RAPPORT DU PROJET « DÉFI-IA 2022 »
ÉLABORÉ PAR :

Thomas FRAMERY
Yoann MAAREK
Dorian VOYDIE

Dans le cadre du Mastère Spécialisé VALDOM

II. Table des matières

I. Introduction

Dans le cadre de l'UE « IA Frameworks » du master 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. Vous pouvez trouver les données sur le site MeteoNet : https://meteonet.umr-cnrm.fr/. L'évaluation de cette compétition était effectuée à travers la plateforme Kaggle : https://www.kaggle.com/c/defi-ia-2022

II. 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.

Initialement, METEO FRANCE nous a fourni 2 datasets d'entrainement à travers la plateforme Kaggle :

- X_station_train (= X_train)
- Y_train

Figure 1 : Les 5 premières lignes du dataset X_station_train

	number_sta	date	ff	t	td	hu	dd	precip	Id
0	14066001	2016-01-01 00:00:00	3.05	279.28	277.97	91.4	200.0	0.0	14066001_0_0
1	14066001	2016-01-01 01:00:00	2.57	278.76	277.45	91.4	190.0	0.0	14066001_0_1
2	14066001	2016-01-01 02:00:00	2.26	278.27	277.02	91.7	181.0	0.0	14066001_0_2
3	14066001	2016-01-01 03:00:00	2.62	277.98	276.95	93.0	159.0	0.0	14066001_0_3
4	14066001	2016-01-01 04:00:00	2.99	277.32	276.72	95.9	171.0	0.0	14066001_0_4

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_0_0 → Heure

Jour

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

Figure 2 : Les 5 premières lignes du dataset Y_train

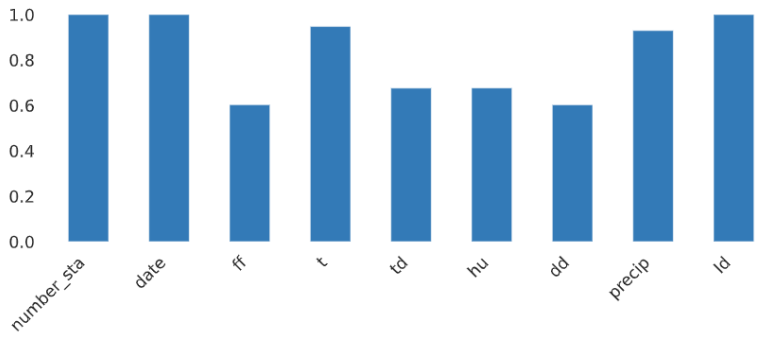
	date	number_sta	Ground_truth
Id			
14066001_0	2016-01-02	14066001	3.4
14126001_0	2016-01-02	14126001	0.5
14137001_0	2016-01-02	14137001	3.4
14216001_0	2016-01-02	14216001	4.0
14296001_0	2016-01-02	14296001	13.3

Ici, on voit que contrairement à précédemment, l'identifiant contient le numéro de station ainsi que le jour.
Par conséquent, il va falloir effectuer un travail de preprocessing sur X_train et plus précisément au niveau de l'identifiant pour qu'il contienne seulement le numéro de station et le jour.

Statistiques du jeu X_train

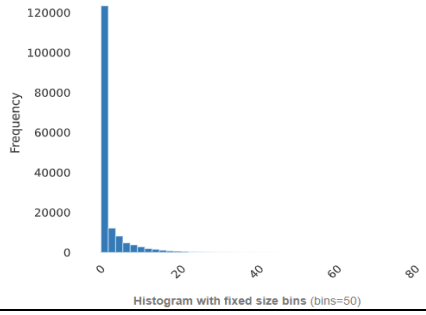
Number of Variables	9
Number of rows	4409474
Missing cells (%)	17.4%
Variable types	Numerical : 7 Categorical : 2 (date & number_sta)

Missing Values (in **blue** percentage of not NaN values) :



Statistiques du jeu Y_train (target feature = « Ground_truth »)

Number of Variables	4
Number of rows	183747
Missing cells (%)	2.9%
Variable types	Numerical : 2 Categorical : 2 (Id & number_sta)



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.

Preprocessing

Figure 3 : Fonction de preprocessing concernant X_train

```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

Figure 4 : Application du preprocessing sur X_train

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

Figure 5 : Fonction de preprocessing concernant Y_train

```
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%

Figure 6 : Application du preprocessing sur Y_train

	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.

