INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

Défi IA 2022

Prédire les précipitations cumulées en des stations météorologiques

Quentin Douzery
Alexia Ghozland
Dario Moed

AI Frameworks

Travail présenté à

David Bertoin
Adil Zouitine

Département de Mathématiques Appliquées
Institut National des Sciences Appliquées
Université Fédérale Toulouse Midi-Pyrénées

14 janvier 2022

Table des matières

Introduction

La prédiction météo est une tâche complexe, faisant intervenir de nombreux phénomènes physiques interdépendants. Historiquement, ces prédictions étaient réalisées à l'aide de modèles physiques numériques, mais ceux-ci sont de plus en plus souvent complétés par de l'Intelligence Artificielle pour améliorer les performances de prédiction.

Dans ce contexte, la sixième édition du Défi IA vise à améliorer ces modèles en proposant une compétition *Kaggle* dont l'objectif est de prédire les précipitations cumulées, pour des stations météorologiques dans le Nord-ouest de la France. Météo France, qui organise cette édition, fournit les données dans le jeu de données *MeteoNet*, qui comprend à la fois les mesures de chaque station, ainsi que les prédictions futures réalisées par un modèle de Météo France. Les données appartiennent à la période $2016 - 2019$.

En participant à ce défi dans le cadre du cours d'*AI Frameworks*, nous avons ainsi pu appliquer concrètement nos connaissances dans le domaine de l'apprentissage machine. L'intérêt est d'autant plus important que le but est ici d'améliorer des modèles utilisés au quotidien par Météo France, pour qui la qualité des prédictions de précipitations est un enjeu de taille.

Dans une première partie, nous allons tout d'abord aborder le pré-traitement des données que nous avons réalisé, après avoir brièvement décrit les jeux de données. Nous nous attarderons ensuite sur les différentes méthodes de *features engineering* utilisées au cours de ce défi pour améliorer nos prédictions. Par la suite, nous détaillerons certains des modèles que nous avons soumis sur *Kaggle*, ainsi que les méthodes d'apprentissage associées. Enfin, nous détaillerons l'évolution des résultats obtenus au cours du défi et les conclusions que nous avons pu en tirer.

1 Pré-traitement des données

1.1 Description des jeux de données

Pour ce défi, nous disposons de plusieurs jeux de données :

- $X_{station}$ contient les observations météorologiques au jour $J-1$, pour 325 stations différentes. Le jeu de données est séparé en deux : une partie *train* et une partie *test*. Celles-ci correspondent respectivement aux observations entre 2016 et 2017, et entre 2018 et 2019.

  $X_{station_{train}}$ (abrégé en X_{train}) est composé de 4409474 lignes (une ligne par heure disponible pour chaque station) et 9 colonnes, une pour chaque paramètre :

Paramètre	Description
$number\_sta$	numéro de la station
$date$	au format YYYY-MM-DD HH :mm :ss
$\boldsymbol{ff}$	vitesse du vent en $m.s^{-1}$
$\boldsymbol{t}$	température en Kelvin
$\boldsymbol{td}$	température du point de rosée en Kelvin
$\boldsymbol{hu}$	humidité en pourcentage
$\boldsymbol{dd}$	direction du vent en degrés
$\boldsymbol{precip}$	précipitation à la $date$ indiquée, en $kg.m^2$
Id	identifiant de la station, au format : $number\_sta\_jour\_HH$

TABLE 1 – Tableau descriptif des paramètres de X_{train}

  $X_{station_{test}}$ (abrégé en X_{test}) est composé de 2304802 lignes et 8 colonnes. Il contient les 6 paramètres météorologiques - indiqués en gras dans le tableau ci-dessus - et les variables *Id* et *month*, allant de 1 à 12. X_{test} ne fournit pas d'informations quant à la date précise (quel jour au sein du mois).

