



# Défi Kaggle 2022 : Prédiction de cumul de précipitations

AFESSA Emanuel | AZOUGAGH Fouad | DIALLO Thierno | GODIN Antoine | SEBA Yacine  
EQUIPE DEEPCAMP

Mars 2022

Master 2 SID UPS

## Table des matières

Introduction .....	3
Partie 1 : Sans utiliser les prédictions Météo France .....	3
A. Présentation des données disponibles .....	3
B. Analyse descriptive des données .....	3
B.1. Analyse de tendance centrale et dispersion.....	4
B.2. Analyse de corrélation.....	6
C. Preprocessing des données.....	8
C.1. Gestion des valeurs manquantes.....	8
C.2. Agrégation des données horaires en jours .....	9
C.3. Utilisation des heures comme colonnes.....	10
C.4. Création des features .....	11
D. Approche de modélisation sans deep learning .....	12
D.1. Régression Linéaire .....	12
D.2. DecisionTreeRegressor & RandomForest.....	13
E. Approche de modélisation avec deep learning .....	13
E.1. Démarche de développement .....	13
E.2. Méthodes d'optimisation utilisées.....	14
E.3. Modèle MLP .....	16
E.4. Modèle LSTM .....	18
Partie 2 : En utilisant les prédictions Météo France.....	22
A. Présentation des données disponibles .....	22
B. Interpolation des données aux stations météo .....	22
C. Différentes approches .....	23
Partie 3 : Bilan .....	27
A. Analyse des résultats et stratégie finale .....	27
B. Conclusion.....	29

## Table des figures

Figure 1 : Analyse de tendance centrale et dispersion.....	4
Figure 2 : Analyse de valeurs manquantes.....	4
Figure 3 : Histogrammes de distribution.....	5
Figure 4 : Corrélation entre variables explicatives et variable réponse.....	6
Figure 5 : Représentation de la corrélation par nuage des points.....	7
Figure 6 : Top 8 des variables les plus corrélées avec la variable de réponse.....	8
Figure 7 : Formule de Haversine codée.....	9
Figure 8 : Evolution de la corrélation des variables en fonction des heures conservés pour l'agrégation "Mean".....	10
Figure 9 : Technique de drop out.....	15
Figure 10 : Technique de batch normalization.....	15
Figure 11 : Fonction Loss.....	16
Figure 12 : Simple Perceptron.....	16
Figure 13 : Données retraitées pour le LSTM.....	18
Figure 14 : Entraînement du Vanilla LSTM sur 10 epochs.....	19
Figure 15 : Entraînement du Stacked LSTM sur 10 epochs.....	19
Figure 16 : Entraînement du Bidirectional LSTM sur 10 epochs.....	20
Figure 17 : Répartition des clusters de stations.....	26
Figure 18 : Bilan général des scores.....	28

## Introduction

L'objectif de ce projet Kaggle est de prédire le cumul des précipitations journalières sur les stations d'observation de Météo France. Les données proviennent de MeteoNet, une base de données météorologiques, et sont présentées pour la période 2016-2019 de la région nord-ouest de la France. La métrique d'évaluation à utiliser est la Mean Absolute Percentage Error (MAPE). Pour répondre au problème posé il est demandé d'adopter deux démarches d'apprentissage : La première n'utilisant pas les prédictions faites par Météo France et une seconde qui utilise ces dernières.

## Partie 1 : Sans utiliser les prédictions Météo France

### A. Présentation des données disponibles

Les données de ce projet sont rendues disponibles à deux endroits : On retrouve d'abord sur Kaggle les données peu volumineuses dans l'onglet 'Données', celles restantes sont stockées sur le serveur de MeteoNet. Pour cette première partie de modélisation sans les prédictions Météo France, nous utilisons seulement les données se trouvant sur la page Kaggle, on retrouve alors :

- Le fichier 'X\_station\_train.csv' qui représente les observations horaires de chaque station. Ce fichier plat présente les mesures terrestres de 6 paramètres météorologiques que sont la direction et la vitesse du vent, la température, l'humidité, la température de condensation et les précipitations.
- Le fichier 'Y\_train.csv' qui présente le cumul quotidien de précipitations.
- Le fichier 'stations\_coordinates.csv' qui donne les coordonnées géographiques pour chacune des 325 stations d'observation selon le couple latitude, longitude.

### B. Analyse descriptive des données

L'analyse statistique descriptive nécessite une description des données liée à toutes les variables de la population enquêtée. Celle-ci est faite via une analyse de la tendance centrale des données, de la dispersion des données, de la distribution des données et de quelques graphiques statistiques de base.

### B.1. Analyse de tendance centrale et dispersion

Nous utilisons la fonction 'describe()' pour avoir une analyse synthétique de tendance centrale et de dispersion sur la plage considérée. La sortie correspondante est affichée dans la figure suivante. Le numéro de station n'est pas à prendre en compte dans cette analyse car ce n'est qu'un identifiant.

	ff	t	td	hu	dd	precip
mean	3.773663	284.905957	281.340489	80.902381	192.057181	0.080940
std	2.598984	6.674721	5.270046	15.663812	94.703783	0.444065
min	0.000000	233.800000	222.350000	1.000000	0.000000	0.000000
25%	1.970000	280.320000	277.560000	71.600000	110.000000	0.000000
50%	3.260000	284.480000	281.640000	85.000000	212.000000	0.000000
75%	4.970000	289.300000	285.470000	93.600000	266.000000	0.000000
max	33.110000	325.070000	324.670000	100.000000	360.000000	61.000000

Figure 1 : Analyse de tendance centrale et dispersion

Comme tout projet de machine learning, il est important de découvrir et visualiser les données pour mieux les comprendre. Tout d'abord, nous avons commencé par observer les valeurs nulles ou manquantes.

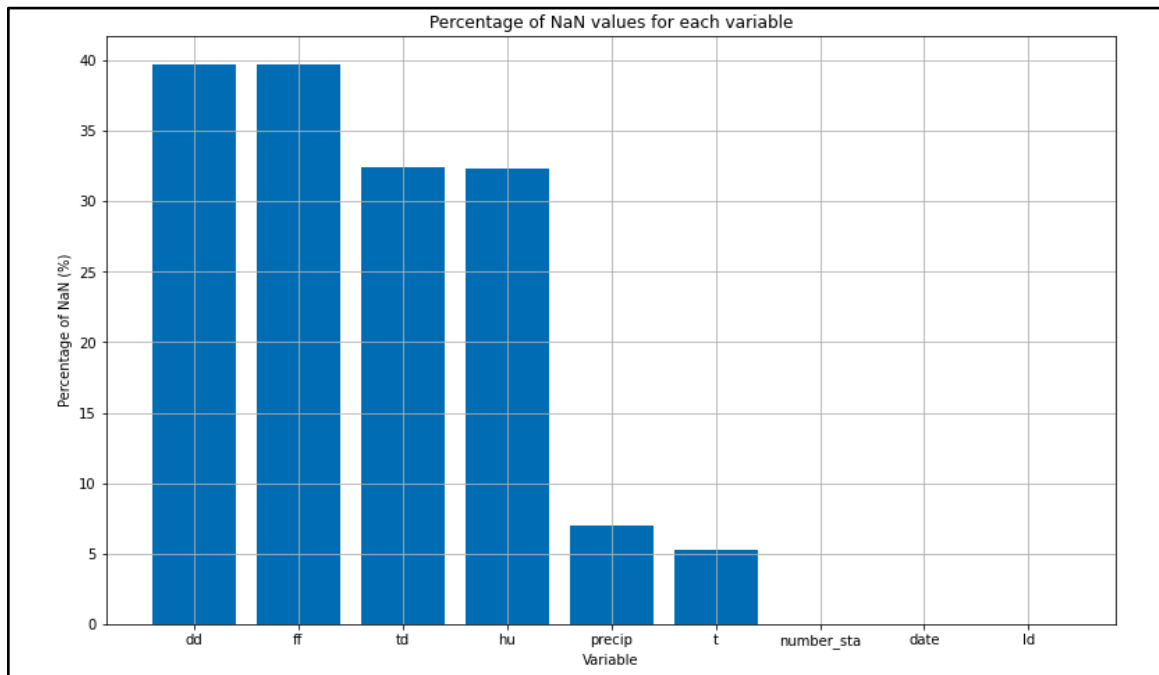


Figure 2 : Analyse de valeurs manquantes

Globalement nous remarquons que l'ensemble des variables ont des données manquantes. D'une part, les variables dd (wind direction), ff (wind speed) ont respectivement chacune 39,7% de valeurs nulles, quant aux variables td (dew point temperature) et hu (humidity) elles ont chacune 32,3% de valeurs nulles. D'autre part, les variables precip (précipitation) et t (température) ont largement moins de valeurs manquantes comparé aux autres (7% pour la variable précipitation, et 5% pour la variable température).

