INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES TOULOUSE

INP TOULOUSE ENSEEIHT

RAPPORT DU PROJET « MACHINE LEARNING »

ÉLABORÉ PAR :

Dorian VOYDIE
Thomas FRAMERY

Dans le cadre du Mastère Spécialisé VALDOM

I. Table des matières

II. Introduction

Dans le cadre de l'UE « Machine Learning » du Mastère VALDOM, nous avons pu travailler sur un projet en Data Science en collaboration avec METEO FRANCE. En effet, grâce aux données fournies par l'organisme météorologique, notre objectif était de prédire la quantité cumulée de pluie tombée sur une certaine station au sol. Par conséquent, nous avons élaboré une stratégie de conception pour obtenir la meilleure précision dans nos prédictions. Le jeu de données a été déposé sur Moodle.

III. Transformation des datasets

Tout d'abord, cette transformation se subdivise en plusieurs étapes :

1. Écrire les fonctions de preprocessing
2. Appliquer ces fonctions aux datasets associés
3. Merge les 2 datasets résultants pour construire le trainset
4. Introduire les données AROME et ARPEGE (fournies par MeteoNet)
5. Introduire les coordonnées des stations pour plus de précision

Conformément à la stratégie mentionnée ci-avant, examinons les fonctions de preprocessing.

```python
def X_train_preprocessing(df):

    # Récupération mois/jour/heure par ligne
    df['date'] = pd.to_datetime(df['date'])
    df['month'] = df['date'].dt.month
    hour = df["Id"].str.split("_", n = 2, expand = True)[2]
    df['hour'] = hour.astype(int)
    day = df["Id"].str.split("_", n = 2, expand = True)[1]
    df['day'] = day.astype(int)

    # Création d'un Id quotidien pour pouvoir merge X_train avec Y_train
    df['Id_merge'] = df['number_sta'].astype(str).str.cat(day,sep="_")

    # Mise en ordre des features
    df = df[['dd','hu','td','t','ff','precip','month','Id','Id_merge','number_sta','hour','day']]
    df['precip'] = df['precip']*24
    df['month'] = df['month'].astype(int)

    # Tri des features : N° Station / Jour / Heure
    df = df.sort_values(["number_sta","day",'hour'])
    df = df.drop(['hour','day'],axis=1)

    return df
```

Création de colonnes « jour », « mois », « heure »

Création d'une colonne pour merge avec Y_train

Mise en ordre des colonnes et au format int

Tri des lignes chronologique

Attribut de jointure

	dd	hu	td	t	ff	precip	month	Id	Id_merge	number_sta
0	200.0	91.4	277.97	279.28	3.05	0.0	1	14066001_0_0	14066001_0	14066001
1	190.0	91.4	277.45	278.76	2.57	0.0	1	14066001_0_1	14066001_0	14066001
2	181.0	91.7	277.02	278.27	2.26	0.0	1	14066001_0_2	14066001_0	14066001
3	159.0	93.0	276.95	277.98	2.62	0.0	1	14066001_0_3	14066001_0	14066001
4	171.0	95.9	276.72	277.32	2.99	0.0	1	14066001_0_4	14066001_0	14066001

Ici, l'enjeu était de transformer le dataset X_train pour que l'identifiant soit composé du numéro de station ainsi que du jour. Ce nouvel identifiant, nommé « Id_merge » a servi d'attribut de jointure avec le dataset Y_train.

L'autre enjeu concernait la colonne « precip ». Nous avons décidé de la multiplier par 24 mais cette opération prend tout son sens après la jointure des datasets et plus précisément lors du calcul de la moyenne. Mais avant cela, visualisons la façon dont le dataset Y_train a été mis en forme

```python
def Y_preprocessing(df):

    df = df.drop(['date','number_sta'],axis=1)
    df = df[['Id','Ground_truth']]
    df['Id_merge'] = df['Id']
    df = df.dropna()

    return df
```

On ne garde que le label Ground_truth, on crée l'Id_merge pour merge avec X_train et on enlève les lignes NaNs ➔ perte = 3%

	Ground_truth	Id_merge
0	3.4	14066001_0
1	0.5	14126001_0
2	3.4	14137001_0
3	4.0	14216001_0
4	13.3	14296001_0

Attribut de jointure

Ensuite, nous devons donc effectuer la jointure des 2 datasets et créer ainsi le trainset.

```python
trainset = X_train_df.merge(Y_train_df,how="inner",on="Id_merge")
trainset['Id'] = trainset['Id_merge']
trainset = trainset.drop(['Id_x','Id_merge','Id_y'],axis=1)
trainset = trainset.groupby("Id").mean()
trainset = trainset.reset_index()
trainset['month'] = trainset['month'].astype(int)
trainset = trainset.merge(coords,how="inner",on="number_sta")
```

Jointure des datasets

Moyenne des attributs de X_train sur 24h car X_train est décomposé en heures et Y_train en jours.

Jointure des coordonnées de stations

Moyenne des features sur la journée

Somme des précipitations sur un jour

	Id	dd	hu	td	t	ff	precip	month	number_sta	Ground_truth	lat	lon	height_sta
0	14066001_0	146.500000	88.591667	278.514583	280.333750	3.913750	0.2	1	14066001.0	3.4	49.334	-0.431	2.0
1	14066001_1	205.625000	82.300000	279.997500	282.936667	8.041250	3.4	1	14066001.0	11.7	49.334	-0.431	2.0
2	14066001_10	209.541667	86.750000	277.497917	279.557917	5.408750	6.0	1	14066001.0	1.0	49.334	-0.431	2.0
3	14066001_100	134.958333	76.408333	277.944583	282.112917	4.296250	11.6	4	14066001.0	5.6	49.334	-0.431	2.0
4	14066001_101	167.208333	88.745833	281.003333	282.805000	1.754583	5.6	4	14066001.0	3.2	49.334	-0.431	2.0

Features | Target | Features

Si nous suivons notre stratégie mentionnée en début de partie, elle comporte l'introduction des données AROME (données supplémentaires) fournies par METEO FRANCE.

L'identifiant d'un dataset AROME étant composé du numéro de station ainsi que du jour, nous n'avons pas de modifications particulières à apporter pour l'intégrer dans notre trainset.

```python
def train_imputation(df):

    # Version 1 : DropNaNs
    # df = df.dropna()

    # Version 2 : IterativeImputer
    temp = df[["Id","number_sta","month","Ground_truth"]]
    imp_mean = IterativeImputer(random_state=0)
    df = pd.DataFrame(imp_mean.fit_transform(df[["ff","t","td","hu","dd","precip","lat","lon","height_sta"]]))
    df = pd.concat([temp,df],axis=1)
    df.columns = ["Id","number_sta","month","Ground_truth","ff","t","td","hu","dd","precip","lat","lon","height_sta"]

    # Version 3 : KNNImputer
    # temp = df[["Id","number_sta","month"]]
    # imputer = KNNImputer(n_neighbors=2)
    # df = pd.DataFrame(imputer.fit_transform(df[["ff","t","td","hu","dd","precip","lat","lon","height_sta"]]))
    # df = pd.concat([temp,df],axis=1)
    # df.columns = ["Id","number_sta","month","ff","t","td","hu","dd","precip","lat","lon","height_sta"]

    return df
```

Nous avons essayé plusieurs imputations, d'abord dropna, puis un IterativeImputer (BayesianRidge) afin d'aller chercher les valeurs manquantes dans des lignes similaires. Le KNNImputer étant très long nous ne l'avons pas testé

```python
# Imputation et encodage
trainset = train_imputation(trainset)
encoder = LabelEncoder()
trainset["number_sta"] = encoder.fit_transform(trainset["number_sta"].astype(int))
trainset.head()
```

Encodage de la seule colonne qualitative : number_sta

Il ne nous reste plus qu'à extraire une station de ce jeu de données pour avoir le jeu de données final. De là, on peut le séparer un jeu d'entrainement (dataapp) et un jeu de test (datatest).

IV. Analyse exploratoire des données

Les jeux de données

Dans cette première partie, nous allons analyser le contenu des données (la forme, les dimensions, la cardinalité, …) proposées par Météo France. Cette première phase est cruciale car elle permet de prendre en main les données et de savoir véritablement ce que l'on manipule.

Dans ce projet, nous avons décidé de travailler sur la station portant l'identifiant 14066001.

A partir de cela, nous avons créé un jeu de données d'entraînement et un autre de test.

	Id	dd	hu	td	t	ff	precip	month	number_sta	Ground_truth	ws	p3031	u10
0	14066001_158	300.916667	90.050000	286.600417	288.247083	2.550000	0.2	6	14066001	0.0	3.455806	307.139567	2.638302
1	14066001_413	156.916667	92.275000	277.223750	278.399167	1.950000	0.2	2	14066001	0.2	2.484051	210.949178	0.872938
2	14066001_207	240.958333	77.779167	285.496250	289.606667	2.724583	0.2	7	14066001	0.0	3.394090	255.705485	2.150585
3	14066001_503	308.750000	92.241667	282.962083	284.201667	4.929167	11.1	5	14066001	3.2	3.959870	225.794413	1.918382
4	14066001_175	243.791667	82.520833	286.026250	289.084167	2.829167	0.0	6	14066001	0.8	3.277412	191.829834	1.609237

Target

Toutes les colonnes ne sont pas affichées, notamment les features Arpege.

Ici, ce qu'il a été important de remarquer, ce sont les informations contenues au niveau de l'identifiant du ligne. Par exemple, pour la première ligne, l'identifiant est :

Numéro de station ← 14066001_158 → Jour

Ainsi, une ligne peut se lire comme suit : « Pour cette station, ce jour, nous avons ces différents paramètres météorologiques ».

