forecast}$). Le réseau est entraîné avec la Mean Absolute Error (MAE) comme fonction de perte. Les paramètres du réseau sont les suivants :

- `batch_size` = 128,
- `nb_epochs` = 50.

Il est précisé que lors du calcul du score MAPE, chaque valeur observée était augmentée de 1, afin d'éviter la division par zéro. C'est pourquoi avant chaque soumission, nous ajoutons 1 à toutes nos prédictions.

## 3.5 Entraînement et évaluation

### 3.5.1 Entraînement des modèles

L'entraînement de notre modèle a été effectué sur l'échantillon d'apprentissage. Nous l'avons optimisé relativement au score MAE (Mean Absolute Error). Celle-ci est très proche de la métrique utilisée pour évaluer nos prédictions sur `Kaggle` : la métrique MAPE.

**Définition 1 (Mean Absolute Percentage Error)**

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

*où*

- $F_t$ *est la valeur de la prédiction* $t$,
- $A_t$ *est la valeur observée* $t$,
- $n$ *est le nombre de valeurs à prédire.*

### 3.5.2 Évaluation de la performance de nos modèles

Afin d'éviter le sur-apprentissage, nous évaluions donc les propriétés de généralisation de nos différents modèles en calculant le score R2 et la MAPE sur l'échantillon de validation.

Une fois que l'on obtenait des valeurs intéressantes de ces deux points de vue là, c'est-à-dire une valeur assez basse de la MAPE ainsi qu'un score R2 assez élevé, alors nous soumettions une tentative sur Kaggle. En fonction des résultats de la soumission, nous avisions pour améliorer nos modèles en réfléchissant aux principales possibilités.

# 4 Synthèse et résultats obtenus

## 4.1 Résultats et performances du modèle

Après avoir entraîné notre réseau précédemment présenté (cf. section 3.4), nous obtenons les résultats indiqués ci-dessous (Table 1).

| | |
|---|---|
| $MAPE$ loss | 18.81 |
| $MAE$ loss | 0.61 |
| $R^2$ | 0.69 |

TABLE 1 – Résultats obtenus après entraînement du réseau de neurones.

Après prédiction sur le jeu de données test, composé d'environ 85 000 lignes, le score Kaggle (en MAPE) obtenu est indiqué ci-après (Table 2). Le classement public est calculé sur 47% du jeu de données test, le classement privé, qui est aussi le classement final du concours, est déterminé à partir des 53 % restants.

| | |
|---|---|
| $MAPE$ (public leaderboard) | 30.60052 |
| $MAPE$ (private leaderboard) | 30.46552 |

TABLE 2 – Résultats obtenus sur Kaggle en score MAPE.

En termes de prédictions, les résultats apparaissent comme acceptables. En effet, nous avons dépassé aisément les deux baselines du concours (`Baseline_observation` et `Baseline_forecast`). Le score de 30 est satisfaisant, et d'autant plus que nous constatons qu'il ne varie pas entre les deux classement, traduisant le fait que nous ne sommes pas en situation de surapprentissage. On peut expliquer la différence de MAPE entre l'échantillon de validation et celui de test par notre séparation aléatoire des données. Les données de validation sont donc très proches des données d'apprentissage, ce qui n'est pas le cas pour les données de test.

## 4.2 Synthèse

Le cheminement suivi lors de ce concours aura été le suivant :

1. Téléchargement et traitement des données ($X_{station}$ et $X_{forecast}$), dont le temps d'exécution est d'environ 30 minutes,
2. Remplissage des valeurs manquantes par l'algorithme *MissForest*, pour un temps de calcul atteignant jusqu'à 7 heures,
3. Entraînement du réseau de neurones et prédiction de la variable réponse `Ground_truth`. Cette dernière étape requiert environ 15 minutes d'exécution.

Cette méthodologie est entièrement reproductible en suivant les instructions indiquées sur le dépôt Git associé [1]. Les étapes 1 et 2 sont optionnelles, il est possible d'entraîner directement le réseau sur le jeu de données préalablement traité.

Ce concours nous aura permis d'appréhender plusieurs méthodes de machine learning et plusieurs stratégies de prédiction pour répondre à un problème complexe, mais bien connu du quotidien, le cumul de précipitations. Nous avons également pu nous confronter aux difficultés liées au traitement de données réelles (valeurs manquantes, *outliers*, jointure de deux types de données différents, ...). Les résultats de prédiction obtenus sont acceptables, mais perfectibles ; c'est en ce sens que nous détaillons dans le paragraphe suivant (section 4.3) des pistes qui peuvent être explorées pour améliorer la performance des modèles.

1. `https://github.com/Thomas-Nivelet/Defi-IA-2022`.

### 4.3 Perspectives d'amélioration

Afin d'améliorer les performances en termes de prédiction, nous citons quelques idées qui peuvent être approfondies. La première consisterait à ajouter de nouvelles features à notre jeu de données grâce aux fonctions `rolling` et `lag`. Une idée serait par exemple d'introduire des moyennes mobiles sur des heures d'une journée, cela permettrait d'obtenir un modèle plus robuste.

Une deuxième idée concernerait le traitement des valeurs manquantes. Au lieu d'utiliser les fonctions `bfill` et `ffill` de Python, il serait intéressant de tester la fonction `interpolate`. Par ailleurs, certaines variables contiennent jusqu'à 40% de valeurs manquantes. On pourrait alors penser qu'il serait plus opportun de supprimer les observations correspondantes du jeu de données, plutôt que de chercher à les combler. En effet, ce remplissage peut parfois conduire à l'introduction de bruit dans le modèle et donc fausser les prédictions.

Enfin, étant donné que seule la variable `month` nous permet de conserver une idée du jour dans le jeu de données, il serait intéressant de traiter autrement cette variable dans le but de conserver l'ordre. En effet, nous l'avons traité comme une variable catégorielle (*dummy variable*), on a alors perdu la notion d'éloignement entre les différents mois. En d'autres termes, en traitant ainsi cette feature, le mois de juillet était aussi proche du mois d'août que du mois de décembre, alors que l'on peut pertinemment concevoir que le cumul de précipitations est différent entre l'été et l'hiver. Conserver cette notion d'ordre sur les mois pourrait ainsi permettre d'avoir une feature significative supplémentaire dans le modèle, et donc possiblement d'améliorer la qualité de la prédiction.