Pour approfondir notre connaissance des données, nous avons choisi de construire un histogramme pour chaque variable explicative comme illustré ci-dessous :

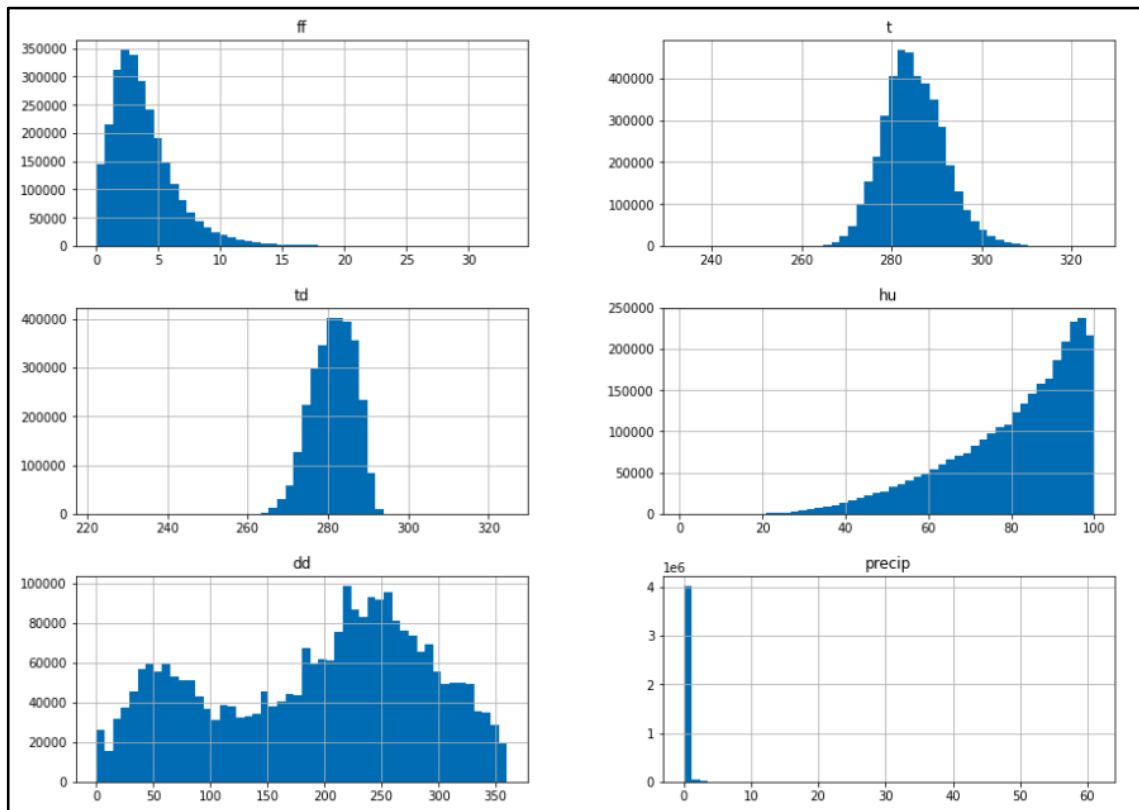


Figure 3 : Histogrammes de distribution

Premièrement, nous constatons que les variables ont des échelles très différentes. Il sera donc utile par la suite de se pencher sur cette différence avant de mettre en place un modèle de prédiction : une solution consisterait à appliquer une normalisation.

Deuxièmement, nous remarquons que certains histogrammes sont fortement dissymétriques. Ces structurations peuvent être difficiles à détecter par certains algorithmes de machine learning : une des solutions consisterait à centrer et réduire ces variables.

## B.2. Analyse de corrélation

Nous allons ici analyser les coefficients de corrélation entre nos variables. Cette analyse nous permettra d'établir une mesure de la force de la relation linéaire entre deux variables. L'analyse de corrélation calcule la quantité de changement dans une variable due au changement dans l'autre.

Tout d'abord, nous avons remarqué que les données des stations sont représentées en heures sur les variables explicatives, tandis que sur la variable à prédire l'échelle est quotidienne. Ceci crée une très large différence entre les dimensions, en nombre de lignes, des deux fichiers. Ainsi pour pouvoir faire toute étude croisée entre les variables explicatives et la variable à expliquer nous avons choisi de transformer les variables explicatives. Cela afin de représenter les données des stations en jours. La section C sur le pré-traitement de données présentera en détail ces transformations.

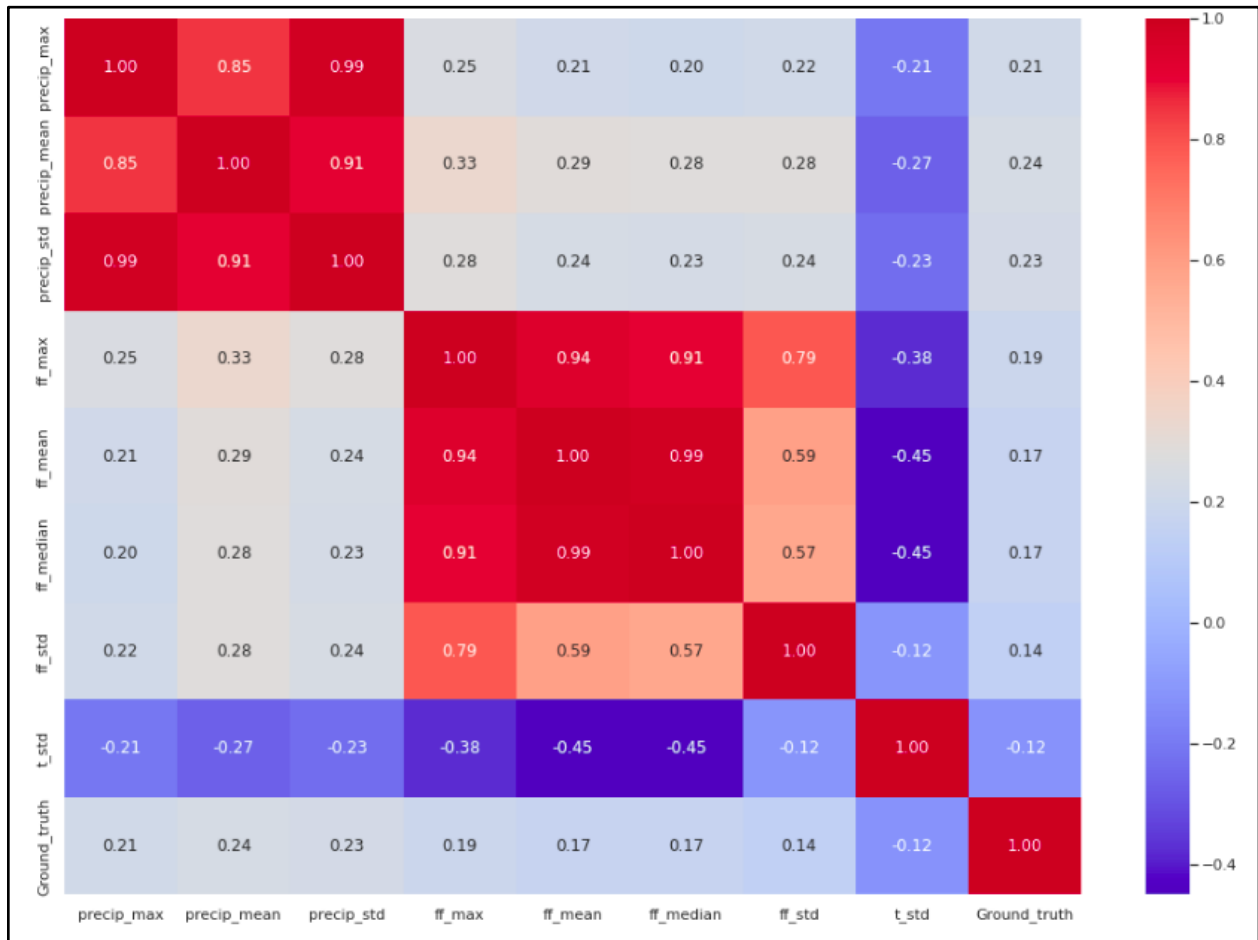
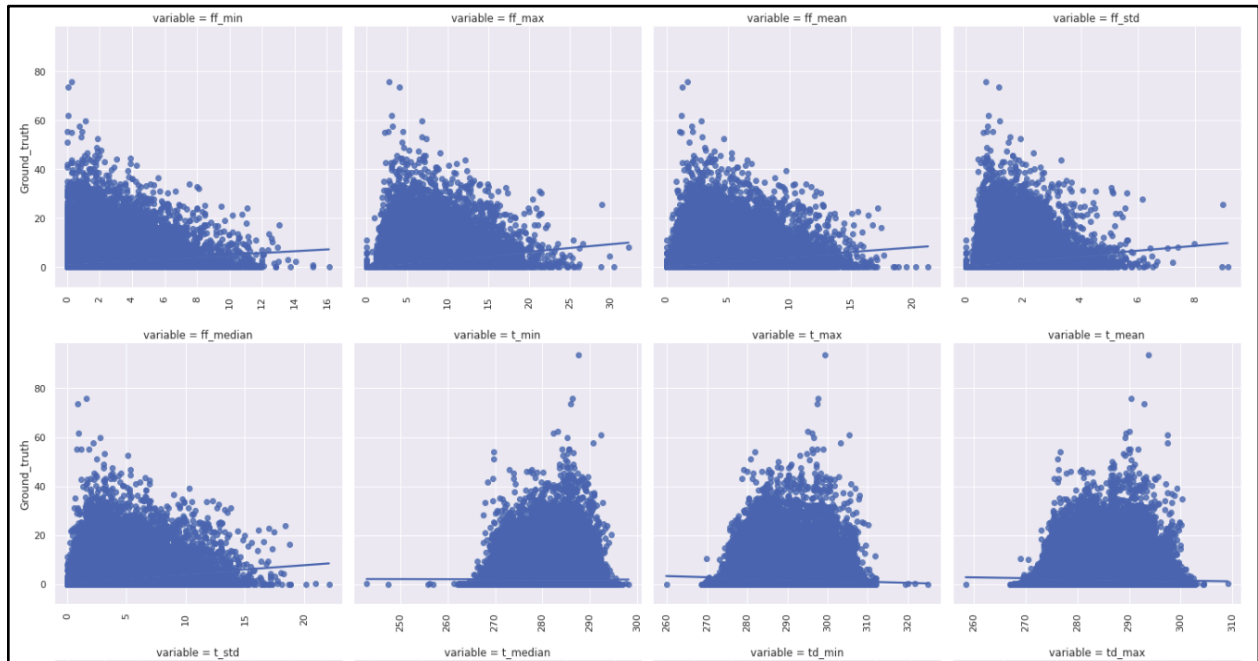