Figure 7 : Création du trainset

```
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

Figure 8 : Visualisation du trainset

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.

Figure 9 : Imputation

```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

Figure 10 : Encodage

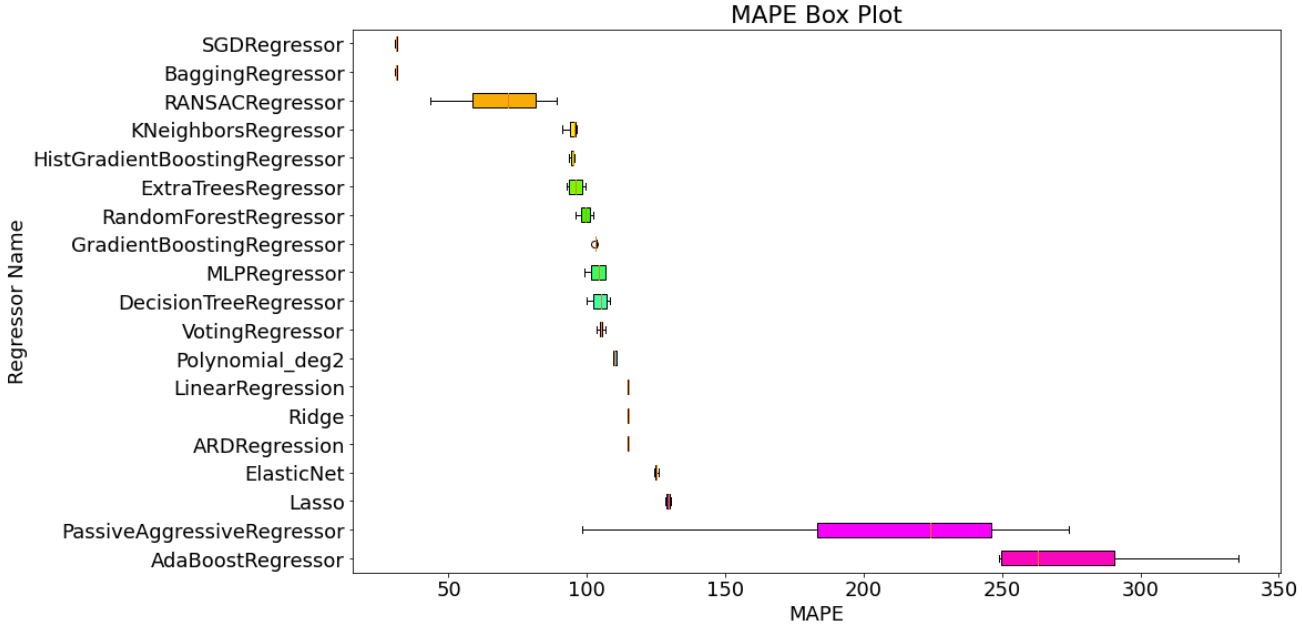
```python
# Imputation et encodage
trainset = train_imputation(trainset)
encoder = LabelEncoder()
trainset["number_sta"] = encoder.fit_transform(trainset["number_sta"].astype(int))
trainset.head()
```

Encodage de la seule colonne qualitative : number_sta

IV. Modélisation

En l'état actuel du jeu de données, nous avons réussi à entrainer plusieurs modèles de machine learning via la bibliothèque SKLearn et voici les résultats (MAPE) :

Figure 11 : Boxplot MAPE (sans AROME et ARPEGE) (N_FOLDS = 10)



Les meilleurs résultats sont ceux du KNRegressor, RandomForestRegressor, ExtraTreesRegressor, à savoir que le SGD ainsi que le BAGGING supposés être aux alentours de 35 ne prédisent en réalité que 0 (impertinent).