Statistiques des jeux de données (apprentissage + test)

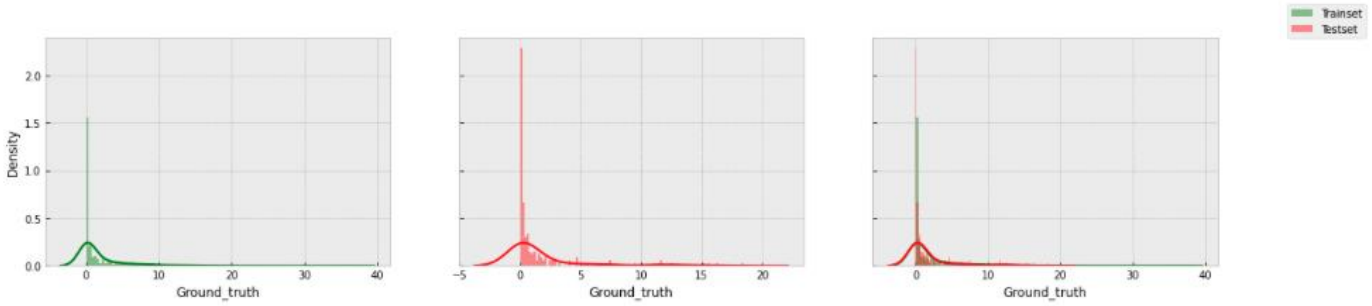
	Dataset d'apprentissage	Dataset de test
Number of Variables	30	30
Number of rows	543	181
Variable types	Quantitative : 29 Qualitative : 1 (Id)	Float : 29 Int : 1 (Id)

De manière générale, on a observé que 4 features avaient des valeurs constantes (variance nulle) :

- Le numéro de station (number_sta)
- La taille de la station (height_sta)
- La position d'un point de vue latitude de la station (lat)
- La position d'un point de vue longitude de la station (lon)

Ceci est complètement normal puisque l'objectif du projet est de travailler sur une station en particulier. Étant donné la variance nulle, nous avons décidé d'éliminer ces variables car elles n'influent pas sur la prédiction. Par défaut, la variable « month » était qualitative mais nous avons pris la décision de l'encoder pour qu'elle devienne quantitative avec 12 valeurs différentes correspondant aux 12 mois de l'année.

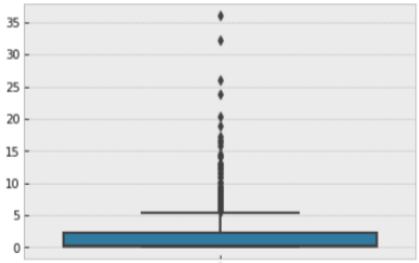
Comparaison des distributions de la target (apprentissage + test)



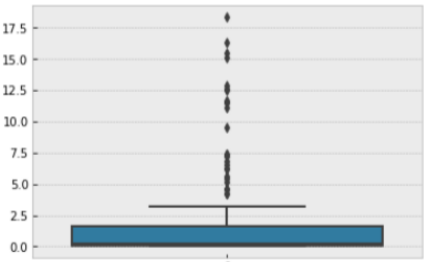
L'objectif général de la comparaison des distributions est de voir si nos datasets sont comparables. Par exemple, sur les graphiques ci-dessus, on voit que les valeurs du Ground_Truth sont (dans les 2 datasets train et test) proche de 0. On a vu que les moyennes et les écarts-types étaient assez proches, donc on peut dire que les datasets sont assez similaires.

La différence réside dans les valeurs extrêmes : celles du trainset atteignent presque 40 alors que dans le testset, on dépasse à peine 20 (il y a très peu de jours dans l'année où il pleut à torrent).

Il peut être intéressant d'analyser ces divers propos en utilisant une boîte à moustaches :



Box plot Ground_Truth Trainset

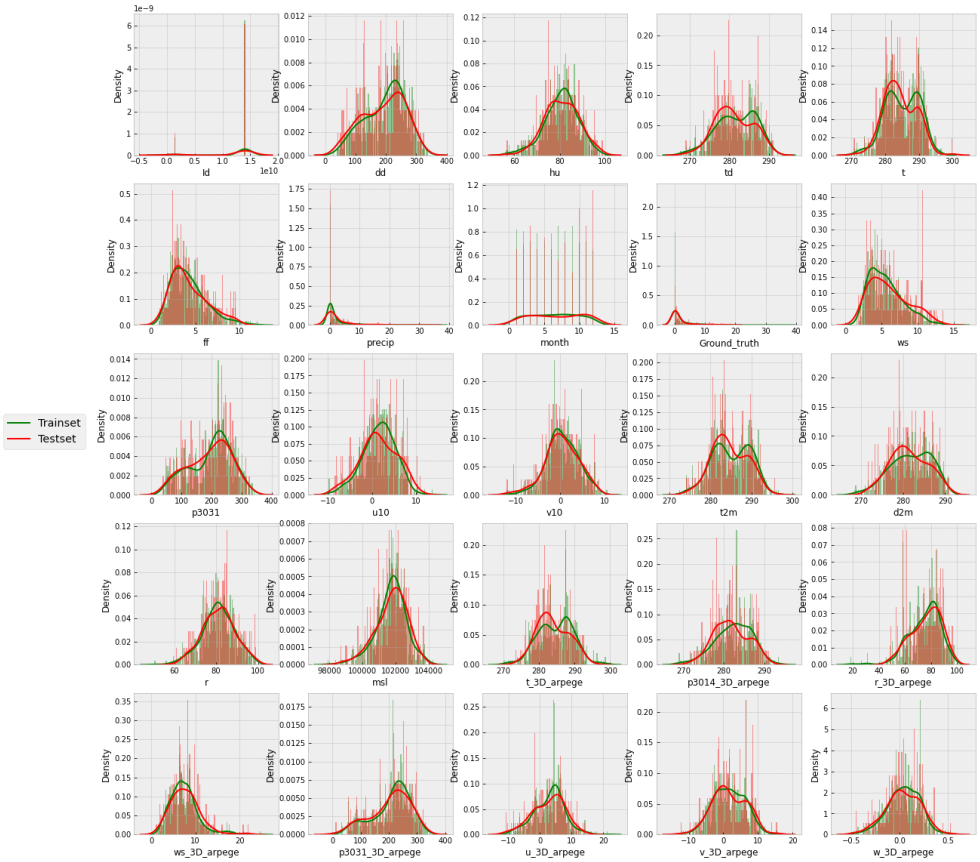


Box plot Ground_Truth Testset

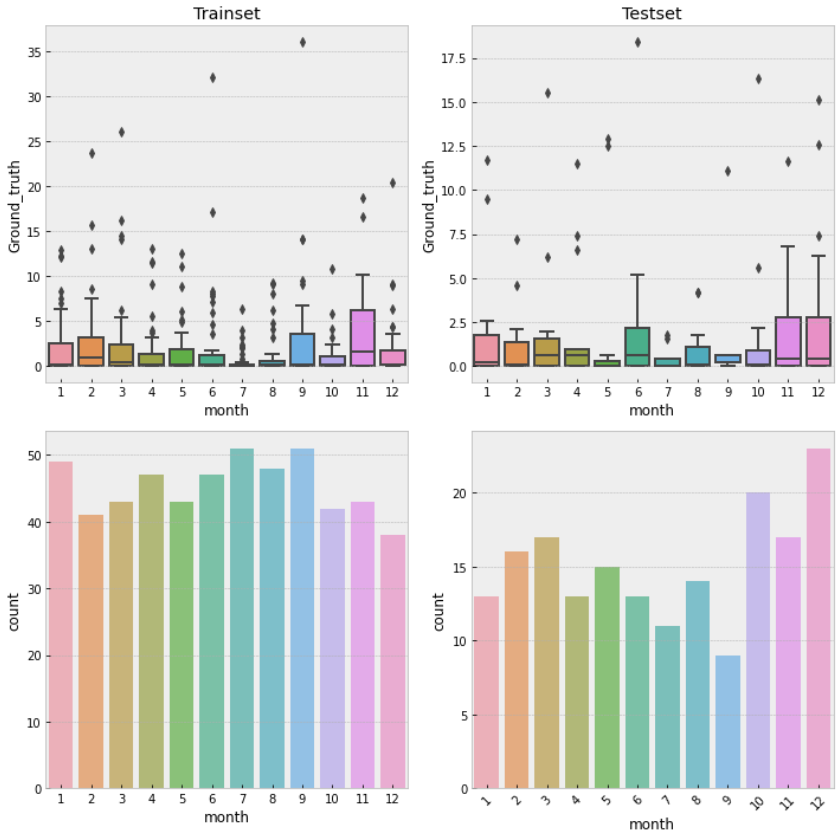
Sur le dataset d'apprentissage, on un maximum du Ground_truth égal à 36.1 alors que dans le dataset de test, il vaut 18.4. De plus, dans le dataset d'apprentissage, on peut voir qu'au moins 75% des valeurs sont inférieures à 2.20 alors que dans le dataset de test, le 3ème quartile vaut 1.6. Par conséquent, on comprend que le dataset d'entraînement contient des valeurs plus élevées concernant le Ground_truth.

Comparaison des distributions des variables

Sans trop nous éterniser sur ce graph, on voit que les distributions des variables entre le trainset et le testset sont très similaires. Pour conclure, nous pouvons dire que ces deux jeux de données sont fiables pour l'entrainement de modèles de ML ou de DL.



Équilibrage du jeu de données



Les données étant des séries temporelles quotidiennes, qui plus est pour des prédictions météorologiques, il est intéressant de se demander si les données sont équitablement réparties dans le temps entre le trainset et le testset.

En l'occurrence, les relevés du trainset sont assez équilibrés dans le temps. Cependant on voit sur le testset qu'entre certains mois la différence va du simple au double (par exemple entre Septembre et Décembre)

Analyse des corrélations avec la target

Afin de comprendre quelles variables impactent le Ground_truth et ainsi comprendre quelles variables choisir pour les prédictions, il est nécessaire d'effectuer une analyse des corrélations entre les features et notre target :