Figure 4 : Corrélation entre variables explicatives et variable réponse

La matrice ci-dessus nous montre la corrélation entre les différentes variables, en particulier entre la quantité de pluie et les autres variables explicatives. Nous constatons qu'aucune des variables explicatives n'est fortement corrélée, positivement ou négativement, avec la variable à expliquer qu'est la quantité de pluie.

Pour voir plus clair sur les corrélations, nous avons utilisé la fonction `'regplot'` de [seaborn](#) qui permet de tracer des données et d'y ajouter une droite de régression linéaire ou une courbe de régression polynomiale.



*Figure 5 : Représentation de la corrélation par nuage des points*

Les nuages des points nous montrent bien l'absence de corrélation entre les variables explicatives et la quantité de pluie; sur chaque figure la droite de régression linéaire a une pente très faible.



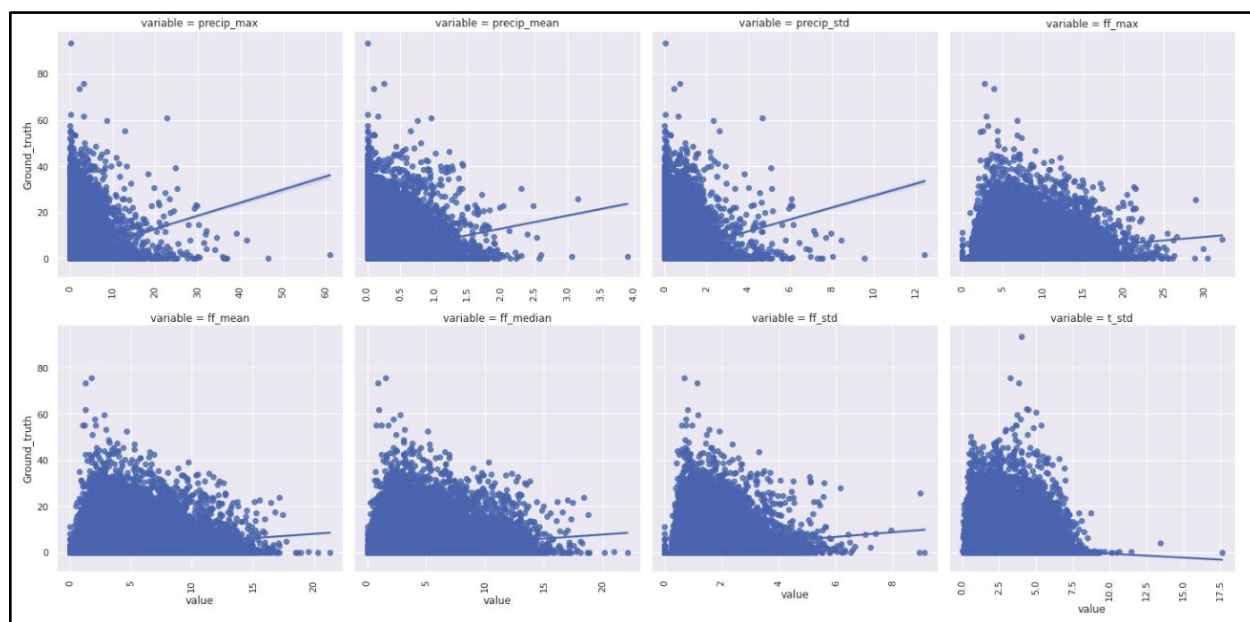


Figure 6 : Top 8 des variables les plus corrélées avec la variable de réponse

Par ailleurs, parmi les variables explicatives nous en avons certaines qui ont les plus grandes valeurs de la mesure de corrélation avec la quantité de pluie, dont le maximum est de 0.24 (entre la précipitation moyenne et la quantité de pluie).

## C. Preprocessing des données

Notre dataset contient plusieurs points nécessitant un pré-traitement. En effet, les variables présentent des grandeurs d'ordre différentes, les variables explicatives sont observées sur une échelle horaire tandis que la précipitation cumulée à prédire est sur une échelle quotidienne et certains champs montrent un très grand nombre de valeurs manquantes.

### C.1. Gestion des valeurs manquantes

Nous avons développé une stratégie de remplacement des valeurs manquantes basée sur la proximité inter-station en établissant pour chaque station un ordre de proximité avec l'ensemble des autres stations.

Au départ l'idée était de remplacer les valeurs manquantes de chaque variable explicative d'une station à un jour et une heure donnée par celle de la station la plus proche, cependant ce qui arrive souvent est que si une station a une valeur manquante la station la plus proche se trouvera dans le même cas. Ainsi, plutôt que de prendre la première station la plus proche nous avons choisi de prendre la première station non nulle la plus proche qui pouvait parfois correspondre à la huitième station la plus proche.

Afin de calculer les distances inter-station, nous nous sommes basés sur les coordonnées latitude/longitude des stations à partir de la formule Haversine suivante fréquemment utilisée dans des problématiques à coordonnées sphériques.

```
[ ]: def haversine(lon1, lat1, lon2, lat2):
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    # Radius of earth in kilometers is 6371
    km = 6371 * c
    return km
```

*Figure 7 : Formule de Haversine codée*

## C.2. Agrégation des données horaires en jours

Nous avons mis en place une intégration visant à mettre toutes les données sur la même échelle temporelle. Pour cela, nous avons identifié deux approches possibles. La première idée que nous avons eu fut d'agréger les données horaires en jour en utilisant une fonction d'agrégation adéquate. Pour déduire quelle fonction choisir, on s'est demandé s'il fallait en combiner plusieurs, s'il était pertinent d'agréger toutes les heures ou si certaines heures étaient plus importantes que d'autres.

Le seul moyen de répondre à ces questions fut d'essayer toutes les possibilités puis de les comparer selon un critère intéressant. Le critère qui nous a semblé le plus adéquat fut la corrélation avec notre variable à prédire "Ground Truth". Ainsi nous avons réalisé un code qui sélectionnait les x dernières heures de la journée, en faisant varier x de 1 à 24, puis appliqué une ou plusieurs agrégations telles que "Min", "Max", "Mean", "Var", "Std" ou "Max-Min". Une fois que nous avons agrégés nos heures en données journalières, nous pouvions désormais calculer une corrélation avec le Ground Truth de notre jeu de données d'apprentissage, et obtenir le graphique suivant :

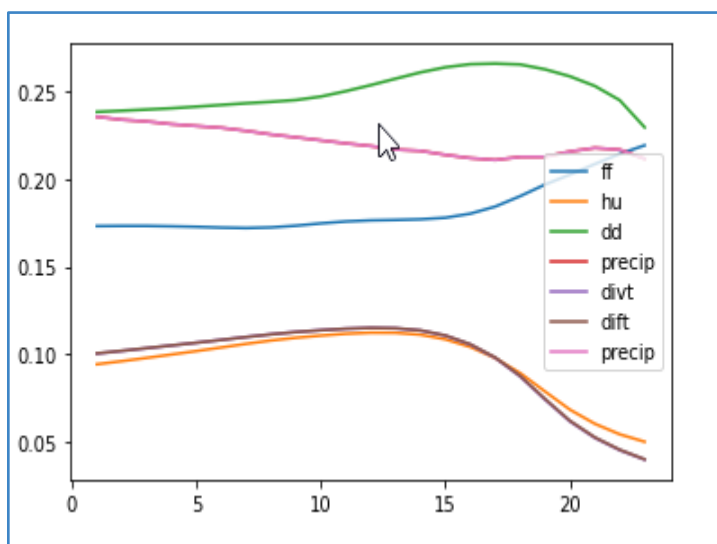


Figure 8 : Evolution de la corrélation des variables en fonction des heures conservées pour l'agrégation "Mean"

Nous avons pour chaque variable, l'évolution de la corrélation avec le Ground Truth à prédire en fonction des heures sur lesquelles ont été appliquées la fonction d'agrégation. A titre d'exemple, à l'abscisse 20 on peut lire les valeurs de corrélation pour chaque variable si l'on avait gardé seulement les heures supérieures ou égales à 20.

En comparant tous nos graphes obtenus (un pour chaque type d'agrégation) nous avons déterminé pour chaque variable le nombre d'heures à conserver et le type d'agrégation idéal.