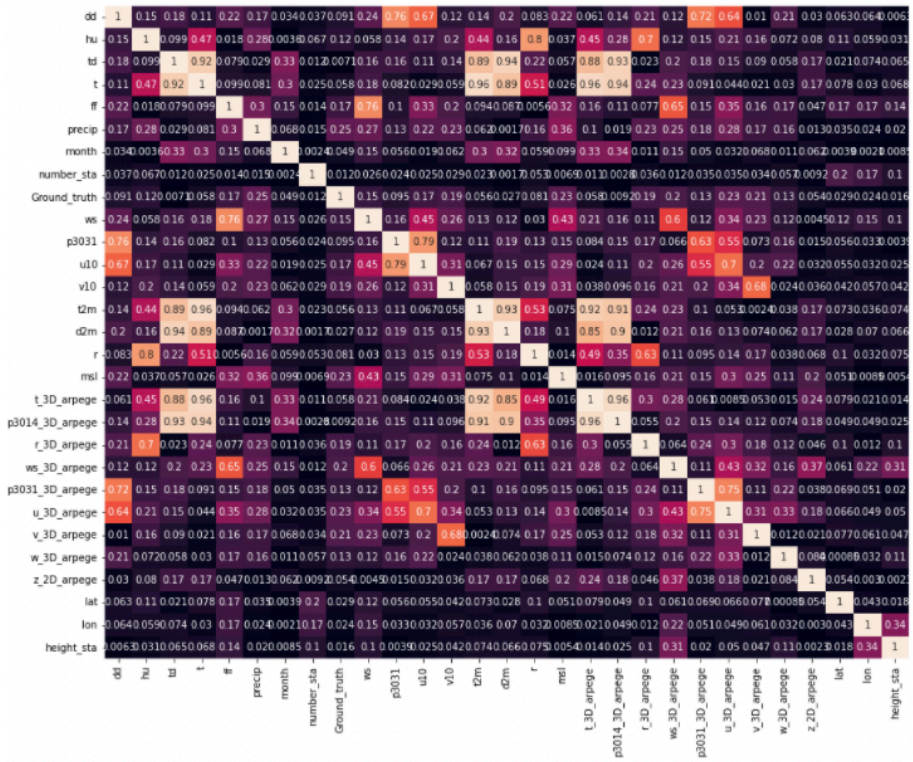
V. Introduction des datasets AROME et ARPEGE (site MeteoNet)

Pour ce Défi nous pouvions utiliser les données fournies directement dans Kaggle comme expliqué auparavant. Nous avons aussi des données disponibles sur les serveurs de MeteoNet. Ces données étaient composées de fichiers AROME 2D, Arpège 2D et de fichiers Arpège 3D. La différence majeure entre les fichiers AROME et arpège 2D est la précision (meilleure précision pour AROME).

Premièrement nous allons aborder les fichiers 2D. Nous avons utilisé uniquement les fichiers AROME car les variables sont les mêmes qu'ARPEGE mais la précision est meilleure.
Ces données sont assez particulières car ce sont des tableaux de données en 4 dimensions (latitude, longitude, heure, Id). Une fois la sélection dans ces dimensions faites nous avons accès à plusieurs variables. Notamment la vitesse et la direction du vent (ws, u10, v10), la température et la température de rosée (t2m, d2m), l'humidité (r), les précipitations sur l'heure en cours (tp) et la pression (msl).
Afin de récupérer toute ces valeurs il nous a fallut sélectionner les latitudes et longitudes les plus proches de chaque station. Une fois les coordonnées sélectionner nous faisons la moyenne de chaque donnée sur 24h pour chaque journée de l'année (il y a un fichier par jour).

Une fois les données 2D extraites nous passons aux données 3D avec ARPEGE. Ici la récupération est différente car les données ont une dimension supplémentaire (niveau isobare). Chaque jour contient deux fichiers, le premier contient différents niveaux de hauteur (en m). Chaque niveau est relié à un niveau isobare. Ce fichier nous sert donc juste à récupérer les niveaux isobares pour le second fichier. Une fois les isobares récupérer pour chaque station on peut sélectionner les données comme AROME 2D. Nous avons les mêmes variables que pour les fichiers 2D en plus de la hauteur en m (z) et la températures pseudo adiabatique (p3014).

Figure 12 : Table des corrélations complète



On a ici la table des corrélations du jeu de données complet (X_train + AROME + ARPEGE).
Nous pouvons voir plusieurs features en lien les unes avec les autres, notamment « td », « t », « t2m », « d2m », « t_3D_ARPEGE », « p3014_3D_ARPEGE »