- $X_{forecast}$ contient les prévisions des modèles physiques de Météo France au jour J. Nous disposons d'un modèle à maille fine appelé *AROME* et d'un modèle à grande maille appelé *ARPEGE*. Pour chaque zone géographique, les deux modèles ont été exécutés chaque jour à 00h et leurs prévisions vont de 00h à 24h, avec un pas de temps de 1h. On dispose des précipitations totales depuis le début de l'exécution du modèle. A noter cependant que nous n'utilisons pas ce jeu de données pour entraîner notre modèle.
- Y contient les précipitations cumulées au jour J. y_{train} est composé de 183747 lignes et 4 colonnes :

Paramètre	Description
$date$	au format YYYY-MM-DD HH :mm :ss
$number_sta$	numéro de la station
$Ground\_truth$	précipitations cumulées
Id	identifiant de la station, au format : $number\_sta\_jour\_HH$

TABLE 2 – Tableau descriptif des paramètres de y_{train}

- Précisons enfin, que nous n'avons pas de y_{test} : les soumissions déposées sur *Kaggle* font office de test de notre modèle.

1.2 Méthodes de pré-traitement

Intéressons-nous maintenant aux méthodes de pré-traitement que nous avons implémentées.

- **Ajout des coordonnées géographiques**
  Nous ajoutons les paramètres *latitude*, *longitude*, et *altitude* aux échantillons X_{test} et X_{train} suivant $number\_sta$ (extrait d'Id dans X_{test}). Ces deux variables nous apportent des caractéristiques géographiques pour notre algorithme de prédiction.
- **Réorganisation des jeux de données**
  Nous ne gardons de y_{train} que les observations présentes dans X_{train}.
  Les colonnes de X_{test} sont réordonnées suivant celles de X_{train}.
- **Gestion des données manquantes**
  D'abord, nous avions supprimé les données manquantes du X_{train} et rempli celles de X_{test} avec les données de la ligne du dessous - c'est la méthode *backfill*. Plus tard, nous avons décidé d'utiliser la méthode $backfill$ pour remplir X_{train} et X_{test}. Cela s'est avéré plus probant. A noter que la méthode *backfill* fait ici sens, puisque remplir une donnée manquante à l'heure h par la valeur à l'heure $h+1$ semble cohérent physiquement parlant.
- **Suppression des outliers**
  Nous supprimons les lignes de X_{train} pour lesquelles la valeur de *precip* est supérieure au maximum de $Ground\_Truth$ observé dans y_{train}. En effet, une fois le remplissage des données manquantes réalisé, nous sommons les *precip* sur la journée (cf. Partie 2) ; il se peut alors que les valeurs de *precip* additionnées soient désormais plus grandes que celle maximale se trouvant dans y_{train}.

  Nous supprimons également les observations qui sont sensiblement différentes des autres : pour chaque variable météorologique est représenté un histogramme des observations. Si certaines observations possèdent une valeur extrême bien distincte des autres, alors elles sont supprimées.

  Nous supprimons aussi toutes les observations des stations pour lesquelles la distribution de *precip* est extrême : si l'écart type des valeurs de *precip* pour une station est bien supérieur à l'écart type des *precip* des autres stations, alors les observations correspondant à cette station sont supprimées. A noter que toutes ces opérations ne sont effectuées que sur le jeu de données X_{train}.
- **Normalisation des données**
  Finalement, les données de X_{train} et X_{test} sont normalisées selon la moyenne μ_{train} et l'écart-type σ_{train}. En effet, considérer la moyenne et la variance de l'ensemble de données complet introduirait des informations futures dans les variables explicatives de X_{train}. Cette étape est nécessaire puisque les modèles utilisés sont des réseaux de neurones.

2 *Features engineering*

Nous nous intéressons maintenant aux différentes *features* que nous avons créé au cours du défi. Rappelons que nous entendons par *feature* une variable explicative absente des jeux de données initiaux, et que l'on crée artificiellement dans le but d'apporter des informations supplémentaires à l'algorithme de prédiction, afin que celui-ci gagne en performance et en robustesse.

Voici ci-dessous une revue des *features* que nous avons incorporées à notre modèle à un moment ou à un autre du défi. À noter que toutes n'ont pas eu le même impact sur les résultats.