Cette première approche fut simple à implémenter, et elle résolvait notre problème de différence d'échelle entre nos variables d'apprentissage et notre variable à prédire (heure vs jour). Cependant nous nous rendions bien compte que synthétiser 24 lignes de données en une seule pouvait présenter quelques inconvénients. En effet, il était important de comprendre que nos données suivent une évolution temporelle, et que ces évolutions peuvent être exploitées pour nous aider à mieux prédire le cumul de pluie, ce qui nous a poussé à tester une seconde approche, que nous avons aussi gardé par la suite.

### C.3. Utilisation des heures comme colonnes

Cette deuxième idée fut de sélectionner les 24 heures disponibles et de les utiliser comme variables explicatives. Nous passons donc de 24 lignes et 6 colonnes par jour et par station à une ligne et 6\*24 colonnes.

Nous avons ainsi conservé toutes les informations disponibles tout en gardant en tête que la plupart de ces données pouvaient apporter beaucoup de "bruit" pour nos modèles de machine learning. Nous savions alors que le modèle le plus efficace serait un mo-

dèle de Deep Learning, car s'il était bien construit il pourrait déterminer lui-même les dépendances entre les variables et identifier les motifs qui reviennent dans les données lorsqu'il est susceptible de prévoir le lendemain.

#### C.4. Création des features

A chaque création de feature, une analyse de corrélation avec la variable réponse est faite pour juger de la pertinence de la variable créée. Toutefois nous n'avons pas créé nos variables en les combinant aléatoirement entre elles. Afin de créer les variables les plus pertinentes, nous avons commencé par élargir nos connaissances en météorologie en se renseignant sur les principes physiques sous-jacents avec une question en tête "qu'est ce qui permet d'expliquer la survenue de précipitations ? ". Dès lors, nous avons entrepris une étape de recherche et de documentation dans un domaine que nous ne maîtrisons pas. Cette phase de recherche nous a également permis de réellement comprendre les données à notre disposition et les questions que l'on pouvait se poser : Comment est calculée l'humidité ambiante ? Que représente le point de rosée ? Pourquoi certaines variables sont-elles corrélées entre elles ? Par exemple, pour la variable dew point (Température de rosée) dont la formule est indiquée ci-dessous on trouve l'explication de son lien avec les variables température (t) et humidité (hu) :

$$T_r = \frac{b \alpha(T, \varphi)}{a - \alpha(T, \varphi)}$$

où :

$T_r$  désigne le point de rosée,

$T$  la température mesurée,

$\varphi$  l'humidité relative, entre 0 et 1 (soit 100%)

Cette compréhension de nos données a été essentielle à la mise en place de nouvelles features. Nous présenterons ici le cas de deux variables que nous avons créées et les idées qui ont motivé ces choix.

##### C.4.1. Direction du vent modifiée

La direction du vent est une variable numérique dont les valeurs sont comprises entre 0 et 360 degrés. Il est déjà important de noter ici que 360 et 0 représentent la même direction (direction Nord) bien qu'elles représentent les valeurs Min et Max de notre variable. Nous sommes ensuite partis de l'hypothèse qu'il devait exister une plage de valeurs pour lesquelles le risque de pluie était le maximum (par exemple, il faut que la direction du vent soit dirigé de l'océan atlantique vers la grande bretagne pour que les nuages formés au-dessus

de l'océan atteignent le continent et puissent causer des précipitations). Pour cela nous avons regardé l'écart en valeur absolue de nos données avec toutes les valeurs comprises entre 0 et 360 ( $\text{abs}(\text{wind\_direction} - x)$  pour  $x$  variant de 0 à 360) et nous avons gardé la valeur de  $x$  qui maximise la corrélation entre la nouvelle variable et le Ground Truth à prédire : 192. Ceci a donné lieu à une nouvelle variable :

$$\text{wind\_direction\_modified} = \text{abs}(\text{wind\_direction} - 192)$$

#### C.4.2. Combinaison température/humidité

Pour la création de cette variable nous sommes partis du postulat qu'il fallait considérer non pas la température et l'humidité ambiante individuellement mais comme deux variables couplées, (car elles sont liées dans le dew point) ainsi nous avons tenté de créer des variables résultantes d'une opération entre ces deux quantités (Température \* Humidité, Humidité / Température, etc...). Cependant, parmi toutes les combinaisons testées, nous n'avons pas pu obtenir de réel gain d'information, nous avons donc décidé d'ignorer ces variables nouvellement créées.

### D. Approche de modélisation sans deep learning

Pour la mise en place de notre modèle, il a fallu d'abord nous interroger sur la façon dont nous voulions prédire le cumul de pluie afin de trouver le modèle qui s'adapterait au mieux à notre jeu de données. Instinctivement, étant donné que la variable à prédire est quantitative, nous nous sommes intéressés à différents modèles de régression. Nous avons testé plusieurs modèles sans chercher directement à les optimiser avant de faire ressortir ceux qui étaient les plus pertinents pour notre prédiction.

#### D.1. Régression Linéaire

Le modèle de régression linéaire est un modèle prédictif simple et dont l'interprétation est facile. Nous avons choisi de commencer notre démarche d'analyse par ce modèle avec comme seul paramètre la normalisation des données.

Comme on s'y attendait étant donné que nous n'avons trouvé aucune corrélation linéaire entre une ou plusieurs variables explicatives et notre "ground\_truth" ce modèle ne nous donne pas des résultats très intéressants avec un score de 49.3615 mais il s'agissait d'une bonne base pour commencer.

## D.2. DecisionTreeRegressor & RandomForest

Un second modèle tout aussi classique mais qu'il convenait d'essayer est l'arbre de décision (voir ci-dessous). Nous nous attendions ici à un meilleur résultat en considérant que le cumul de pluie dépendait uniquement des autres variables météorologiques et des valeurs de chacune d'entre elles ce qui n'était pas forcément le cas au vu du score qui a avoisiné les 52.

```
regressor = DecisionTreeRegressor()
regressor.fit(Xtrain, Ytrain)
Ypredict = regressor.predict(Xtest)
scores = np.mean([np.abs((Ytest[i] - Ypredict[i])/(Ytest[i] + 1)) for i in range(len(Ypredict))])
print('Score : ', 100*scores)
```

Score : 52.497999178460084

Nous avons ensuite préféré utiliser le RandomForest qui en utilisant la méthode de rééchantillonnage nous préparait à avoir des résultats plus intéressants.

```
from sklearn.ensemble import RandomForestRegressor
regr = RandomForestRegressor(max_depth=40, random_state=0)
regr.fit(Xtrain, Ytrain)
Ypredict = regr.predict(Xtest)
scores = np.mean([np.abs((Ytest[i] - Ypredict[i])/(Ytest[i] + 1)) for i in range(len(Ypredict))])
print('Score : ', 100*scores)
```

Score : 43.05037364250913


Le score ayant diminué, cela nous a confortés à l'idée qu'utiliser du bootstrap pouvait être intéressant dans ce projet.

## E. Approche de modélisation avec deep learning

Les prédictions effectuées seulement à partir de modèles de machine learning ayant donné des résultats peu concluants, nous nous sommes intéressés à des modèles de deep learning.

### E.1. Démarche de développement

Nous avons développé deux types de réseaux de neurones : le LSTM et le MLP. Nous rappelons que le MLP est un type de réseau neuronal artificiel organisé en plusieurs couches au sein desquelles une information circule de la couche d'entrée vers la couche de sortie uniquement ; il s'agit donc d'un réseau à propagation directe (feedforward). Chaque couche



est constituée d'un nombre variable de neurones, les neurones de la dernière couche (dite « de sortie ») étant les sorties du système global.

Le LSTM est un réseau où chaque unité computationnelle est liée non seulement à un état caché  $h$  mais également à un état  $c$  de la cellule qui joue le rôle de mémoire.

Notre démarche de modélisation avec deep learning a consisté à établir un premier modèle baseline et procéder à ajuster différents paramètres de ce-dernier dans un objectif d'optimisation.

## E.2. Méthodes d'optimisation utilisées

Par rapport au baseline, nous avons utilisé plusieurs leviers d'optimisation que nous définissons brièvement ci-dessous.

### E.2.1. Stacking

La technique de stacking consiste à rendre le réseau plus profond en augmentant le nombre de couches. L'idée étant que plus on augmente le nombre de couches, plus le réseau de neurones apprend des notions complexes apportant plus d'informations. Concrètement, les premières couches permettront d'extraire des caractéristiques simples que les couches suivantes combinent pour former des concepts de plus en plus complexes et abstraits.

Pour illustrer le propos, dans notre modèle LSTM nous utilisons un stacking simple qui consiste à ajouter une couche LSTM de plus au modèle baseline qui ne possède qu'une seule couche.

### E.2.2. Drop Out

La prédiction de séries temporelles, principalement avec le modèle LSTM, est connue pour être impactée par un overfitting important lors des phases d'entraînement. C'est donc pour répondre à ce problème que nous avons utilisé la technique de dropout. Cette technique fait partie de la famille des méthodes de régularisation et est particulièrement pertinente dans le cadre de notre projet. Cette technique nous permet de désactiver temporairement certains neurones dans le réseau, ainsi que toutes ses connexions entrantes et sortantes. Le choix des neurones à désactiver est aléatoire. On attribue une probabilité  $p$  à tous les neurones qui déterminent leur activation.