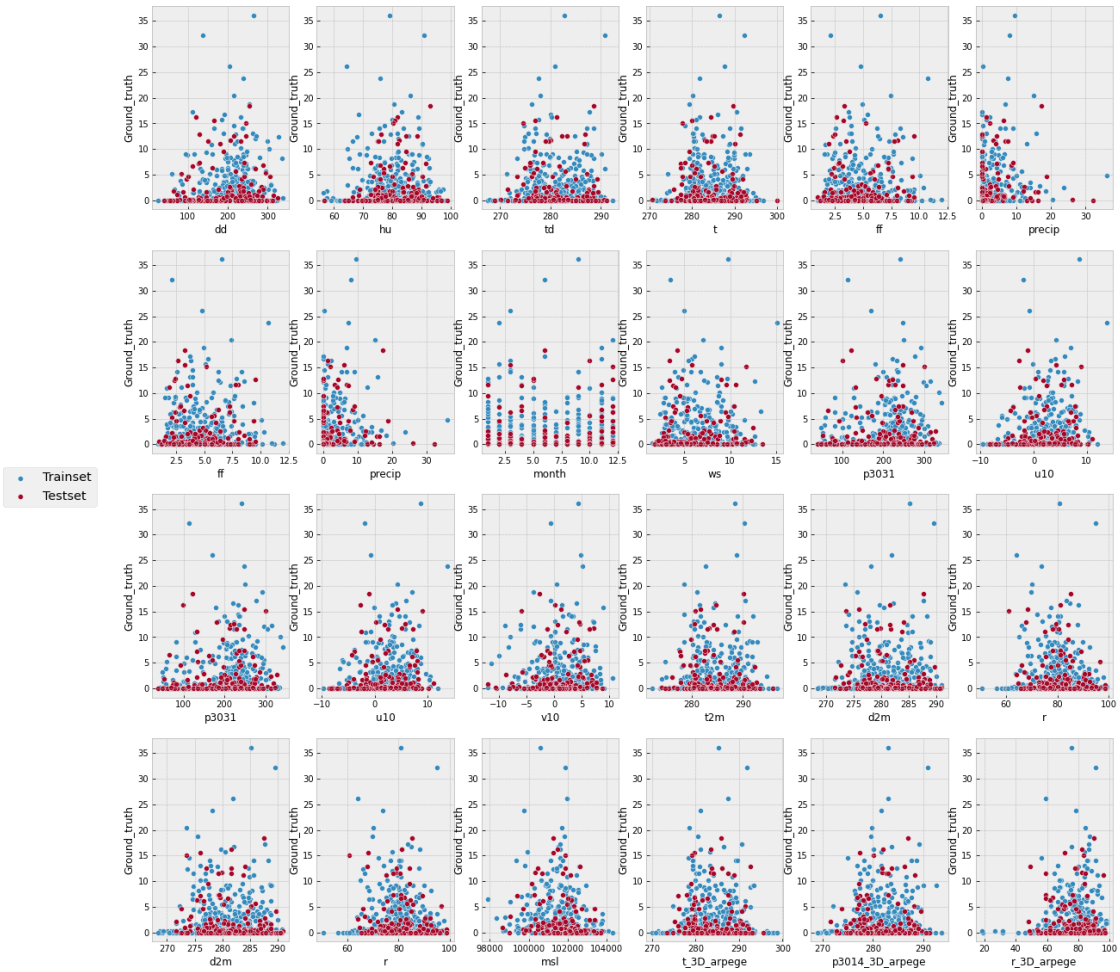
```
There are 7 strongly correlated values with Ground_truth in trainset:
precip          0.253079
ws              0.200266
u10             0.192514
ws_3D_arpege    0.179662
ff              0.177599
u_3D_arpege     0.171108
msl            -0.218911
Name: Ground_truth, dtype: float64
--------
There are 2 strongly correlated values with Ground_truth in testset:
precip    0.196791
msl      -0.198063
Name: Ground_truth, dtype: float64
```

Nous pouvons penser que cela peut suffire pour choisir nos variables mais en réalité ce n'est pas le cas puisque l'analyse des corrélations est très sensible aux outliers (valeurs extrêmes).
Par conséquent nous pouvons procéder selon la stratégie suivante pour éliminer les features qui n'ont pas de corrélation avec le Ground_truth sans leurs outliers :

1) Afficher les features qui disposent d'outliers
2) Parmi elles, supprimer les outliers et retenir celles qui ont encore une corrélation avec le Ground_truth

D'autre part, la corrélation entre les variables à elle seule n'explique pas les relations entre les données. Ainsi, les représenter sur un graph pourrait nous apporter de nouvelles informations. L'idée est de vérifier si les variables fortement corrélées entre elles sont linéairement liée à Ground_truth

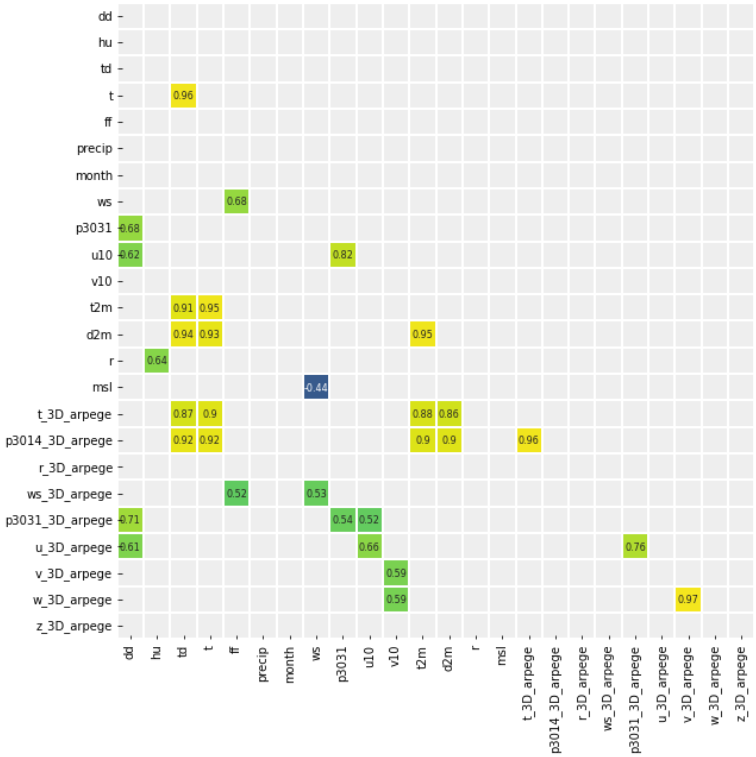
Par exemple, des relations telles que curvilinéaire ne peuvent pas être devinées au premier regard sur la valeur de la corrélation. Prenons à présent les variables les plus corrélées de notre table et observons le graphe.



Malgré le fait que certaines variables semblent légèrement corrélées avec Ground_truth, il reste impossible à l'œil d'identifier une quelconque relation entre une/plusieurs variables et la target. Même la variable « precip » qui est la plus corrélée avec la target (0.25 tout de même) ne montre pas de signe flagrant de relation linéaire

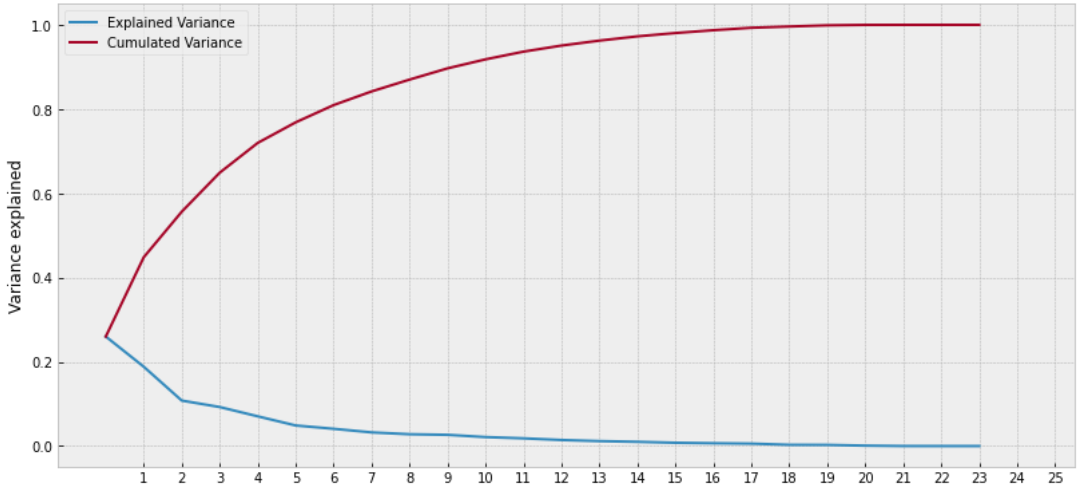
Analyse des corrélations entre les variables

On peut ensuite identifier des corrélations inter-variables, notamment sur les relevés de température (t, td, t2m, d2m, t_3D_arpege, p3014_3D_arpege)



Analyse en Composantes Principales

La dernière étape de notre analyse des données a été d'estimer à quel point on pouvait réduire la dimension des variables de notre jeu de données. En effet nous ne possédons que très peu d'observations et il y a beaucoup de paramètres. Nous aurions donc aimé savoir quelles variables jouent un rôle important dans le jeu de données.



On voit que sur les 24 composantes de la PCA, les ~13 premières portent près de 95% de la variance du jeu de données.

Feature Importance

En choisissant un estimateur pour la régression (au hasard, le SVR), on peut également l'entrainer à choisir les variables « importantes ». Cette technique consiste à calculer un « score » représentant l'importance de la variable, sachant qu'un score élevé signifie que la variable a un fort impact sur les performances de l'estimateur.
En utilisant le SVR come estimateur pour la régression, nous retenons 7 features :

	precip	u10	v10	msl	r_3D_arpege	ws_3D_arpege	z_3D_arpege
0	-0.431170	0.266502	-0.644580	0.650313	0.509997	-0.952609	0.421480
1	-0.431170	-0.206230	-0.055708	1.522480	-1.387642	-0.032966	0.638967
2	-0.431170	0.135900	-0.353027	0.585089	0.804588	-0.806922	0.438557
3	2.598228	0.073721	-0.547549	-0.063643	0.339893	-0.345080	-0.541067
4	-0.486755	-0.009063	-0.217898	0.270860	0.746512	-0.963899	0.151625

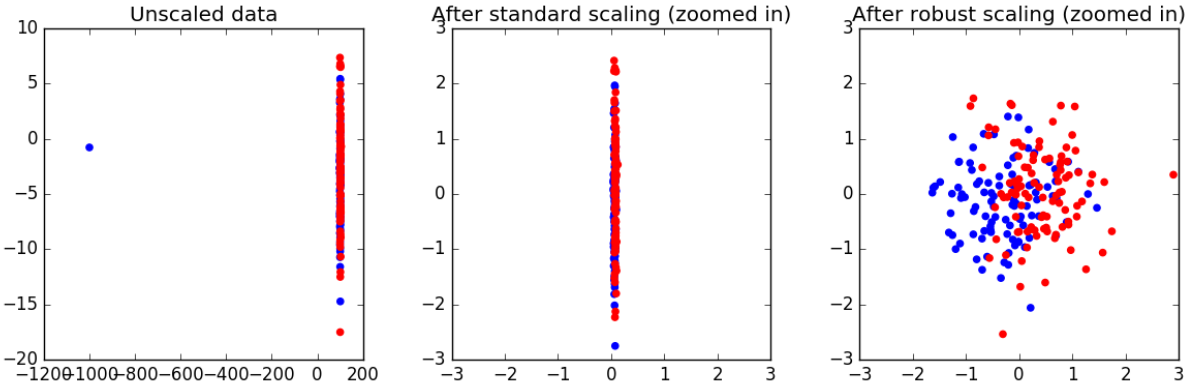
On remarquera que ces 7 features ne sont pas toutes identiques (seulement 3) aux 7 features fortement corrélées avec le Ground_truth que nous avions trouvées 2 pages plus haut.