Somme des précipitations sur la journée

> Pour un jour donné et pour une station donnée, nous sommons la quantité de pluie tombée à chaque heure. Ici, c'est la variable *precip* qui est concernée.
>
> Pourquoi : Les données étant sous la forme d'observations heure par heure, et les prédictions devant se faire quotidiennement, il nous semblait important de connaître le cumul de pluie de la veille. Cette seule variable correspondait d'ailleurs à la *baseline observations* : à chaque prédiction du jour J était attribuée le cumul de pluie du jour $J-1$. Notre tout premier modèle se contentait d'effectuer cette opération, afin de vérifier que nous prenions correctement en main les différents jeux de données.

Moyenne des variables météorologiques

> À l'exception de la variable *precip*, nous effectuons une moyenne jour par jour et station par station de toutes nos variables météorologiques (*ff*, *t*, *td*, *hu*, *dd*). En plus de cela, nous créons trois variables supplémentaires, toujours pour une journée donnée et une station donnée : la moyenne des variables météorologiques la nuit (entre 00 : 00 et 06 : 00), le jour (entre 06 : 00 et 18 : 00), et la soirée (entre 18 : 00 et 00 : 00).
>
> Pourquoi : Dans le même but qu'avec la variable *precip*, la moyenne est ici un moyen de se ramener à des valeurs quotidiennes de nos variables, et non plus juste des observations heure par heure. Cependant, la moyenne journalière n'étant pas forcément entièrement représentative des conditions météorologiques d'un jour donné, nous avons aussi fait le choix de récupérer la moyenne sur différentes périodes de la journée.

Récupération de la dernière observation de la journée

> Pour chaque jour et chaque station, nous stockons dans une nouvelle colonne la dernière valeur enregistrée de chacune des variables météorologiques (variable *precip* incluse), i.e. l'observation enregistrée entre 23 : 00 et 00 : 00.
>
> Pourquoi : L'objectif est ici de capter une certaine temporalité. Puisque nous devons utiliser les données de la veille pour prédire les précipitations, prendre l'observation la plus proche temporellement de la journée à prédire nous semblait pertinent. En effet, celle-ci peut être porteuse d'un changement météorologique récent qui n'aurait pas été capté par la moyenne de la soirée. A l'inverse, cette valeur peut aussi venir confirmer une tendance observée durant toute la journée.

Récupération du mois et de la saison

La variable *month* est présente dans le jeu de données *test*, mais pas directement dans le *train*. Ayant la date, nous la récupérons néanmoins aisément afin de la stocker dans une variable dédiée. Nous créons aussi une variable qui à chaque observation indique la saison associée. Les saisons sont définies sur la base de mois ayant des cumul de précipitations semblables et qui se suivent dans le temps :

- Hiver : de janvier à mars
- Printemps : de avril à juin
- Été : de juillet à septembre
- Automne : de octobre à décembre

Pourquoi : Il ne pleut pas autant en plein mois de juillet qu'au début du mois de novembre. Spécifier explicitement cette information temporelle au modèle nous semblait donc primordial. D'autant plus que c'est la seule information temporelle que nous possédons pour les observations du jeu de données *test*. De plus, nous avons rajouté la saison afin d'obtenir une information plus générale (comportement global sur trois mois). Nous ne sommes pas sûrs de l'utilité de cette dernière variable, car peut-être redondante avec le mois.

Calcul de la *smooth mean* des précipitations sur le mois et la saison

Le mois et la saison sont des variables qualitatives. Nous les transformons en variables quantitatives à l'aide d'une *smooth mean*, un estimateur bayésien. Cet estimateur calcule la moyenne des précipitations pour un mois donné (ou une saison donnée), tout en la pondérant par la moyenne globale de toutes les observations. Si l'on note $\mu_{v,n}$ la *smooth mean* du niveau n de la variable v, nous obtenons la relation suiavnte :

$$\mu_{v,n} = \frac{N \times \bar{x} + m \times w}{N + m}$$

où N est le nombre de valeurs de modalité n, $\bar{x}$ l'estimateur de la moyenne des précipitations pour la modalité n, w la moyenne globale de toutes les observations, et m le poids attribué à la moyenne globale.