Dans notre projet, nous utilisons  $p = 0.2$ , chaque neurone a une chance sur 20 d'être désactivé. À chaque epoch, on applique cette désactivation aléatoire. Cette technique diminue donc le surapprentissage en générant des modèles différents avec des configurations de neurones différentes à chaque itération

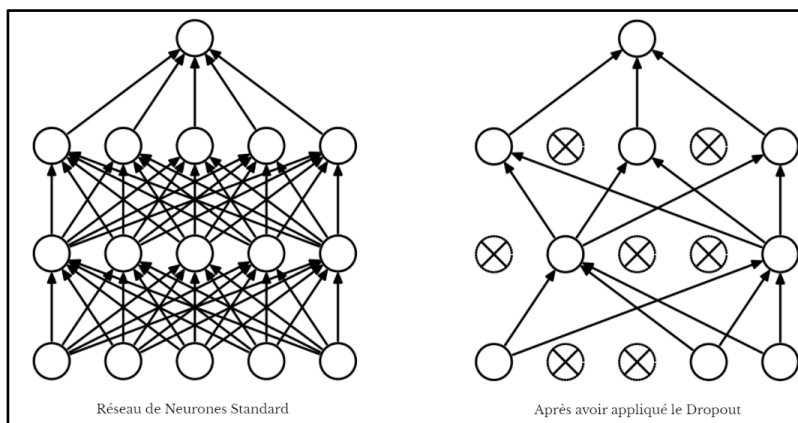


Figure 9 : Technique de drop out

### E.2.3. Batch Normalization

Cette technique nous permet de calculer les poids optimaux des neurones après que 'plusieurs exemples' aient été montrés au réseau et en normalisant les valeurs qui sont entrées dans les neurones. Cette normalisation se fait donc par batch en faisant en sorte que les valeurs d'entrée soient entre des minimums et des maximums acceptables.

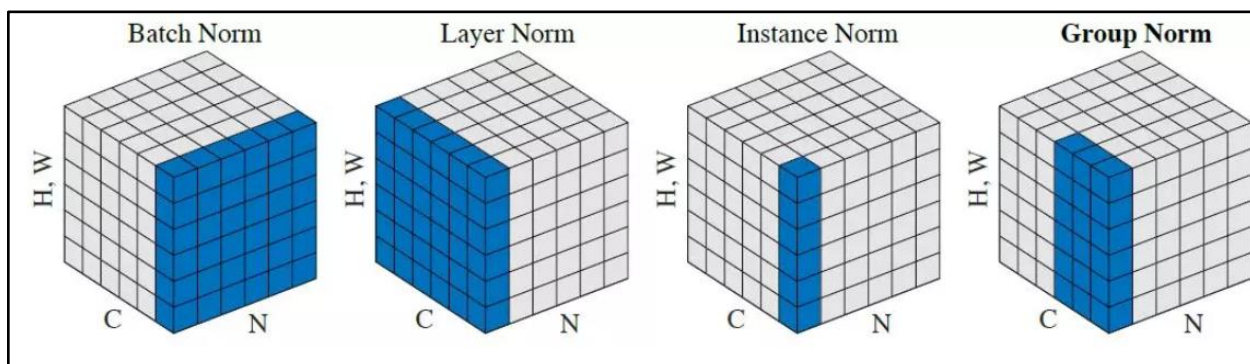


Figure 10 : Technique de batch normalization

### E.2.4. Weight Decay

Le weight decay est une régularisation qui ajuste à la baisse les poids au sein d'un modèle proportionnellement à la somme des carrés de leurs valeurs afin de contrer le surajustement. La régularisation L2 vise à ce que les poids dont la valeur est marginale, valeur positive ou négative très élevée ou très faible, se rapprochent le plus possible de 0 sans jamais l'atteindre.



### E.3. Modèle MLP

#### E.3.1. Fonctions Loss & Simple perceptron

Comme nous avons pu l'évoquer lors des précédentes parties, nos données initiales ont été traitées pour obtenir une granularité de données de l'ordre du jour, pour cela nous avons décidé de transformer nos lignes de données (sur les 24h journalières) en colonnes.

Ce traitement ayant engendré une forte augmentation du nombre de variables d'apprentissage, tout laissait supposer qu'un Modèle de Deep Learning serait plus adapté qu'un modèle de Machine Learning classique.

C'est porté par cette idée que nous avons décidé d'impairtir une grande partie de notre temps à l'élaboration et l'entraînement d'un réseau de neurones profond (MLP).

Nous avons d'abord commencé par créer la fonction MAPE en utilisant les outils de Tensorflow, dans le but de la déclarer en tant que fonction loss dans notre modèle.

```
def custom_loss_function(y_true, y_pred):
    return tf.math.reduce_mean(tf.abs(y_true-y_pred)/(y_true+1))*100
```


Figure 11 : Fonction Loss

```
def getSimplePerceptron(nb_classes=2):
    model = Sequential()
    model.add(InputLayer(input_shape=(24)))
    model.add(Dense(16))
    model.add(Dense(16, activation='sigmoid'))
    model.add(Dense(16, activation='sigmoid'))
    model.add(Dense(1, activation='sigmoid'))
    model.compile(loss=custom_loss_function, optimizer="adam", metrics=custom_loss_function)
    return model

model=getSimplePerceptron()
model=getSimplePerceptron()
batch_size = 256
epochs=5
model.fit(np.array(x_train).astype('f'), np.array(y_train).astype('f'), batch_size=batch_size, epochs=epochs, verbose=1)
pred = model.predict(np.array(x_test).astype('f'))
score(pred,np.array(y_test))
```

Figure 12 : Simple Perceptron

Suivant nos différents tests, notre couche d'entrée était constituée de 50 à 150 perceptrons (un par variable d'apprentissage) et nous avons fait varier ensuite le nombre de couches du modèles et le nombre de perceptrons par couche suivant une approche ascendante



En plus des différentes méthodes d'évaluations présentées en amont de cette partie, nous avons identifié les paramètres du modèle qu'il était intéressant de faire varier afin d'obtenir des prédictions de plus en plus fiables. Ceci est également valable pour notre réseau LSTM.

### E.3.2. Paramètres importants

Nous présentons ci-dessous les paramètres importants que sont la fonction d'activation, l'optimiseur, le nombre d'épochs et le batch size. En addition à une définition brève, nous indiquons comment nous avons utilisé les paramètres dans ce projet.

#### Fonction d'activation

Présentons premièrement la fonction d'activation. Rappelons qu'un perceptron n'est qu'une composition d'une fonction linéaire et d'une autre fonction d'activation (non linéaire ou linéaire) de  $\mathbb{R}$  dans  $\mathbb{R}$ . Il est parfois judicieux de tester plusieurs fonctions d'activations afin de savoir celle qui marche le mieux selon le modèle de nos données. Nous avons donc établi une liste de fonctions à tester que voici : [Sigmoid, Tangente Hyperbolique, Rectified Linear Unit (ReLU), Heaviside, identité]. Nous avons principalement travaillé avec la fonction ReLu car elle semblait plus robuste.

#### Optimiseur

L'optimiseur avec lequel nous travaillons par défaut est Adam qui est une extension de la descente de gradient stochastique. Son principal atout est qu'il peut adapter le Learning Rate en fonction des paramètres. Dans la pratique il fut l'optimisateur qui nous a permis d'obtenir les meilleures prédictions.

#### Nombre d'épochs

Le nombre d'epoch représente le nombre de fois que notre modèle aura vu l'ensemble de notre dataset pendant sa phase d'apprentissage, il peut être inutile d'avoir un nombre d'epoch trop élevé car le modèle risque fortement de converger après un certain nombre d'itérations, cependant le nombre doit être suffisant pour que les paramètres du modèles soit bien ajustés

#### Batch\_size

Pour ce paramètre, nous avons dû trouver le nombre le plus adapté afin d'obtenir les meilleures optimisations à chaque epochs. Si le nombre est trop faible, nous pouvons avoir une moins bonne estimation du gradient mais une meilleure utilisation de la mémoire et si

le nombre est très élevé nous aurons moins de phases de forward-back propagation mais une meilleure approximation du gradient.

Les résultats de notre réseau MLP seront présentés et analysés dans la partie 3 de ce rapport.

## E.4. Modèle LSTM

### E.4.1. Préparation des données pour LSTM

Nous définissons le problème d'apprentissage supervisé comme une prédiction de du cumul de précipitations au jour actuel (t) compte tenu de la mesure des conditions météorologiques au pas de temps précédent (t-1).

Nous avons construit des fonctions retraitant les données de façon à les présenter au format nécessaire pour le réseau LSTM. La figure suivante montre les données après re-traitement.

↳	var1(t-1)	var2(t-1)	var3(t-1)	var4(t-1)	var5(t-1)	var6(t-1)	var7(t)
1	0.160655	0.432794	0.691741	0.884764	0.338439	0.002139	0.125134
2	0.330084	0.484125	0.711108	0.821212	0.542413	0.036364	0.010695
3	0.222023	0.417495	0.678463	0.866162	0.558652	0.064171	0.059893
4	0.176356	0.467880	0.684296	0.761700	0.304853	0.124064	0.034225
5	0.072024	0.481528	0.724245	0.886322	0.402903	0.059893	0.008556

*Figure 13 : Données retraitées pour le LSTM*

Nous rappelons que dans la phase de preprocessing nous avons appliqué une normalisation.