V. Modélisation

Nous partons donc avec un jeu de 543 lignes et 28 colonnes. Toutes les colonnes sont de type « float64 » sauf la colonne « month » qui est de type « int64 » car variable catégorielle. Nous pouvons dans un premier temps supprimer les colonnes qui n'ont aucune variance, à savoir les attributs de la station (« number_sta », « lat », « lon ») car dans ce projet nous n'utiliserons les données que d'une seule station.

Preprocessing

Dans un premier temps, nous avons normalisé les variables. De ce fait, lorsque nous utiliserons des algorithmes de régression régularisée, le coefficient de pénalité α agira de façon homogène sur l'ensemble des coefficients de la régression. Nous avons choisi pour commencer un "RobustScaler" car nos données sont très sensibles aux outliers. De ce fait la normalisation vis à vis de la médiane est plus robuste que celle vis-à-vis de la moyenne :



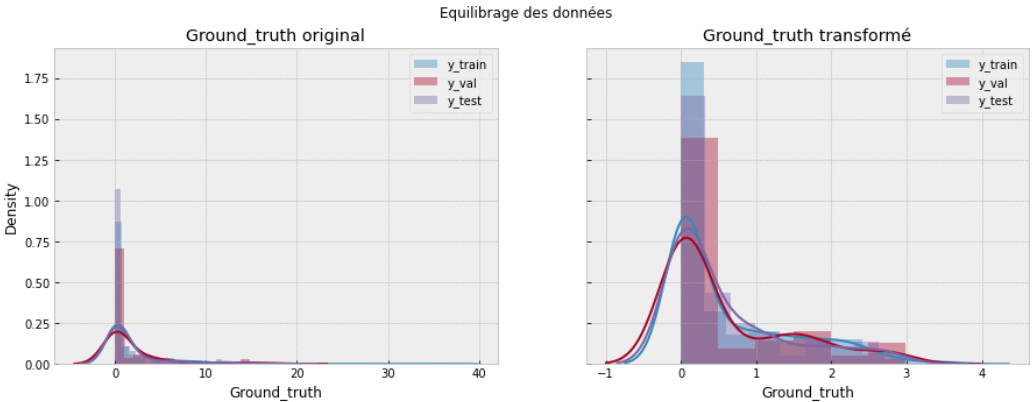
La formule de transformation s'écrit (individu n°i, variable Xj) :

$$\text{RobustScaler: } z_{ij}^{train} = \frac{x_{ij}^{train} - \left(x_{med}^{train}\right)}{p75^{train} - p25^{train}}$$

Nous avons également ajouté une colonne "intercept" remplie de 1 afin de modéliser la constante dans les calculs des modèles linéaires

Une autre transformation que nous avons effectuée consiste à rééquilibrer les distributions entre le trainset et le testset. A regarder de plus près la distribution de la target, on s'aperçoit qu'il y a beaucoup d'outliers. Un des moyens de "rééquilibrer" cette distribution est d'utiliser cette transformation :

$$y_{val} = log(y_{val} + 1) + min(y_{val})$$



Instauration d'une métrique et comparaison de modèles

La métrique que nous utiliserons pour évaluer nos modèles est la même que pour le Défi-IA : la MAPE. C'est une métrique très souvent utilisée chez METEO France pour les prédictions de précipitations. Elle est définie par :

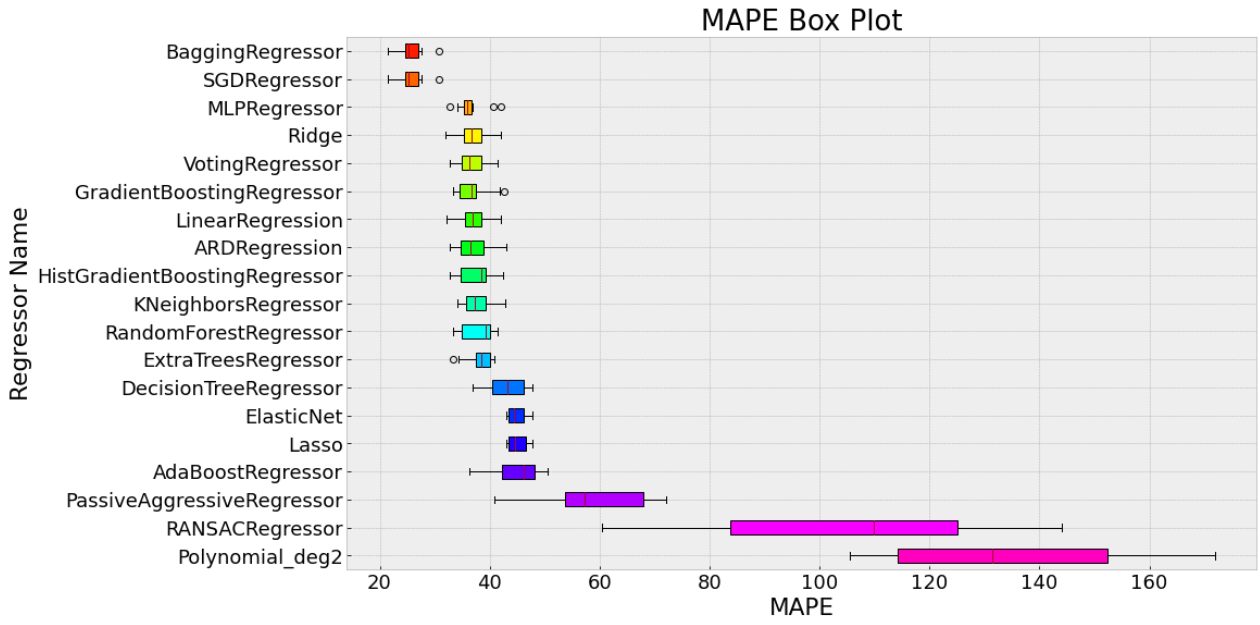
$$MAPE = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t}\right|$$

Avec chaque prédiction Ft comparée à la vraie valeur At pour l'observation t parmi n.

Important : Il est très fréquent de ne pas avoir de précipitation lors d'une journée. Afin d'éviter les divisions par 0 lorsqu'At=0, on va plutôt chercher à prédire la **vraie valeur +1** :

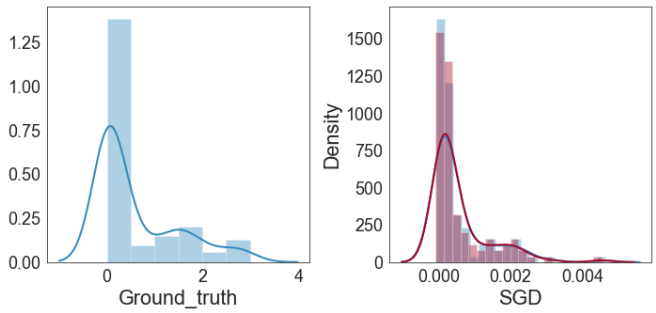
$$MAPE = \frac{100}{n}\sum_{t=1}^{n}\left|\frac{A_t - F_t}{A_t + 1}\right|$$

Afin d'avoir une idée de l'ordre de grandeur de notre métrique, nous avons pris une dizaine de modèles de régression dont nous avons laissé tous les paramètres par défaut. Nous avons effectué une Cross-Validation avec 10 Folds, entrainé les modèles sur les données d'entrainement, puis évalués sur les données de validation. Voici les résultats :



On peut tirer plusieurs informations de ce graphe :

- Le BaggingRegressor et le SGDRegressor ont exactement le même profil car l'estimateur de base du BaggingRegressor est en réalité un SGDRegressor. Il va falloir creuser, notamment en observant la distribution de leur prédiction pour avoir une idée de ce qui les rend si performants (entre 23 et 27 de MAPE).



On peut voir que la prédiction des modèles SGD s'apparente à une ligne de 0. En effet il y a une majorité de jours sans pluie dans l'année. On a donc un score plus qu'acceptable en prédisant 0mm de pluie par jour. Pour la suite, il faudra « inciter » les modèles à prendre des risques pour prédire les pics de pluie (cf. pénalisations)

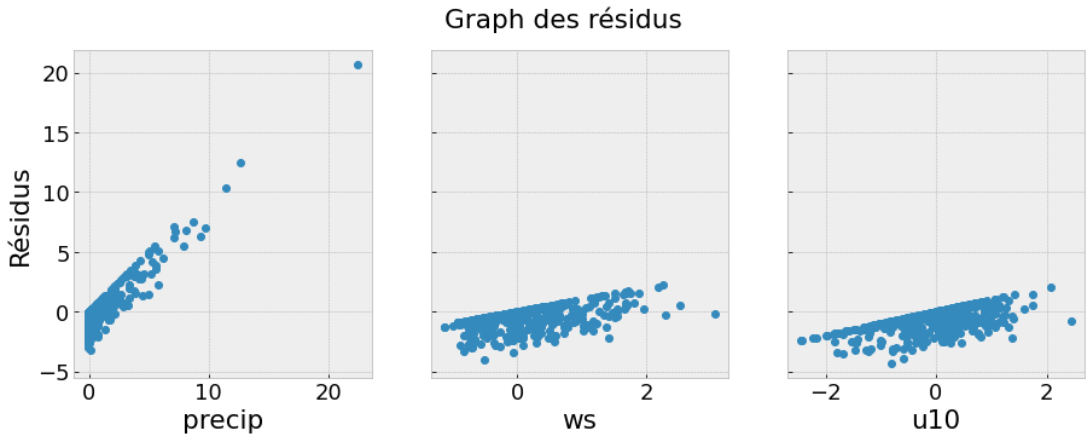
- Une majorité des modèles ont des performances situées entre 30 et 50 de MAPE.

- Les modèles PassiveAggressive, RANSAC et Polynomial sont assez fluctuants vis-à-vis de la MAPE. Cette dernière demeure également élevée comparée aux autres modèles.