Pourquoi : Dans un premier temps, nous ne voulions pas conserver l'information du mois sous forme d'une variable qualitative. En effet, afin que le modèle puisse traiter l'information, il aurait fallu la convertir en 12 variables différentes pour se ramener à des valeurs chiffrées (des *dummies*, i.e. des 0 et des 1). Nous avons donc préféré remplacer chaque modalité par la moyenne associée afin de ne pas augmenter le nombre de variables. De plus, nous avons utilisé une *smooth mean* plutôt qu'une moyenne classique afin d'éviter le surapprentissage qui serait dû à certains mois possédant moins d'observations que les autres. Nous avons fixé le paramètre m à 300. Le raisonnement a été le même pour la variable saison.

Conversion du mois en une combinaison d'un cosinus et d'un sinus

En plus de remplacer la variable qualitative *month* par sa *smooth mean*, nous créons aussi deux *features* supplémentaires : *cos_ month* et *sin_ month*. Chaque mois (numéroté de 1 à 12) est converti en une combinaison d'un cosinus et d'un sinus, stockée dans les deux variables créées. Pour cela, nous normalisons les 12 mois pour les relier au cercle trigonométrique $[0, 2\pi]$. Si nous notons m_i un mois quelconque, $m_{i,2\pi}$ la valeur normalisée de m_i, et cos_{m_i} la valeur de *cos_ month* de m_i, nous obtenons :

$$cos_{m_i} = cos(m_{i,2\pi}), \qquad m_{i,2\pi} = \frac{2\pi \times m_i}{max_{i\in\{1,\dots,12\}}(m_i)}$$

Le principe est ensuite exactement le même pour la variable *sin_ month*, mais en utilisant la fonction sinus. Les correspondances *month* - *cos/sin* sont à retrouver dans le tableau ci-dessous.

month	***cos_ month***	***sin_ month***
1	0.87	0.50
2	0.50	0.87
3	0	1
4	-0.50	0.87
5	-0.87	0.50
6	-1	0
7	-0.87	-0.50
8	-0.50	-0.87
9	0	-1
10	0.50	-0.87
11	0.87	-0.50
12	1	0

TABLE 3 – Correspondances entre les variables *month* et *cos_ month/ sin_ month*

Pourquoi : Dans une année calendaire, le mois d'août est situé après le mois de juillet, mais avant le mois de septembre. De la même manière, le mois de janvier arrive juste après le mois de décembre. Il y a donc une périodicité et un ordre des mois. Indexés par une variable qualitative, nous perdons cette temporalité. Le passage à une combinaison d'un cosinus et d'un sinus s'effectue ainsi dans ce but : donner à l'algorithme l'information que les mois se suivent, jusqu'à même former un cycle. Nous pouvons alors représenter nos mois sur un cercle trigonométrique grâce à leurs coordonnées dans la base cosinus/sinus.

Ajout de la prévision météorologique des modèles physiques

Nous n'avons pas utilisé les données *forecast* au cours du projet, mais nous avons tout de même incorporé les prévisions des modèles physiques à nos jeux de données, par le biais de la *baseline forecast*. Le fichier, téléchargeable directement sur *Kaggle*, renvoie pour chaque jour et chaque station le cumul de pluie prédit par le modèle météorologique au point le plus proche - géographiquement parlant - de la station concernée.

Pourquoi : Rien de savant, juste l'ajout du maximum d'information supplémentaire. En l'occurrence, c'est ici les prévisions météorologiques au jour J, qui viennent compléter les informations enregistrées au jour $J - 1$.

3 Modèles et apprentissage

Après avoir expliqué comment les données ont été pré-traitées et complétées avec de nouvelles *features*, notre attention se porte sur les modèles d'apprentissage que nous avons construits.

3.1 Les différents modèles implémentés

Afin de prendre en main les données et pour avoir un premier aperçu des résultats que nous pouvions obtenir, nous avons tout d'abord implémenté quelques méthodes de machine learning basiques, à savoir :

- Une simple régression linéaire.
- Un arbre de décision optimale : il s'agit d'un unique arbre de décision, dont l'optimisation se fait pour l'arbre entier, et non noeud par noeud comme pour un arbre de décision classique. Cela permet de prendre en compte les interactions complexes entre les différentes variables.
- Une forêt aléatoire, où la prédiction est réalisée par un vote majoritaire de nombreux arbres de décision.
- Un réseau de Neurone, ou plus précisément un Perceptron multicouche.