Plus bas, nous comparons le score MAPE de plusieurs types de LSTM. Notre stratégie est de sélectionner le type présentant le meilleur score. Par la suite, nous procédons à optimiser le type de LSTM sélectionné avec différentes techniques.

### E.4.2. Vanilla LSTM : Modèle Baseline

Notre premier modèle est une seule couche LSTM suivie d'une couche dense. Dans cette couche, nous définissons en premier le nombre de neurones. Ce nombre sera fixé à 256 pour tous les modèles LSTM de façon à ce que ces derniers soient comparables.

Nous définissons par la suite le nombre de pas de temps également appelé 'lookback period' qui est toujours égale à 1. Finalement, nous paramétrons le nombre de features à 6.

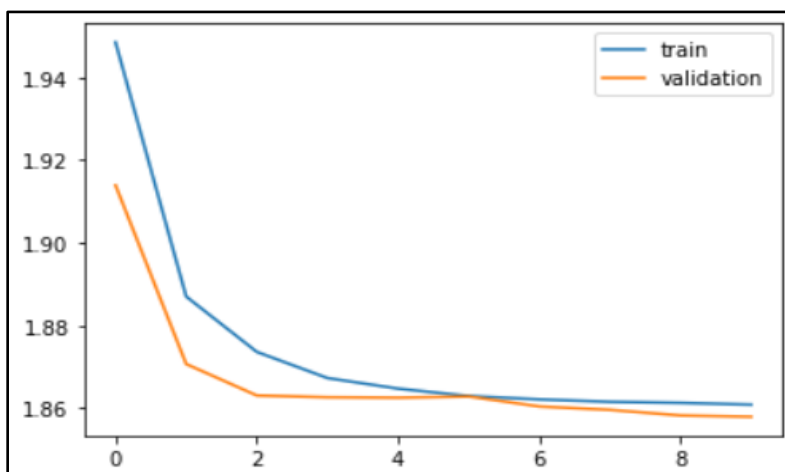


Figure 14 : Entraînement du Vanilla LSTM sur 10 epochs

Le vanilla model obtient un score MAPE de 18.8882.

#### E.4.3. Stacked LSTM

Nous utilisons un stacking simple qui consiste à ajouter une couche LSTM de plus au modèle baseline qui ne possède qu'une seule couche.

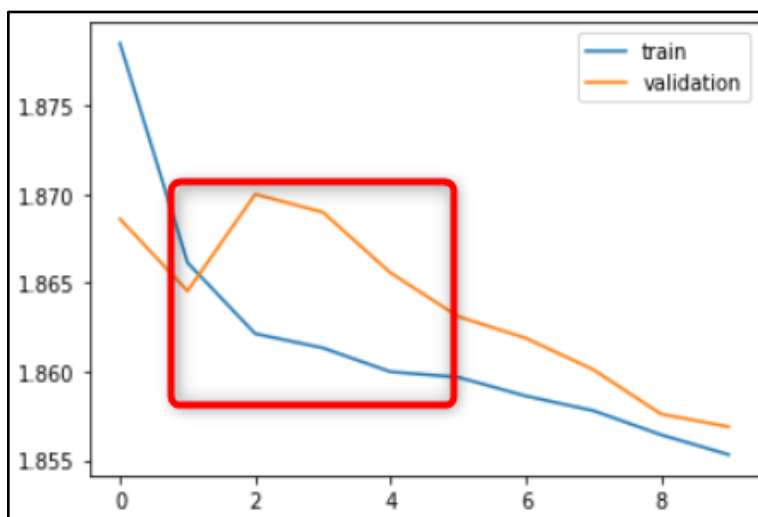


Figure 15 : Entraînement du Stacked LSTM sur 10 epochs

Un overfitting important commence dès l'epoch 1. Le stacked model obtient un score MAPE de 19.0076

#### E.4.4. Bidirectional LSTM

Dans ce modèle, nous partons du principe qu'il peut être avantageux de permettre au modèle LSTM d'apprendre la séquence d'entrée à la fois vers l'avant et vers l'arrière et de concaténer les deux interprétations.

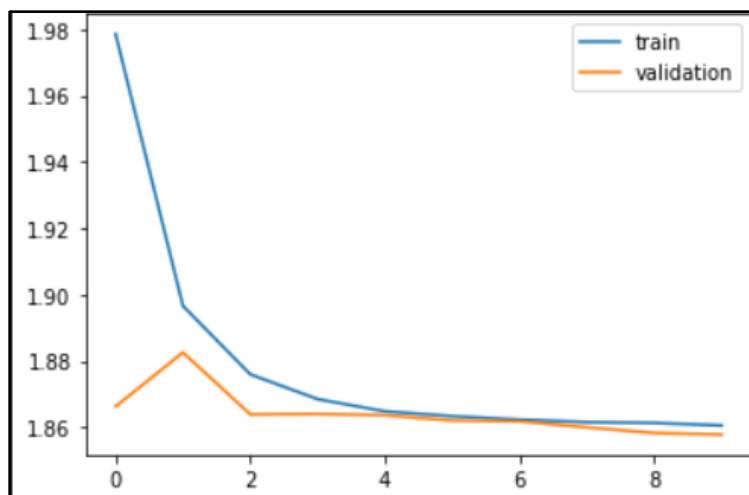


Figure 16 : Entraînement du Bidirectional LSTM sur 10 epochs

Le bidirectional model obtient un score MAPE de 18.894. Nous n'observons pas d'overfitting.

#### E.4.5. Best LSTM avec régularisation

A ce niveau, nous visons à prendre le meilleur type de modèle LSTM parmi les trois explicités plus haut. Par la suite, nous appliquons les deux méthodes de régularisation que l'on a définies dans la section 'Méthodes d'optimisation'.

Le meilleur LSTM est le Vanilla model. Les deux autres modèles empirent légèrement le score. Ainsi, avoir un réseau plus profond ou plus sophistiqué n'est pas gage de performance. Avec un drop out de 0.2 le score passe à 18.8419 et avec un weight decay cela passe à 18.863

#### E.4.6. Bilan de performance des modèles LSTM

A ce niveau, nous faisons un bilan de performance. Nous nous basons sur ce bilan pour tirer des conclusions sur nos modèles et notre démarche de deep learning LSTM en générale.

Modèles	Scores MAPE
Vanilla LSTM	18.8882
Stacked LSTM	19.0076
Bidirectional LSTM	18.8940
Vanilla + Drop out	18.8419
Vanilla + Weight Decay	18.8603

Nous observons que le modèle 'Vanilla + Drop Out' présente le modèle le plus performant. Il semble que la simplicité du modèle est un avantage. En effet, il semble que pour ce cas d'étude plus le modèle est simple plus le risque de surapprentissage est faible. Sans les prédictions Météo France, le dimensionnement des données d'entraînement reste trop simple pour l'architecture très complexe du LSTM. Dans ce contexte, le LSTM apprend alors non seulement les caractéristiques des données mais aussi le bruit sur le dataset d'entraînement. Ainsi, sa capacité de généralisation sur les données test est fortement compromise et le modèle fait un 'overfitting'. Ceci car les données test ont un dimensionnement logiquement différent des données d'entraînement.

## Partie 2 : En utilisant les prédictions Météo France

### A. Présentation des données disponibles

Dans cette partie nous exploitons des fichiers netCDF files. Le NetCDF (network Common Data Form) est un format de fichier qui permet de stocker des données scientifiques multidimensionnelles (variables), telles que la température, l'humidité, la pression, la vitesse et la direction du vent.

Ici, les fichiers 2 dimensionnels sont présentés sous les 2 systèmes de prévision de météo France que sont les systèmes ARPEGE et AROME. Les fichiers 3 dimensionnels sont présentés sous ARPEGE seulement.

### B. Interpolation des données aux stations météo

Les prédictions météo France sont présentées sur un “regular grid” en latitude et longitudes. L’enjeu est donc d’aligner ces prédictions avec la station d’observation qui est la plus proche.

Notre approche d’interpolation est basée sur deux étapes :

1. Dans un premier temps nous avons utilisé la formule haversine qui permet de calculer la distance entre deux points du globe. Ainsi nous avons calculé la distance entre les points de prédiction Météo France et les coordonnées des stations, et nous avons pris la station la plus proche à chaque point de prédiction.
2. Puis dans un deuxième temps, en se basant sur le numéro de chaque de station (number\_sta), nous avons fait la jointure entre les stations les plus proches des prédictions Météo France et les données d'entraînement des stations.

## C. Différentes approches

Ayant déjà présenté la plupart des modèles en première partie, nous verrons ici les différentes approches que nous avons essayées à partir des modèles de machine et deep learning.

### C.1. Diviser la prédiction en deux

La première approche consiste à diviser le processus de prédiction en deux parties afin de faire un premier tri où l'on détecte d'abord les jours où on est sûr qu'il ne pleut pas et d'ensuite essayer de prédire le cumul pour les jours restants.