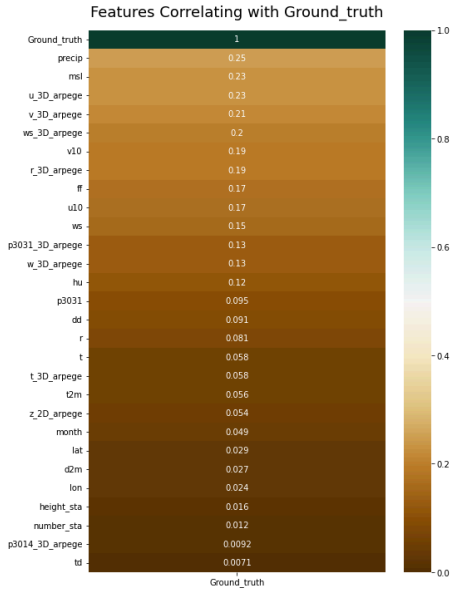
L'introduction de ces deux datasets a joué un rôle majeur dans les performances de nos prédictions :

Id	dd	hu	td	t	ff	precip	month	number_sta	Ground_truth
14066001_1	205.625000	82.300000	279.997500	282.936667	8.041250	3.4	1	14066001.0	11.7
14066001_10	209.541667	86.750000	277.497917	279.557917	5.408750	6.0	1	14066001.0	1.0
14066001_100	134.958333	76.408333	277.944583	282.112917	4.296250	11.6	4	14066001.0	5.6
14066001_101	167.208333	88.745833	281.003333	282.805000	1.754583	5.6	4	14066001.0	3.2
14066001_102	131.291667	85.595833	280.120000	282.479583	2.407083	3.2	4	14066001.0	0.8

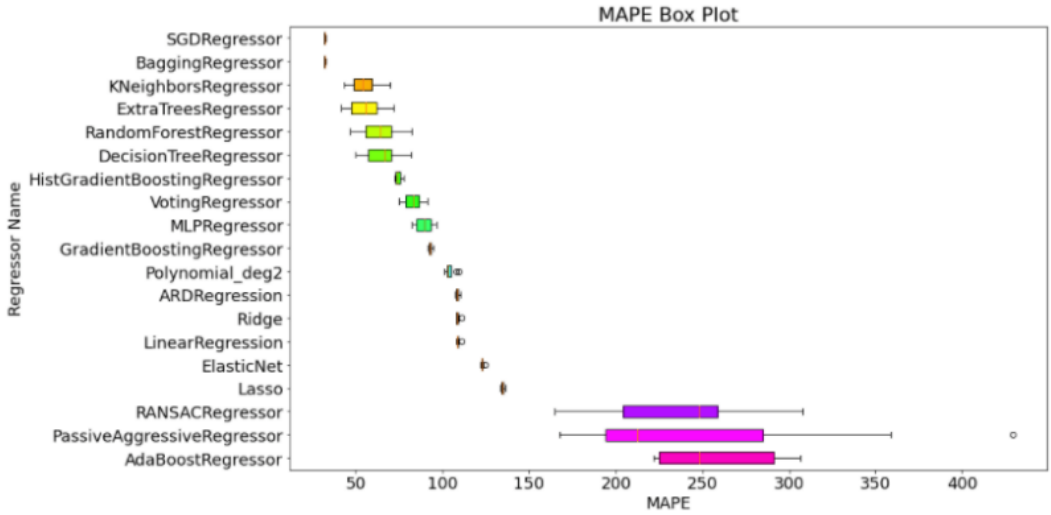
ws	p3031	u10	v10	t2m	d2m	r	msl
10.600895	212.232279	5.115749	7.128517	283.412374	279.927063	79.110545	100319.489583
8.646451	217.574748	5.072654	5.949552	280.440694	277.534383	81.884420	98650.770833
7.242371	122.227295	-6.064361	3.275607	281.778280	278.204183	78.805176	100403.875000
2.840101	146.672373	-0.918895	0.426130	283.172689	281.041545	86.636373	100574.708333
3.132152	129.255636	-1.010434	1.030473	282.560506	280.299581	85.856181	100714.625000

t_3D_arpege	p3014_3D_arpege	r_3D_arpege	ws_3D_arpege	p3031_3D_arpege	u_3D_arpege	v_3D_arpeg
283.509076	281.798254	82.191787	11.340173	198.634464	3.425157	9.25852
279.369284	280.615522	85.268490	14.430385	225.078585	9.116250	9.00608
280.451086	278.395106	78.088846	5.896205	132.943446	-4.325236	3.55461
283.448673	281.952206	86.179709	2.628501	159.980699	-0.801739	1.27523
283.517750	281.349753	81.380716	2.876033	157.384608	-0.612060	2.27168

w_3D_arpege	z_2D_arpege	lat	lon	height_sta
0.359990	173.634967	49.334	-0.431	2.0
0.313872	2982.975414	49.334	-0.431	2.0
0.135928	413.403895	49.334	-0.431	2.0
0.059505	423.532571	49.334	-0.431	2.0
0.063725	590.743509	49.334	-0.431	2.0