On peut également essayer de calculer la valeur de la MAPE entre le Ground_truth et certaines variables. Prenons par exemple les 3 variables les plus corrélées avec la target :

MAPE (precip / Ground_truth) = 60.48
MAPE (ws / Ground_truth) = 49.67
MAPE (u10 / Ground_truth) = 57.73

On peut aussi calculer les résidus entre ces features et la target :



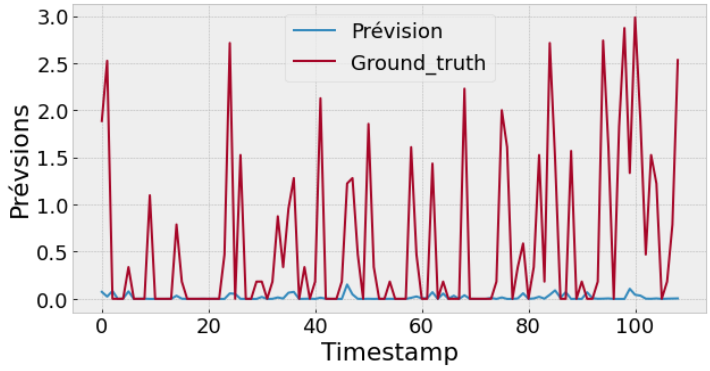
On ne s'attend donc pas à trouver une MAPE supérieure à 60, sans quoi le modèle n'aurait pas de bonnes performances. De plus, il faut se méfier des modèles avec une MAPE autour de 25 car on a vu que la prédiction facile était une ligne de 0. Il faudra donc à chaque fois analyser la distribution ou l'allure des prédictions des modèles.

Modèle LASSO

Nous avons commencé par prendre le modèle linéaire Lasso, avec ses paramètres par défaut :

$$regLasso = sklearn.linear_{model}.Lasso(fit_{intercept} = False, normalize = False)$$

$$Optimization_{Objective} = \frac{1}{(2 * n_{samples})} * ||y - Xw||_2^2 + \alpha * ||w||_1$$

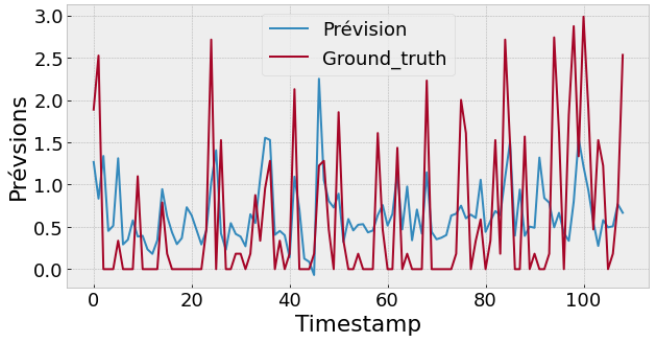


On peut voir que le modèle Lasso avec les paramètres par défaut (notamment $\boldsymbol{\alpha = 1}$) n'a pas de très bonne performances (MAPE = 56.92), un peu comme le SGDRegressor sa prédiction est très proche d'une ligne de 0.

Afin de corriger ce choix facile du modèle, nous avons effectué un GridSearch avec une Cross-Validation sur le paramètre **α.**

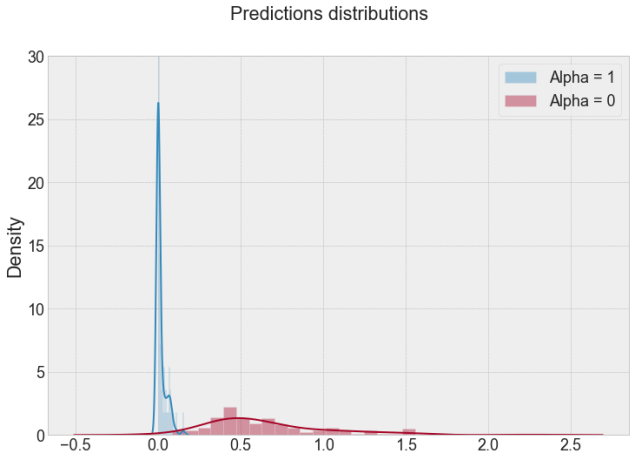
$$param = [\{"alpha": [0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 1]\}]$$
$$regLasso\_grid = GridSearchCV(linear\_model.Lasso(fit\_intercept = False), param, cv = 5, n\_jobs = -1)$$

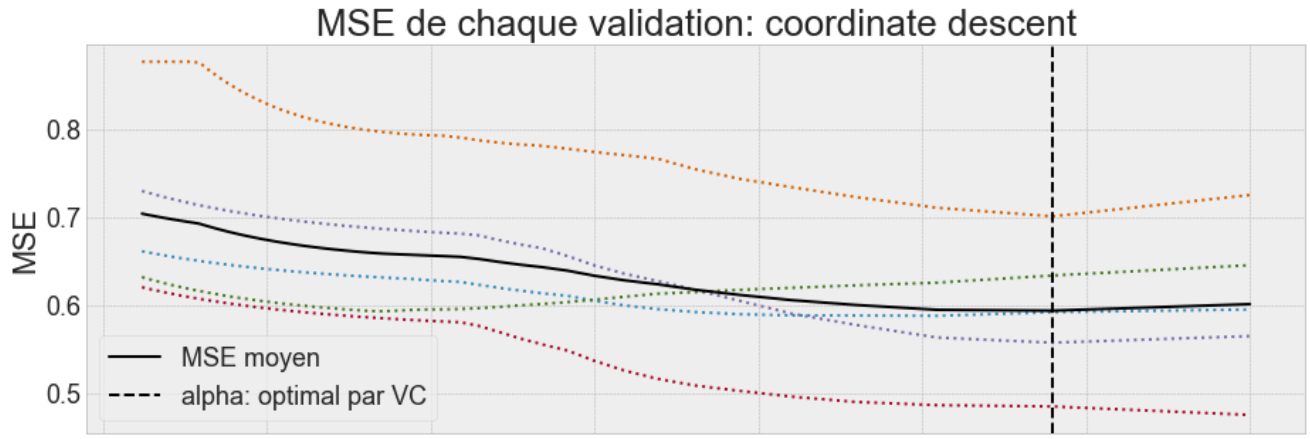
Voici les résultats :

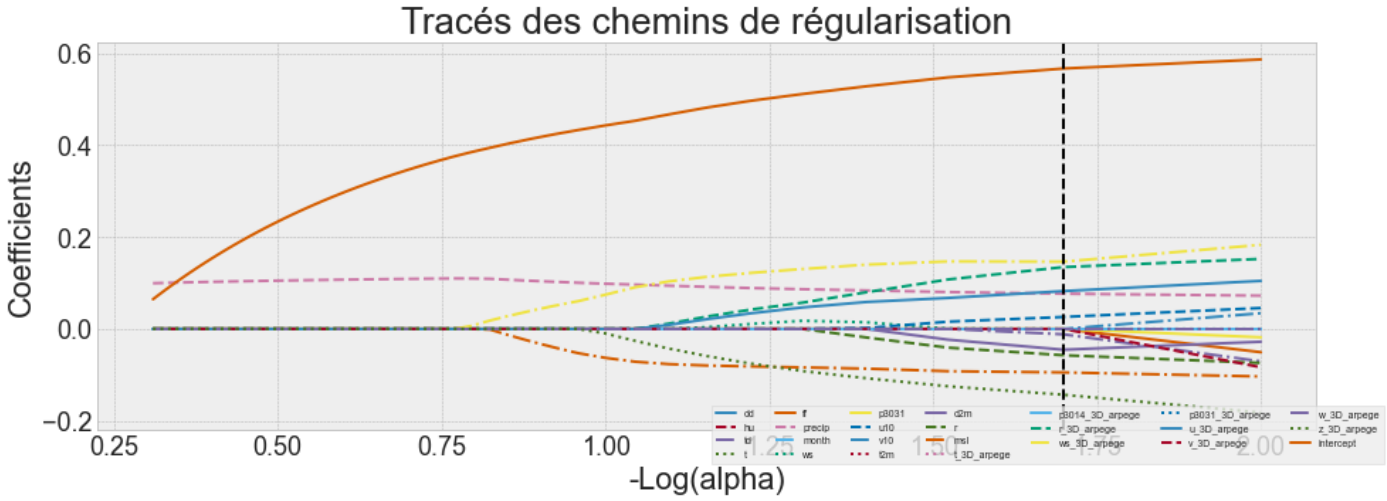


Le GridSearchCV nous retourne « Meilleur paramètre = {'alpha': 0.05} », ce qui nous ramène presque au final pour la fonction d'optimisation à un problème classique des moindre carrés. La MAPE descend à 36.66 et la prédiction a une allure déjà plus correcte.

On peut effectivement voir que ce simple changement de paramètre a drastiquement changé l'allure de la distribution de la prédiction. En fonction du paramètre **α**, le modèle Lasso conserve plus ou moins de variables du jeu de données. On peut le voir dans le tracé des chemins de régularisation. Nous avons également affiché la valeur des coefficients du modèle pour la valeur 0.05 du paramètre **α**.





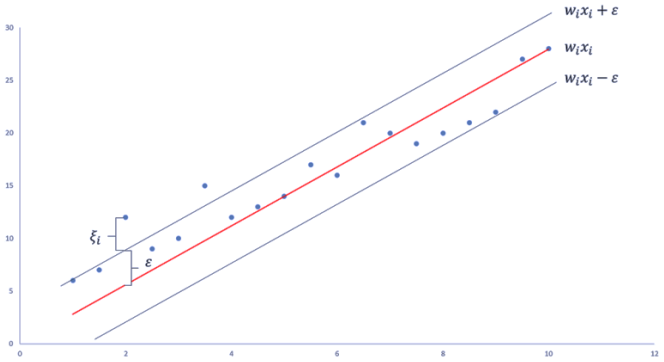


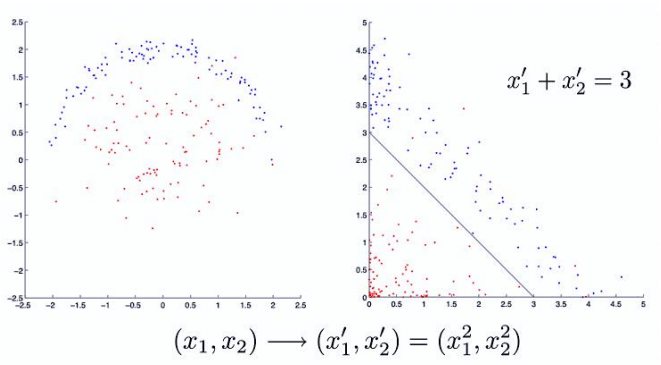
Lorsque α est trop élevé, tous les coefficients de la régression sont nuls, nous en avons une illustration ici ; lorsque α est trop faible, proche de 0, nous obtenons les coefficients de la régression linéaire multiple usuelle. Il faut trouver le juste milieu et c'est toute la difficulté de la régression Lasso. L'outil « Lasso path » peut nous y aider. Il produit un graphique qui met en relation les différentes versions de α avec les coefficients estimés. La régression Lasso, contrairement à Ridge, permet de réaliser une sélection de variables (ici 8) en mettant à zéro sélectivement les coefficients. Nous voyons ainsi se dessiner des scénarios de solutions (au sens « différents ensembles de variables sélectionnées ») tout au long du « Lasso path ».