En effet, cette approche nous a paru pertinente pour deux raisons :

- Certaines de nos prédictions du cumul de pluie sont aux alentours de 0 et non à 0 stricte ce qui fait augmenter l'erreur de prédiction alors qu'en soi le modèle a détecté que ce jour-là il ne pleuvait pas, il ne l'a simplement pas exprimé par une valeur nulle mais très faible.
- Comme cela va se confirmer par la suite, on a remarqué que la situation précédente était assez fréquente ce qui sous-entend que détecter une grande partie des jours où il ne pleut pas est relativement faisable.

#### C.1.1 Prédire les jours non pluvieux

Pour cette première phase de prédiction nous avons testé plusieurs types de modèles différents. Nous avons commencé par le plus intuitif : un modèle de classification.

Pour mettre en place un modèle de classification et pouvoir l'entraîner, il a fallu en amont paramétrer la variable à prédire "pluie" qui indique s'il a plu ou non, pour cela nous nous sommes basés sur la quantité de pluie : Après plusieurs valeurs testées entre 0 et 1 on a estimé qu'un cumul de pluie inférieur à 0.4 correspondrait à une journée non pluvieuse et nous avons créés notre nouvelle variable binaire par conséquent.

Une fois cette préparation effectuée, il s'ensuit une procédure de prédiction classique à partir d'un RandomForestClassifier où nous avons gardé le modèle initial avec les paramètres par défaut :



```

Entrée [34]: xg_cl_default = xgb.XGBClassifier(max_depth=10)
              xg_cl_default.fit(x_train,y_train)
              pred2 = xg_cl_default.predict(x_test)
              conf = confusion_matrix(y_test, pred2)
              (conf[0,0]+conf[1,1])/np.sum(conf)

[19:30:07] WARNING: C:/Users/Administrator/workspace/xgboost-win64_release_1.5.1/src/learner.cc:1115: Starting in XGBoost 1.3.0, the default evaluation metric used with the objective 'binary:logistic' was changed from 'error' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behavior.

Out[34]: 0.8724592707525214

```

Ceci qui nous a donné la matrice de confusion suivante :

```

Out[35]: array([[22186, 2362],
               [ 2570, 11552]], dtype=int64)

```

Le taux de jours non pluvieux détectés a été de 87%. Nous avons ensuite essayé un modèle de régression, ici nous avons d'abord prédit le cumul de pluie et ensuite nous avons déduit de s'il s'agissait d'un jour de pluie ou non (cumul > 0.4 ou non).

Nous sommes partis sur le LogisticRegression où nous avons encore une fois gardé le modèle par défaut qui nous a donné la matrice de confusion suivante :

```

Out[37]: array([[23230, 1318],
               [ 4699, 9423]], dtype=int64)

```

Le taux de jours non pluvieux détectés a été de 84%. Enfin, nous nous sommes essayés à un modèle de deep learning que nous avons essayé d'optimiser et dont le résultat a été le suivant :

```

Entrée [101]:
def getSimplePerceptron(nb_classes=2):
    model = Sequential()
    model.add(InputLayer(input_shape=(10)))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(512, activation='relu'))
    model.add(Dense(1, activation='sigmoid'))
    opt = SGD(learning_rate=0.1, momentum=0.9)
    model.compile(loss='binary_crossentropy', optimizer="adam", metrics=['accuracy'])
    return model

model=getSimplePerceptron()
model=getSimplePerceptron()
batch_size = 256
epochs=5
model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs, verbose=1, validation_data=(x_test, y_test))


...

Entrée [75]: conf = confusion_matrix(y_test, pred_nn)
              (conf[0,0]+conf[1,1])/np.sum(conf)

Out[75]: 0.8544091026635635

```

Le taux de jours non pluvieux détectés a été de 85%. Le modèle ayant donné une accuracy la plus élevée a été le RandomForestClassifier que nous avons donc conservé.



Une fois cette première phase ayant permis d'écarter les jours de non pluie en considérant que le cumul de pluie était nul, nous nous sommes attaqués aux jours restants. A noter que les jours restants ne sont pas forcément des jours où il a plu étant donné que nous n'avons pas détecté tous les jours non pluvieux dans la première étape.

### E.1.2 Prédire les stations restantes

Pour les stations restantes on a d'abord fait une prédiction classique avec notre modèle de deep learning qui nous avait déjà donné les meilleures prédictions précédemment.

Nous avons également tenté une autre alternative où cette fois-ci un jour qui a été prédit comme non pluvieux n'obtient pas par défaut un cumul nul mais correspondra à la moyenne entre un cumul nul et la prédiction faite par le modèle de deep learning (autrement dit la prédiction divisée par 2). Ainsi ici, à contrario de la première idée, les prédictions sont de nouveau faites sur l'ensemble des stations. Nous avons fait cela avec comme idée de réduire le taux d'erreur effectué en estimant qu'un jour était non pluvieux alors qu'il l'était.

Au fur et à mesure que nous optimisons les paramètres de notre modèle, la MAPE a commencée à converger, nous avons donc voulu essayer d'autres types d'approche plutôt que d'essayer d'optimiser encore plus notre modèle et plonger dans du sûr-apprentissage.

### C.2. Un modèle par zone

La seconde idée que nous avons tentée a été de considérer que la région bretonne sur laquelle nous nous concentrons pouvait avoir des particularités météorologiques en fonction des zones. En effet, certaines zones où l'altitude est plus élevée peuvent mener à des pluies plus fréquentes et abondantes. Il en est de même pour des régions situées au bord de mer qui sont souvent plus humides bien qu'il ne pleuve pas.

Ainsi nous en sommes arrivés au fait de considérer plusieurs régions distinctes et pour chacune de ces régions géographiques d'entraîner un modèle sur les stations qui s'y trouvent.

Pour cela nous avons tout d'abord regroupé les stations en 20 clusters à partir d'un Kmeans qui nous a donné la répartition suivante :

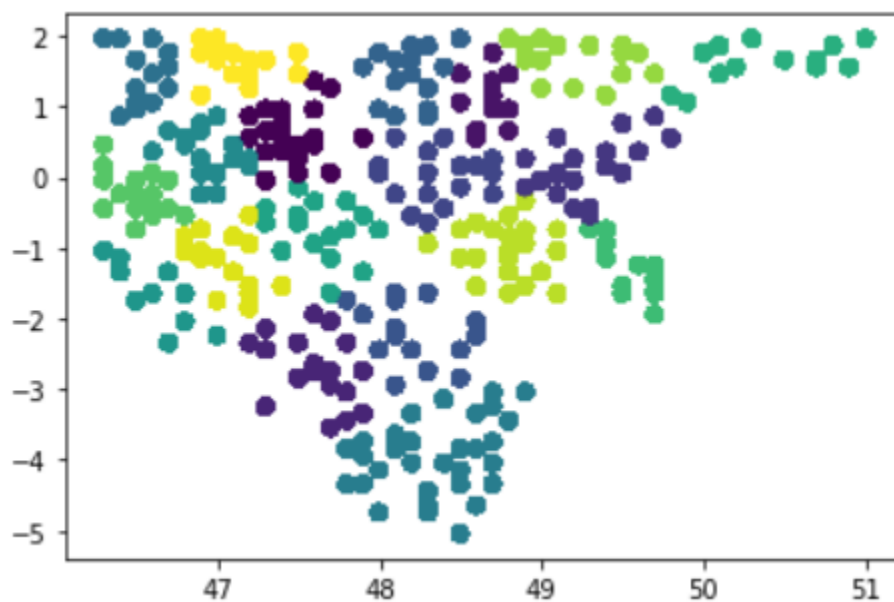


Figure 17 : Répartition des clusters de stations

Sur chacun de ces clusters nous avons entraîné le modèle de deep learning précédent ce qui nous a donné au final 20 modèles distincts qui nous espérons seraient plus sensibles aux particularités de chaque région.

Pour la prédiction du cumul de pluie, nous avons parcouru chacune des stations afin de l'associer au cluster de stations d'entraînement le plus proche et nous lui avons prédit son cumul de pluie à partir du modèle associé.

Malheureusement le score que nous avons obtenu était moins bien que le modèle entier. Par manque de temps nous n'avons pas pu optimiser chacun des 20 modèles ce qui peut expliquer que cette méthode n'ait pas été convaincante avec un score autour de 30.

### C.3. Moyenne des modèles

Le fait d'optimiser chacun des 20 modèles paraissait assez compliqué et long à effectuer mais nous voulions tout de même pouvoir mélanger plusieurs modèles. Ainsi étant donné que nous avons accès à l'ensemble des prédictions effectuées durant le projet avec les différentes méthodes, nous avons pu conserver les 3 modèles les plus pertinents et les prédictions associées. Nous avons fait une moyenne des prédictions obtenues pour chaque station et obtenu notre score de 25.11643 sur le leaderboard privé qui a été notre meilleur score et par conséquent la prédiction qui a été la moins soumise aux erreurs.

Cette première place s'est conservée lorsque nous sommes passés sur le leaderboard privé malgré que le score ait augmenté.

## Partie 3: Bilan

### A. Analyse des résultats et stratégie finale

Notre approche itérative a débuté par l'utilisation de modèles de machine learning classique sans utiliser les prédictions de Météo-France. Cette première approche a la plus mauvaise performance car les modèles ont une architecture beaucoup trop simple pour prendre en compte les dépendances et caractéristiques des données sur une temporalité longue.