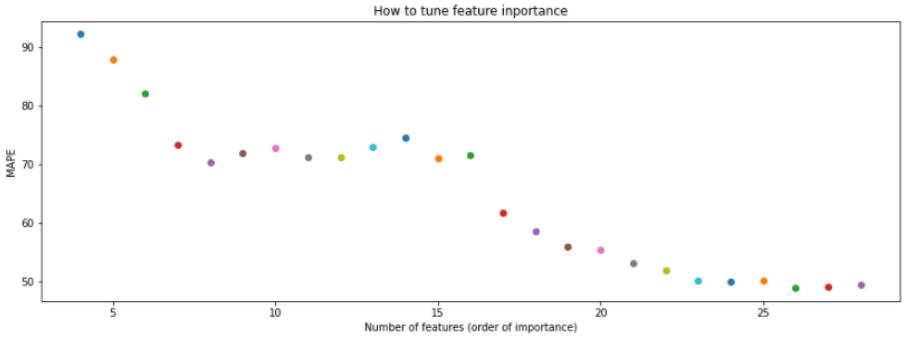
De 11 features nous sommes passés à 28. Ce qui a considérablement augmenté nos possibilités. Nous avons donc commencé par dresser une table des corrélations avec la Target. Les corrélations s'étendent de 0.0071 à 0.25, et on retrouve en majorité des éléments du jeu de données ARPEGE au sommet de la pile.
Avec les mêmes modèles de Machine Learning que sur le jeu de données initial, voici le boxplot des MAPE que nous avons retenu :

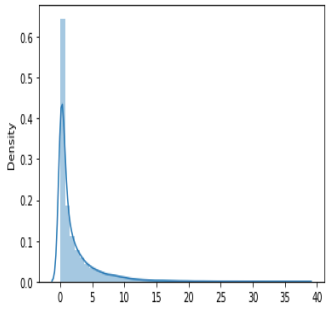


Les valeurs sont bien meilleures, nos algorithmes tournent à présent autour de 50 (contre 100 pour le dataset initial)
Nous avons ensuite effectué de la features importance selon les modèles que nous testions, en voici quelques exemples qui nous montrent la variation de la MAPE en fonction des features retenues par ordre d'importance :