L'implémentation de ces quelques premiers modèles nous a permis d'obtenir nos premiers résultats et de réaliser nos premières soumissions sur *Kaggle*. Obtenant un score du même ordre de grandeur que la *baseline observations*, il restait encore des progrès à faire jusqu'à arriver au même score que les modèles de prévisions de Météo France.

Néanmoins, une fois ces premiers modèles basiques implémentés, nous avions une première *pipeline* complète, de l'importation des données à l'exportation d'une soumission *Kaggle*. Nous pouvions ainsi nous concentrer sur l'amélioration du pré-traitement, l'ajout de nouvelles *features* et la sélection du meilleur modèle. Présentons maintenant les deux modèles avec lesquels nous avons obtenus nos meilleurs résultats.

Les deux modèles les plus probants : Régresseur et Classificateur-Régresseur

Les deux modèles conservés sont basés sur des réseaux de neurones. Plus que ça, le modèle dit "Régresseur" est un Perceptron multicouche, un des premiers modèles que nous avions implémentés. Il est composé de 20 couches de 32 neurones, activés par une fonction *ReLU* pour assurer que les précipitations prédites soient positives. Enfin, la couche de sortie est de dimension 1, étant donné qu'on ne prédit qu'une valeur de précipitation cumulée par jour.

Le second modèle, un peu plus élaboré, est composé d'un classificateur et d'un régresseur. Le but est de classifier en premier lieu les prédictions en deux sous-catégorie : l'une pour les jours sans pluie (le cumul des précipitations vaut donc 0), et l'autre pour les jours où il a plu (précipitations prédites > 0).

Ce classificateur est quant à lui composé de 10 couches de 64 neurones. Les couches cachées sont également activées par une fonction *ReLU*, mais la couche de sortie est cette fois une sigmoïde, pour obtenir une probabilité d'appartenance à la catégorie pluie ou non pluie en sortie de réseau.

Une fois cette classification faite, les jours pour lesquels la prédiction indique de la pluie sont conservés et servent de donnée d'entrée au régresseur, qui va ainsi prédire la quantité de pluie tombée pendant ce jour. Le régresseur utilisé est le même réseau que celui utilisé indépendamment, la seule modification étant le jeu de donnée sur lequel il réalise les prédictions. Notons que le nombre de couches et de neurones n'a pas été optimisé à l'aide d'un *GridSearch*, mais juste à la main en explorant différentes possiblités. Il est quand même dommage que nous n'ayons pas optimisé au maximum ce point pourtant très important.

Rajoutons que la soumission conservée pour le score final utilise seulement le Régresseur, les résultats obtenus avec ce modèle ayant été plus stables et plus probants. Voyons maintenant comment nous avons estimé la précision et la stabilité de nos modèles.

3.2 Apprentissage des modèles

Les modèles sont entraînés sur 80% des données pré-processées et agrémentées des nouvelles features. Le reste des données est utilisé pour la validation, dont nous détaillerons la méthode plus loin. L'optimiseur utilisé pour l'apprentissage des deux modèles est Adam, une méthode d'optimisation aléatoire basée sur un gradient stochastique. Quant à la fonction de perte utilisée, celle-ci est naturellement différente pour le classificateur de celle utilisée pour le régresseur. Il s'agit assez classiquement de la *binary crossenthropy* pour le classificateur, et de la *MAE* (pour *Mean Absolute Error*) pour le régresseur. Cette dernière étant la fonction perte se rapprochant le plus de la *MAPE* (*Mean Absolute Percentage Error*) utilisée pour quantifier le score sur *Kaggle*.