Ainsi, toujours sans les prédictions Météo-France, nous avons opté pour le deep learning. Nos modèles de deep learning affichent de très bons résultats sur les données d'entraînement. Cependant, les résultats sur les données test sont moins pertinents.

#### **Comment expliquer la différence de performance entre les données train et test ?**

Au regard de nos compétences et nos connaissances, nous présentons ici nos hypothèses d'explication.

Une piste expliquant la différence, particulièrement pour le LSTM, est un dimensionnement différent des données test par rapport aux données train. En effet, dans les données test l'aspect chronologique n'est pas respecté ou aligné avec le dimensionnement temporel dans les données train. Ceci posera un problème conséquent dans la mesure où LSTM est basé sur la notion temporelle de 'lookback' pour chaque neurone.

Une deuxième piste est le surapprentissage. Nous rappelons que l'on parle de surapprentissage quand un modèle a trop appris les particularités de chacune des observations fournies en exemple. Il présente alors un taux de succès très important sur les données d'entraînement, au détriment de ses performances générales réelles. Ceci devient un problème conséquent quand on sait que le but d'un apprentissage est de permettre au réseau de tirer du dataset initial des généralités et de pouvoir les appliquer à de nouvelles données par la suite.

## Comment expliquer le surapprentissage dans nos modèles ?

Le surapprentissage provient d'un manque d'adéquation entre la complexité de nos modèles et le dimensionnement du jeu de données. Particulièrement le LSTM a une architecture très complexe qui lui permet d'apprendre les caractéristiques pertinentes du dataset mais également le bruit ou information non-pertinente.

## Quelles solutions avons-nous à notre disposition ? lesquelles avons-nous utilisé ?

Les solutions qui nous viennent donc à l'esprit sont soit de simplifier le modèle soit de réussir à obtenir un plus grand jeu de données. Nous avons utilisé les deux solutions.

Premièrement, nous avons utilisé les données Arpege pour enrichir l'entraînement. Deuxièmement, nous avons utilisé des méthodes de régularisation comme le drop out pour simplifier nos modèles de deep learning. Nous rappelons que toutes nos méthodes d'optimisation sont expliquées dans la section C plus haut. Toutes les méthodes d'optimisation dans cette section ont été testées de façon itérative jusqu'à trouver la combinaison finale utilisée dans notre soumission finale.

Tous les éléments plus haut viennent en description et explication du schéma suivant retraçant nos modèles et leurs scores dans l'ordre chronologique. Nous allons à la suite du schéma, apporter des explications plus détaillées des tendances observées sur le leaderboard privée et public.

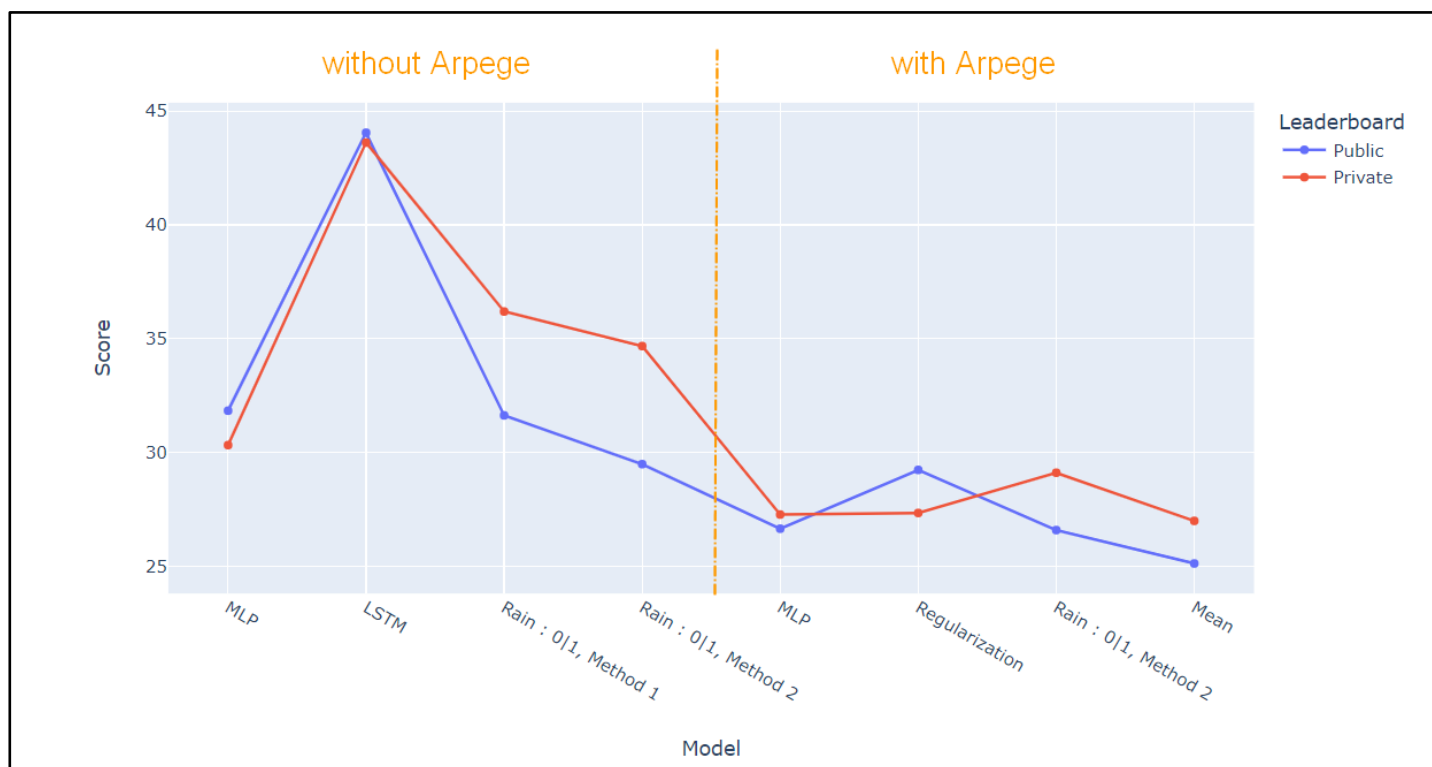



Figure 18 : Bilan général des scores



En considérant seulement les 8 meilleures prédictions que nous avons faites, nous remarquons tout d'abord, que ce soit pour le leaderboard privé ou public, une tendance générale des scores qui diminue au fur et à mesure des soumissions. Ceci illustrant la pertinence de nos modèles au fil du projet.

Comme on le voit, les modèles les plus pertinents mettent en jeu du deep learning, les méthodes de machine learning à elles seules n'ont pas donné de résultats satisfaisants avec nos features. En s'intéressant seulement au leaderboard public auquel nous avons accès pendant la compétition, on constate que le modèle LSTM est celui qui a le score le moins bon.


La stratégie finale a consisté à faire une moyenne des prédictions de plusieurs méthodes utilisées. Cette stratégie fait référence aux méthodes d'ensemble comme le random forest qui évite le sur-apprentissage et stabilise l'erreur de généralisation. Comme avant chaque soumission nous choissions le modèle et nous réglions les paramètres dans le but d'améliorer le score de prédiction, nous sommes partis du principe que s'il y'a eu des modèles qui ont sur-appris ou des modèles qui ont sous-appris, faire la moyenne de toutes nos prédictions nous permettrait de corriger l'erreur globale.

A la fin de la compétition on a constaté que l'ensemble de nos modèles ont été moins performants avec les données du leaderboard privé, là aussi il semblerait que nos modèles n'étaient pas très stables en changeant de jeu de données.

## B. Conclusion

Nos résultats et notre démarche relèvent d'une approche algorithmique et itérative visant à trouver l'estimateur du maximum de vraisemblance. Dans cette démarche d'optimisation, le plus grand défi technique a été celui du surapprentissage. Nous avons utilisé plusieurs méthodes pour surmonter le problème comme expliqué plus haut.

Pour les travaux futurs, nous conseillons un travail poussé pour créer une adéquation entre la complexité des modèles et le dimensionnement du jeu de données. De plus, nous préconisons une stratégie qui se base sur une combinaison de machine et deep learning et opérer de façon séquentielle. En effet, nous avons trouvé beaucoup plus pertinent de séparer nos prédictions en deux en prédisant d'abord les jours non pluvieux qui écartent ainsi un grand nombre de stations et en entraînement ensuite un modèle qui prédirait le cumul de pluie des stations restantes.



Par manque de temps et de compétence nous n'avons pas pu développer la technique consistant à faire un modèle de prédiction par zone géographique. Cependant, nous la sentons intéressante à développer dans la mesure où les spécificités de chaque secteur de la région bretonne pourrait être prises en compte : séparer les zones en bord de mer, les zones à altitudes très différentes, ....

Nous n'avons également pas pu exploiter complètement les réseaux de neurones hybrides CNN-LSTM qui auraient peut-être répondu au défi du surapprentissage. Une autre piste à exploiter serait éventuellement un réseau GAN pour créer des données synthétiques imitant les données de base sans les copier. Ceci pour agrandir le volume des données d'entraînement et réduire l'inadéquation entre des modèles trop complexes d'un côté et un dimensionnement trop faible des données de l'autre.