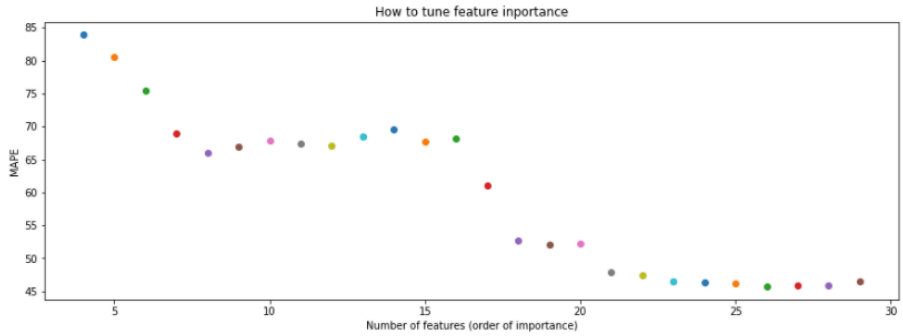
Distribution Prédiction

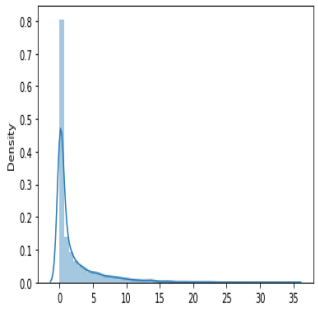
DecisionTreeReg





RandomForestReg

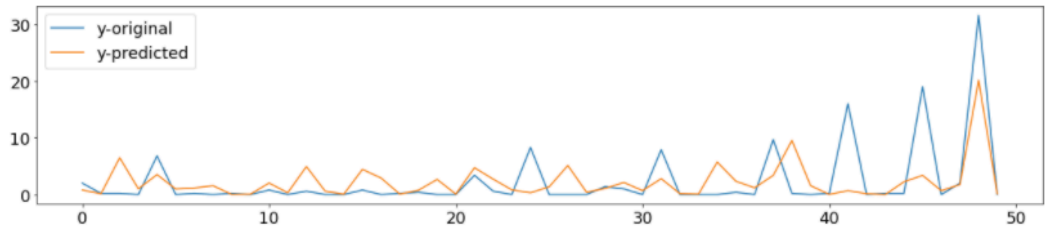




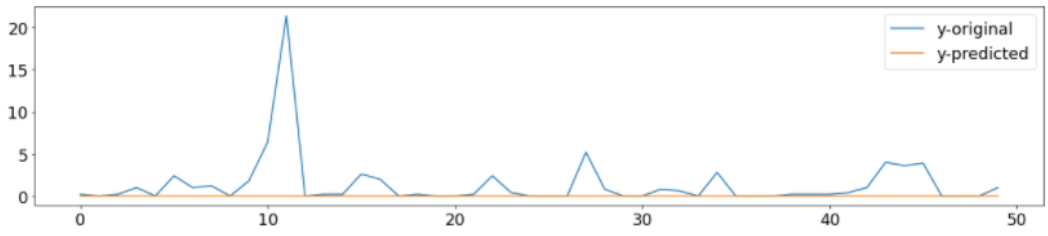
Globalement la MAPE se stabilise pour les 21 plus importantes features, à savoir :
['dd', 'hu', 't', 'ff', 'precip', 'month', 'ws', 'p3031', 'u10', 'v10', 't2m', 'r', 'msl', 't_3D_ARPEGE', 'r_3D_ARPEGE', 'ws_3D_ARPEGE', 'p3031_3D_ARPEGE', 'u_3D_ARPEGE', 'v_3D_ARPEGE', 'w_3D_ARPEGE', 'z_2D_ARPEGE']

Voici les prédictions que l'on peut avoir avec 3 modèles différents :