Les modèles sont entraînés sur 20 *epochs*, dans un compromis fait en tâtonnant entre temps de calcul, robustesse de l'apprentissage et biais de sur-apprentissage. De même, la *batch_ size* est fixée à 200, ce qui représente une très faible partie du dataset total de 4409474 échantillons. Ce choix d'utiliser une faible proportion du jeu de données a plusieurs effets sur l'apprentissage :

- Un faible nombre de *samples* utilisé à chaque itération réduit le besoin en mémoire. Ceci permet de réduire les contraintes de calcul et d'entraîner le modèle sur des machines moins performantes.
- De plus et surtout, l'apprentissage est plus rapide car les calculs de propagation du gradient sont moins coûteux en temps de calcul, tout en mettant à jour les poids à chaque propagation.
- En revanche, utiliser de plus petits échantillons augmente la variance de l'estimation du gradient, et la qualité de l'estimation en est amoindrie.

Il aurait été plus judicieux de faire le choix de ces paramètres *epochs* et *batch_ size* à l'aide là aussi d'un *GridSearch*. Ce serait l'une des pistes d'amélioration qui aurait été intéressante d'approfondir, pour s'assurer que nous ne sommes pas dans un minimum local avec nos paramètres actuels.

Méthode de validation

Comme dit précédemment, le modèle n'est entraîné que sur 80% des données de l'échantillon *train*. Les 20% restants représentent l'échantillon de validation. Nous l'utilisions pour calculer la *MAPE* sur un jeu de données n'ayant pas servi à l'apprentissage. Connaître cette *MAPE* sur des "nouvelles" données permet d'anticiper les scores que nous pourrions réaliser sur les données test du Défi, auxquelles nous n'avons jamais accès.

Cette opération est réalisée dix fois consécutivement pour pouvoir analyser la variance de nos modèles, ainsi que pour augmenter leur robustesse. Il ne s'agit néanmoins pas de *K-fold*, puisque lors d'un découpage du jeu de donnée en une partie apprentissage et une partie validation, on n'exploite pas toutes les combinaisons possibles des différents segments, il s'agit d'une nouvelle répartition randomisée à chaque itération. Nous n'avons pas pris la peine d'effectuer une *K-fold cross validation* pour la raison suivante : nous nous étions aperçu que la non-robustesse du modèle ne dépendait pas du *split train/test*, mais surtout du modèle lui-même (pour un même partage des échantillons, 10 runs de l'algorithme donnaient des *MAPE* ayant une étendue d'environ 5). Si nous avions réussi à implémenter un modèle plus robuste, il est néanmoins évident qu'une *K-fold cross validation* aurait été nécessaire pour évaluer correctement ses performances.

4 Discussions

Avec plus de temps disponible, nous aurions aimé approfondir certains points ou développer de nouvelles idées. Certaines possibilités ont été présentées plus tôt dans le rapport, mais voici un aperçu supplémentaire :

- Afin d'avoir une méthode de validation plus robuste, et un algorithme qui propose les meilleurs résultats possibles sur *Kaggle*, il aurait été intéressant d'utiliser la *MAPE* comme fonction perte. Celle-ci demandait néanmoins à être implémentée à la main, et nous n'avons pas eu le temps de nous pencher dessus.
- Il serait envisageable d'appliquer la méthode des *K-Means* afin de partitionner les stations en N groupes (appelés *clusters*). Nous aurions alors N modèles différents qui seraient entraînés chacun sur chacun de ces groupes.
- Prendre en compte les données de $X_{forecast}$ nous aurait permis, d'une part de corriger les prédictions du modèle physique à l'aide des observations, d'autre part de s'intéresser à prendre les N prévisions les plus proches d'une station plutôt qu'une seule.
- Nous n'avons quasiment exploré que la piste des réseaux de neurones comme modèles à implémenter. Néanmoins, il aurait pu être intéressant de s'attarder sur les algorithmes de *Gradient Boosting*, qui se prêtent bien aux données que nous possédions. Nous aurions notamment aimé nous intéresser à l'algorithme *LightGBM*, connu pour avoir un très bon compromis entre efficacité et rapidité.
- Nous aurions aussi aimé utiliser le fait que nous possédions des données heure par heure, et non pas uniquement quotidiennes. Les réseaux récurrents auraient été une bonne piste de ce point de vue là ; malheureusement, les données *test* n'étaient pas ordonnées et il était dès lors impossible de retrouver la dépendance temporelle des jours entre eux. Une piste à laquelle nous avons pensée aurait été de transformer chaque observation en une nouvelle *feature*. Nous pourrions alors avoir la variable *dd_h22*, la variable *precip_h05*, ou la variable *hu_h12*. En somme, 24×6 variables supplémentaires. Cela pourrait donner des informations sur l'évolution de la météo au cours de la journée, encore plus précises que celles obtenues avec nos moyennes par périodes de la journée (cf. Partie 2).