Modèle SVR

L'intérêt de ce modèle est de pouvoir séparer linéairement les données. Or nous avons vu qu'il était très compliqué de trouver un hyperplan optimal pour les données que nous avons :

$$Min\frac{1}{2}|w^2| + C\sum_{i=1}^{n}|\xi_i|$$
$$|y_i - w_i x_i| \leq \epsilon + |\xi_i|$$



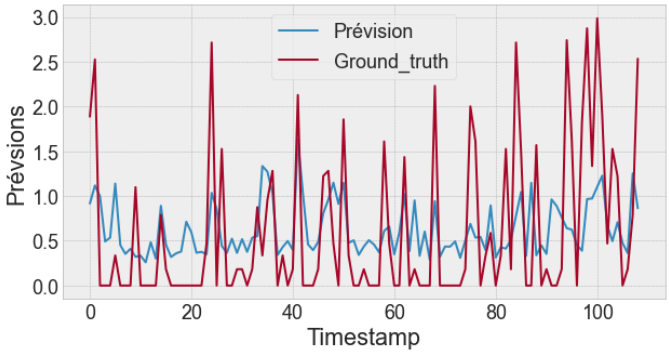


Il faudrait donc utiliser un noyau pour transformer nos données pour passer d'un cas non-linéaire à un cas linéaire

La bibliothèque sklearn nous en propose 4 :

- Linear kernel : $k(x, x') = < x, x' >$
- p degree polynomial kernel : $k(x, x') = (1 + < x, x' >)^p$
- RBD (Gaussian kernel) : $k(x, x') = e^{-\frac{|x-x'|^2}{2\sigma^2}}$
- Sigmoid kernel : $k(x, x') = tanh(\kappa < x, x' > + \theta)$

On peut d'ores et déjà éliminer le noyau sigmoid qui n'est pas défini positif, car le résultat produit n'aide pas à la régression. Nous pouvons également éliminer le noyau linéaire car on a vu qu'il était très dur de déduire un comportement linéaire des données. Il reste donc deux noyaux qui semblent prometteurs : le noyau Gaussien et le noyau Polynomial.

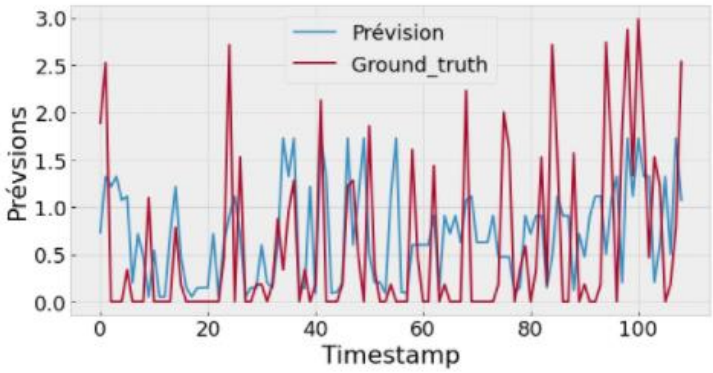


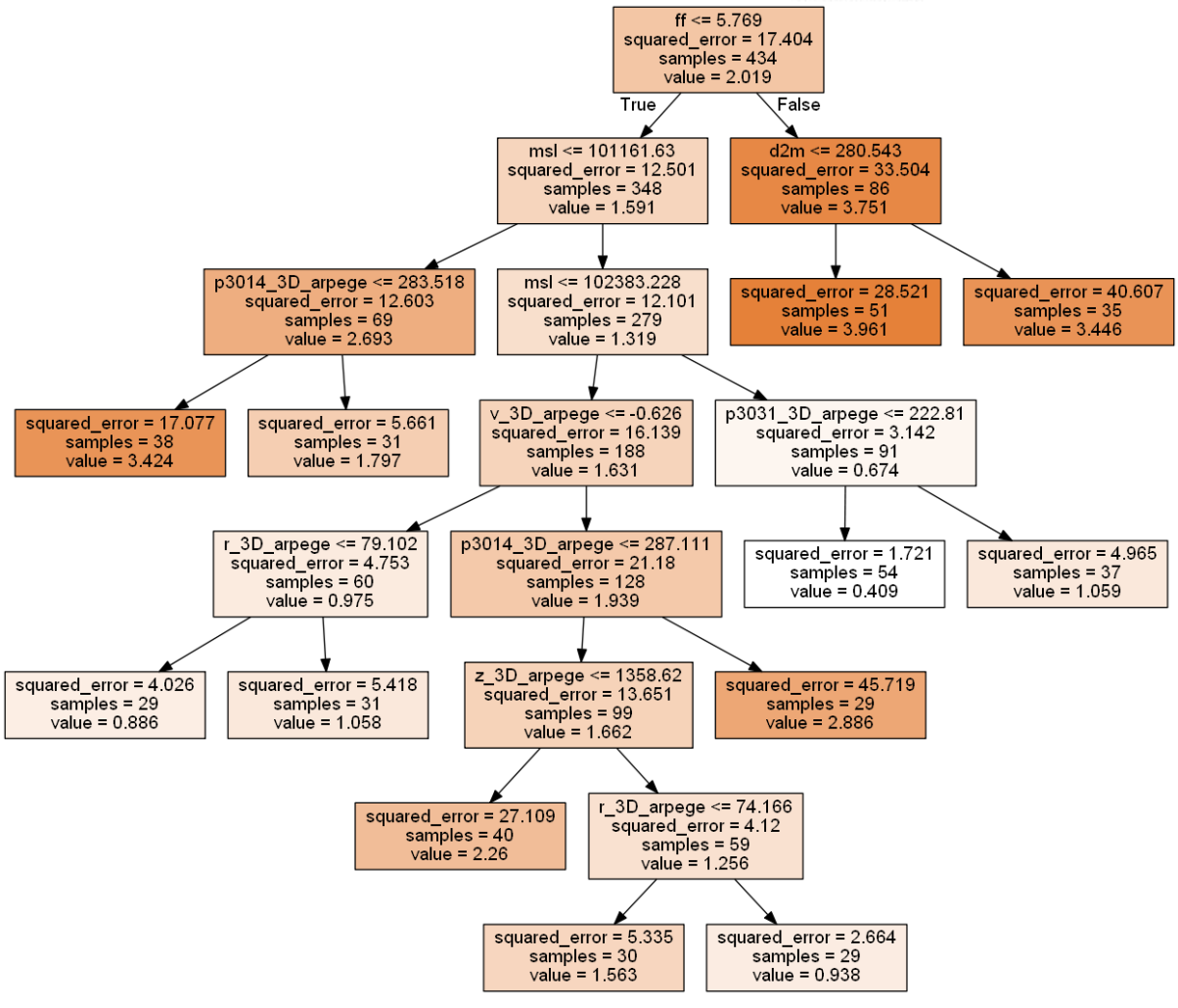
Nous avons ensuite optimisé les hyper-paramètres de notre modèle avec un GridSearchCV et voici le résultat : Meilleurs paramètres = {'C': 0.4, 'epsilon': 0.4, 'kernel': 'rbf', 'degree' :2}.
Le paramètre « degree » n'est pas pris en compte dans le noyau Gaussien.
Nous avons donc $\boldsymbol{C = 0.4}$ et $\boldsymbol{\epsilon = 0.4}$

Modèle DecistionTreeRegressor

L'intérêt du DecisionTree est de pouvoir récupérer à la fin un arbre de décision qui permet de mieux comprendre pourquoi telle entrée est associé à telle sortie. Cela évite l'effet « boite noire » présent dans beaucoup d'autres modèles. Il est possible de jouer sur plusieurs paramètres tel que le nombre d'échantillons par branche et par feuille, la façon de séparer les branches ou encore le critère utilisé pour tester le modèle.

Avec un GridSearchCV nous avons pu optimiser les variables de notre modèle. Meilleurs paramètres = {'criterion': 'squared_error', 'min_samples_leaf': 14, 'min_samples_split': 2, 'splitter': 'random'}. Finalement notre modèle obtient une MAPE de 37,8 ce qui est un bon score en comparaison avec tous les autres modèles

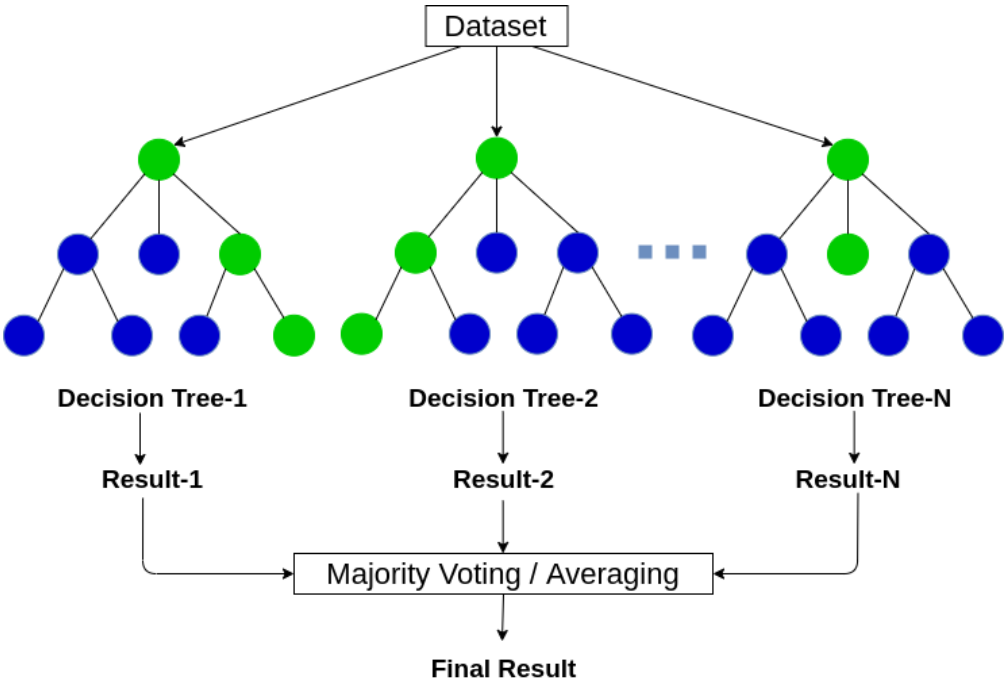




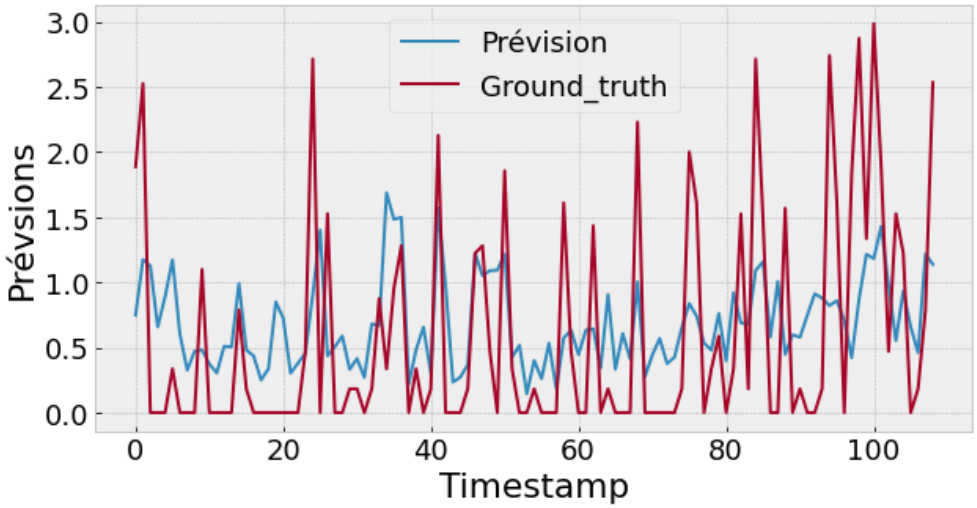
Comme expliqué auparavant, ce modèle nous permet d'avoir en sortie un arbre de décision pour mieux interpréter nos résultats. Grâce à ça nous pouvons facilement expliquer les décisions prises et voir l'importance de chaque variable. Il existe un point négatif sur ce graphique, il est nécessaire d'effectuer les transformations inverse sur X et Y afin d'interpréter correctement ce graphique.

Modèle RandomForest

L'algorithme de RandomForest est une agrégation de petits arbres de décisions (d'où son nom).

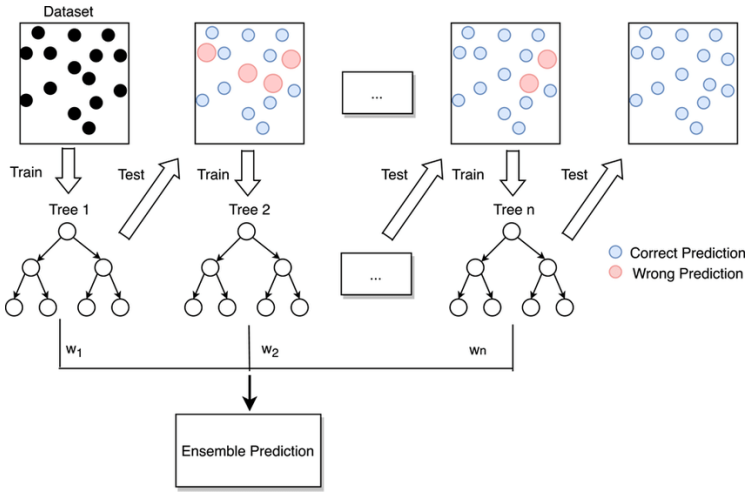


Il y a encore une fois dans ce modèle plusieurs paramètres qu'il est possible de modifier et d'adapter à notre travail. On peut utiliser le bootstrap afin d'entrainer notre forêt. Comme pour le DecisionTree il est possible de choisir le nombre d'échantillons par branche et par feuille de chaque arbre. En plus de cela nous pouvons modifier le nombre d'arbres ou encore la profondeur de chacun de ces arbres. Finalement, les meilleurs paramètres sont : {'bootstrap': True, 'max_depth': 110, 'max_features': 2, 'min_samples_leaf': 3, 'min_samples_split': 10, 'n_estimators': 100}. Nous avons pu obtenir une MAPE de 35.06 ce qui est un bon score.

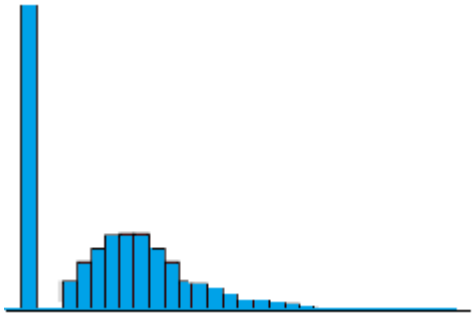


Modèle XGBoost

Le modèle XGBoost est une implémentation d'un ensemble d'algorithmes pour la classification et la régression. Si le proverbe « l'union fait la force » devait avoir une représentation mathématique, elle serait le Gradient Boosting. En effet, cette méthode regroupe plusieurs estimateurs faibles pour combiner tous leurs résultats en une prédiction ensembliste.



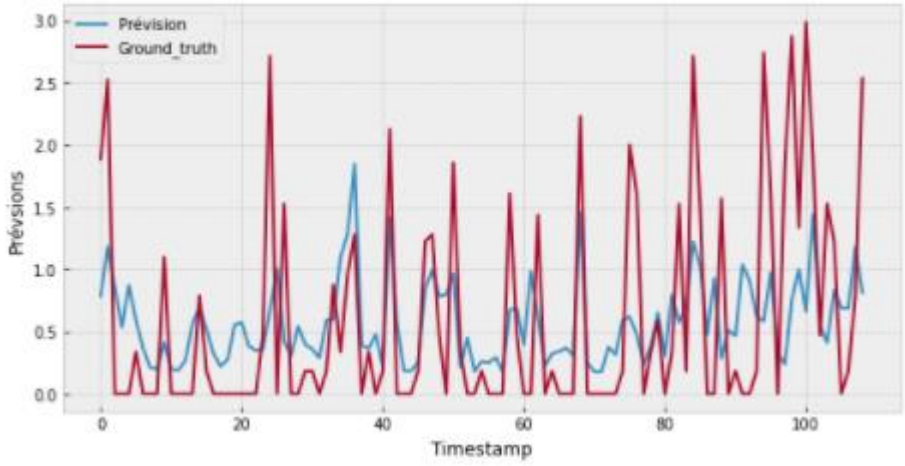
La librairie python xgboost comporte un modèle XGBRegressor qui possède plusieurs paramètres dont le plus important : le nombre d'estimateurs **n_estimators**. Plusieurs paramètres peuvent également être influents comme la taille maximale des arbres de régression **max_depth**, le learning_rate ou encore les paramètres **reg_alpha** et **reg_lambda** qui correspondent aux paramètres de régularisation L1 et L2 sur les poids.



Pour notre problème nous avons une distribution assez particulière du nom de 'Tweedie Distribution' et qui est en fait un cas particulier d'une distribution exponentielle ayant une grande densité sur une certaine valeur (ici 0)

Grâce à un GridSearch CV nous avons pu optimiser différents paramètres comme cité auparavant. Finalement, les paramètres optimales sont : {'eval_metric': 'tweedie-nloglik@1.1', 'learning_rate': 0.05, 'n_estimators': 30, 'objective': 'reg:tweedie', 'tree_method': 'approx', 'tweedie_variance_power': 1.1}.

Avec ces paramètres nous obtenons une MAPE de 34.2 ce qui est un score relativement bas (un des meilleurs scores si nous ne prenons pas en compte les modèles linéaires prédisant constamment 0). Le problème de ce modèle est le manque d'interprétaiblité des résultats. En effet comme d'autres modèles il est difficile d'établir un lien direct entre l'entrée et la sortie.



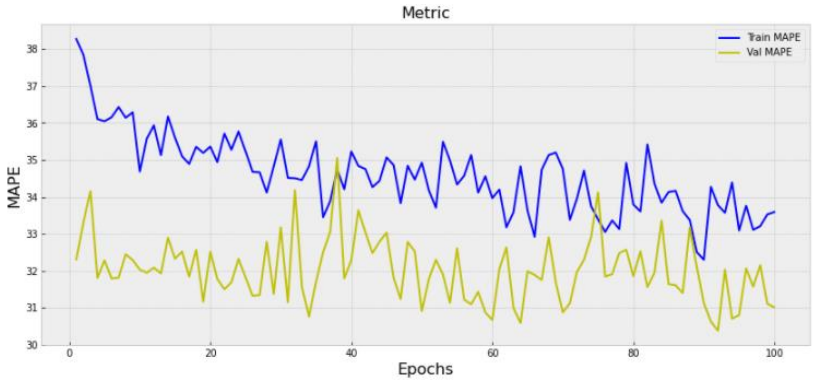
Modèle Réseau de Neurones

Le dernier modèle que nous avons eu l'occasion de tester est un modèle de Réseau de neurones. Le premier choix à faire est le type de modèle utilisé ainsi que le nombre de couches de ce dernier. Ici nous utilisons un perceptron multi couches.

```
ann.add(BatchNormalization())

ann.add(Dense(24 ,activation="relu", kernel_initializer='normal', bias_initializer='zeros',
              kernel_regularizer=regularizers.l1(1e-1)))
ann.add(Dropout(0.2))

ann.add(Dense(256, activation="relu", kernel_initializer='normal', bias_initializer='zeros',
              kernel_regularizer=regularizers.l1(1e-1)))
ann.add(Dropout(0.2))

ann.add(Dense(124, activation="tanh", kernel_initializer='normal', bias_initializer='zeros',
              kernel_regularizer=regularizers.l1(1e-1)))
ann.add(Dense(1))
```

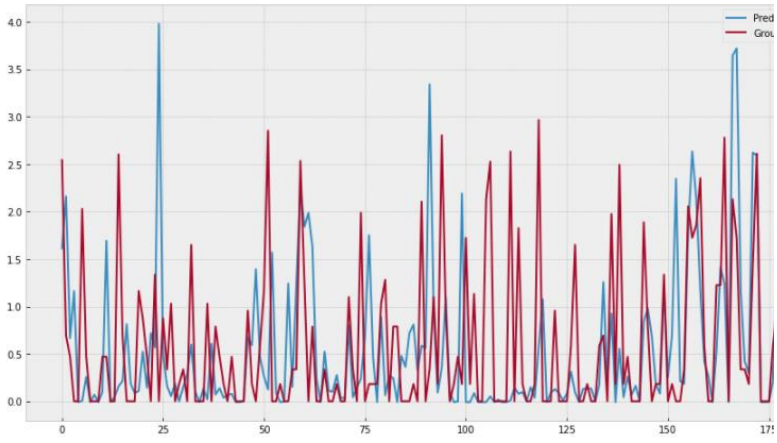
Lors de l'initialisation du réseau nous pouvons choisir différentes régularisations du noyau. Il est nécessaire de faire un choix entre une régularisation L1 et L2. Nous avons fait le choix d'utiliser une régularisation L1 qui permet d'avoir une feature selection plus précise ainsi qu'une certaine robustesse face aux outliers.