5 Résultats

Enfin, pour conclure, il nous semblait intéressant d'analyser notre *MAPE*, selon les méthodes utilisées. Ci-dessous, est représentée l'évolution de la *MAPE* en fonction de nos soumissions sur *Kaggle*, et donc des modèles implémentés (cf. Partie 3).

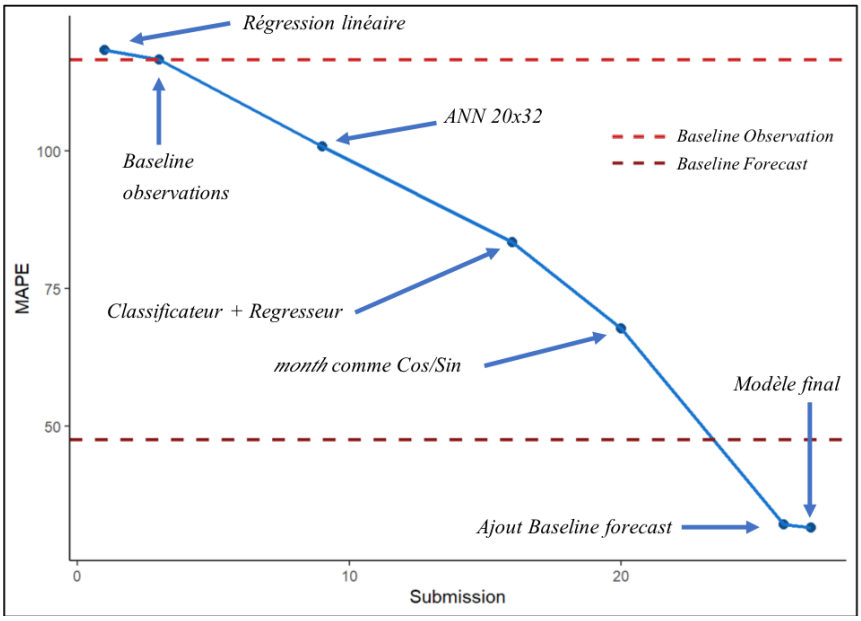

FIGURE 1 – Évolution de la *MAPE* obtenue au fur et à mesure des différentes soumissions

Nous pouvons distinguer trois grandes étapes qui ont contribué à l'amélioration de notre modèle :

- l'implémentation d'un réseau de neurones (20 couches, 32 neurones),
- la considération de la variable *month* comme une variable cyclique, en introduisant les fonctions cos et sin,
- la prise en compte de la *baseline forecast* en tant que *feature*.

Terminons en évoquant le meilleur score que nous avons obtenu sur *Kaggle*. Dans le *private leaderboard*, celui-ci est de $\boldsymbol{MAPE = 29.92}$, ce qui nous place au 26^{eme} rang dans le classement final. Le modèle implémenté à ce moment était le suivant :

- Un régresseur (ANN) de taille 20 couches $\times$ 32 neurones.
- Les $NaNs$ étaient remplacés à l'aide de la méthode *backfill*.
- La *baseline forecast* était prise en compte.
- La *smooth mean* était effectuée sur les précipitations et remplaçait les niveaux des saisons (elle ne remplaçait pas les niveaux des mois).
- La variable *month* avait été convertie sous forme d'une combinaison de cosinus et de sinus.
- Nous ne prenions en compte que la valeur moyenne des paramètres météorologiques sur une journée (pas sur différentes périodes de la journée).

Enfin, les paramètres d'apprentissage étaient les suivants : 20 *epochs* et une *batch size* de 200.