Une fois le réseau initialisé nous l'avons entrainé en utilisant la MAPE comme metrics. On peut voir sur le graphique ci-dessous que la training et validation MAPE ont une allure légèrement décroissante malgré leurs fortes fluctuations. Ces fluctuations peuvent être dues par exemple au modèle qui parfois prédit très largement à côté du Ground Truth (par exemple une grande précipitation alors qu'il n'a pas plu).




```
Ann train MAPE : 29.5
Ann val MAPE : 31.48
MAPE on Testset (data never seen by the
NN): 36.47
```

Nous avons obtenu une MAPE de validation de 34.6 ce qui est un bon score. On peut voir dans les prédictions ci-dessous que le réseau 'ose' des valeurs assez haute quitte à faire monter la MAPE de quelques points. Encore une fois malgré le bon score, un réseau de neurones reste un modèle ininterprétable.



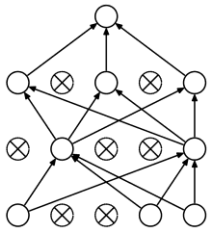
VI. Résumé des performances des différents modèles testés

Modèle	*MAPE*	*Remarques*
Linéaire (meilleur modèle)	23.4	Prédit une ligne de 0
Lasso	36.7	Allure encourageante mais manque de précision
SVR	36	
DecisionTree	37.8	
RandomForest	35.06	Allure de la prédiction très intéressant
Xgboost	34.3	
Réseau Neurones	34.6	

VII. Axes d'amélioration

Éviter le sur-entrainement

Afin d'éviter l'overfitting notamment lors de l'entrainement du réseau de neurones, nous avons utilisé plusieurs outils mis à notre disposition à travers la librairie Keras :



1. **Le dropout** : Il permet de réduire l'overfitting lors de l'entrainement d'un modèle en « oubliant » des neurones dans certaines couches de Deep Learning. On désactive en réalité un pourcentage de neurones du réseau (à chaque fois des neurones différents) ainsi que toutes ses combinaisons entrantes et sortantes.
2. **La régularisation par pénalisation** : La régularisation permet également d'appliquer une pénalité aux paramètres des couches durant l'optimisation. Ces pénalités sont ajoutées à la fonction coût que le réseau optimise.
    a. Pour la **L1-Regularization** la somme des poids en valeur absolue multiplié par une constante α
    b. Pour la **L2-Regularization** la somme des poids au carré multiplié par une constante

$$L1\ Regularization\ = (loss\ function) + \alpha \sum_{j=1}^{p} |b_j|$$

$$L2\ Regularization\ = (loss\ function) + \alpha \sum_{j=1}^{p} b_j^2$$

3. **EarlyStopping**

Sans l'utilisation de ces outils de régularisation, il nous arrivait d'obtenir des MAPE descendant parfois jusque 20 sur notre jeu d'entrainement et de validation. En revanche une fois que l'on soumettait une prédiction sur Kaggle, le résultat ne descendait jamais en dessous de 40 à cause du surentrainement.

Le choix de la pénalisation L1 vient du fait qu'elle est plus robuste aux outliers (très présents dans notre jeu de données) et donc nous permet d'améliorer la qualité de détection des jours atypiques à forte pluie. La pénalisation L2 aurait apporté plus de stabilité.

NaNs :

Le jeu de données comportait beaucoup de NaNs (jusqu'à 50% pour certaines features), et notamment le jeu de données CSV présent sur Kaggle (colonnes « dd », « ff », « hu », « t » en particulier). Nous avons également retrouvé des NaNs lors de nos soumissions par rapport aux Id demandés à la Baseline (pour être plus clair, on nous demandait d'effectuer des prédictions sur des lignes totalement vide : ~4.7%). Pour pallier ce problème, nous remplacions simplement les valeurs manquantes par 0 (pas de pluie le jour j sur la station S). Il y a notamment un gros manque de données (erronées ou simplement non acquises) sur les jeux ce données AROME et ARPEGE (jusqu'à 27% suivant les features). L'imputation n'a pas été évidente.

Une des pistes d'amélioration aurait été de trouver une imputation meilleure, par exemple de remplacer les NaNs par la valeur de la station la plus proche à la même heure (imputation qui est faisable sur le trainset et sur le testset).

Passer en dessous de la valeur de MAPE calculée pour une prédiction constante à 0

Il s'avère que si on s'amuse à soumettre une ligne de 0 (en l'occurrence de 1 car il faut ajouter +1 à notre prédiction pour le calcul de la MAPE), et bien on obtient un honnête **score de 23.4**. En effet la distribution étant centrée très proche de 0 et les valeurs au-dessus de 2 étant peu nombreuses, on ne se trompe pas beaucoup en prédisant qu'il ne pleut jamais… En revanche, le vrai défi réside en notre capacité à réussir à identifier les pics de pluie à travers la majorité de jours sans pluie.

Feature Engineering

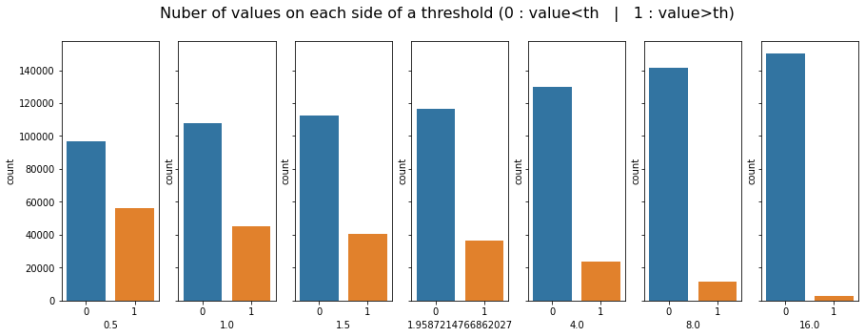
On a pu voir avec tous les modèles utilisés que nous n'arrivions pas à dépasser une certaine barrière pour la valeur de la MAPE, et ce quel que soit le modèle. Ce dont nous avons cruellement manqué est la connaissance des phénomènes météorologiques ainsi que l'influence des différents paramètres les uns sur les autres. Nous aurions alors pu introduire de nouvelles variables, combinaisons non linéaires de celles déjà présentes et représentatives de phénomène météorologiques afin d'aider les algorithmes dans leurs prédictions. Par exemple nous aurions pu introduire la variable « humidex » :

$$humidex = T_{air} + 0.555[6.11e^{5417.7530*\left(\frac{1}{273.16}-\frac{1}{263.15+T_d}\right)} - 10]\}$$

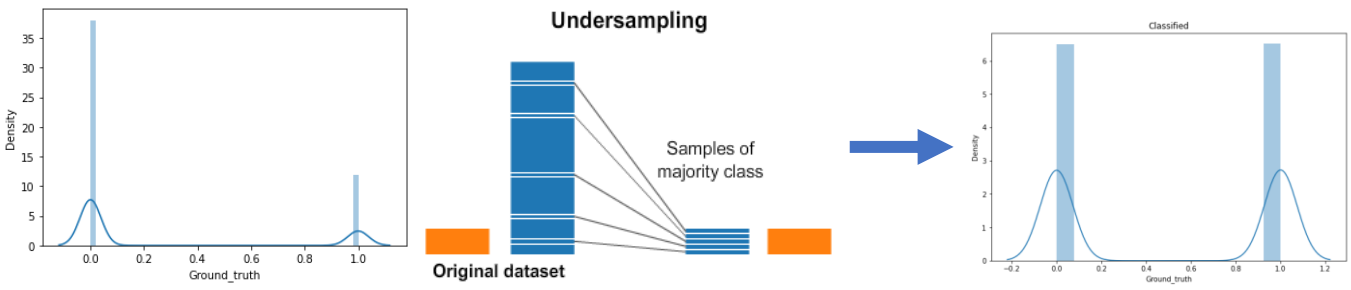
VIII. Autres pistes explorées

La classification binaire

Nous avons également essayé de séparer le jeu de données en 2 classes (0 si Ground_truth < moyenne, et 1 sinon). Nous avons arbitrairement choisi la moyenne comme threshold car elle nous donnait de meilleurs résultats mais on aurait pu prendre n'importe quelle valeur entre le min et le max :

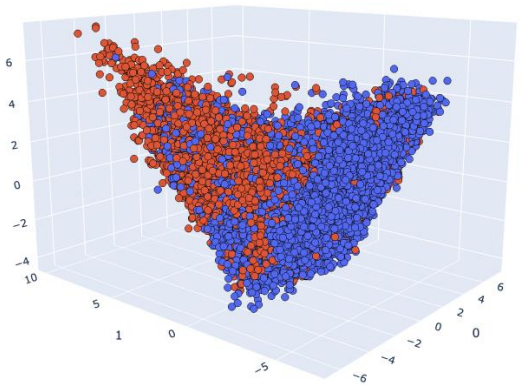


Ensuite, comme les classes étaient déséquilibrées, il a fallu faire de l'undersampling :



Nous avons testé plusieurs modèles de classification dont le meilleur (RandomForestClassifier) nous a donné une **précision de 89%**. Nous n'avons pas poursuivi cette piste (par souci de temps) mais si nous l'avions fait, nous aurions dû entrainer deux modèles de régression différents séparément sur un jeu de données séparé en deux par la classe (un jeu de classe 0 et un de classe 1), puis pour nos soumissions avoir dans notre pipeline la classification d'une observation, et selon le résultat (0 ou 1) la prédiction par le modèle entrainé sur la classe correspondante. On peut modéliser en 3 dimensions la séparation entre les deux classes :

PCA



Courbe ROC