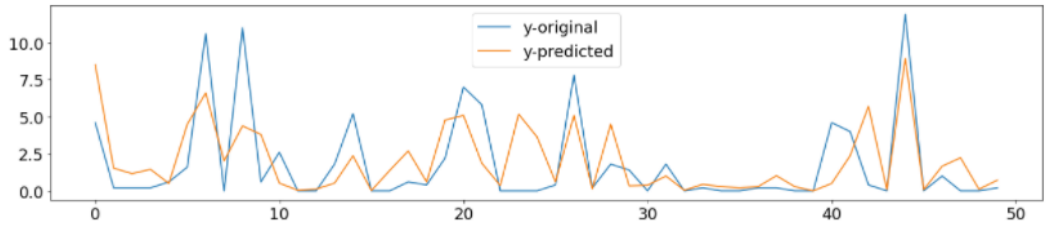
RandomForestRegressor



SGDRegressor



ExtraTreesRegressor



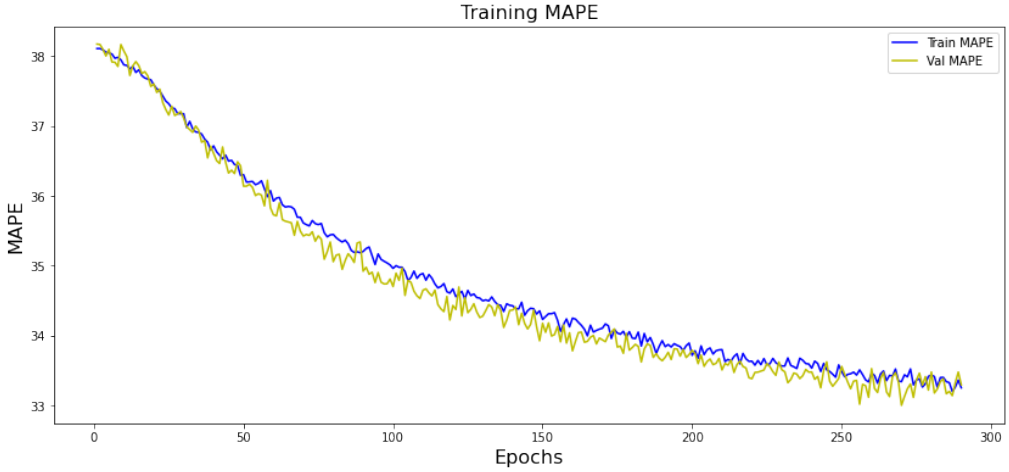
VI. Le Deep Learning

Le modèle

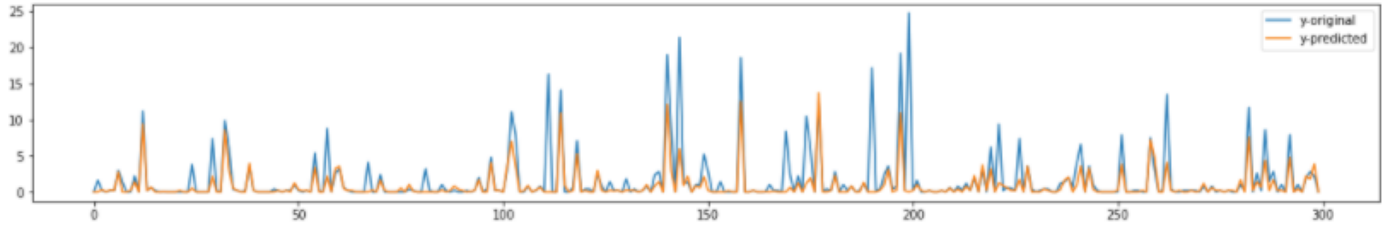
Le modèle nous ayant donné jusqu'alors les meilleurs résultats reste le réseau de neurones profond. En effet, alors que les modèles de Machine Learning classiques, même avec les hyper-paramètres optimisés peinaient à descendre en deçà de 35 de MAPE. Or grâce aux réseaux de neurone, **nous avons réussi à atteindre 31 de MAPE** pour sur nos soumissions Kaggle. Voici comment nous avons procédé :

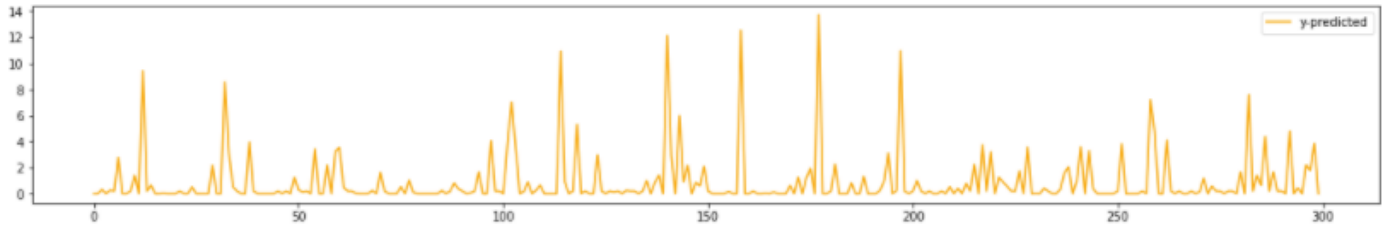
```
ann = Sequential()
ann.add(BatchNormalization())
ann.add(Dense(350, activation="relu", kernel_initializer='he_normal', kernel_regularizer=regularizers.l1(0.001)))
ann.add(Dropout(0.3))
ann.add(Dense(1024, activation="relu", kernel_initializer='he_normal', kernel_regularizer=regularizers.l1(0.001)))
ann.add(Dropout(0.3))
ann.add(Dense(512, activation="relu", kernel_initializer='he_normal', kernel_regularizer=regularizers.l1(0.001)))
ann.add(Dropout(0.3))
ann.add(Dense(128, activation="relu", kernel_initializer='he_normal', kernel_regularizer=regularizers.l1(0.001)))
ann.add(Dense(1))
callback = tf.keras.callbacks.EarlyStopping(monitor='val_loss', patience=40)
ann.compile(loss='mean_absolute_percentage_error', optimizer='adam', metrics=['mean_absolute_percentage_error'])
history = ann.fit(X_train,y_train,epochs=300, batch_size=1024, verbose = 2 ,validation_data=(X_val,y_val), callbacks=[callback])
```

Ann train MAPE : 33.26
Ann val MAPE : 33.3



Un des gros avantages de ce réseau de neurones et qu'il arrive à discerner les pics de pluie parmi la masse de jours sans pluie. Voici un aperçu de 300 prédictions aléatoires effectuées sur le jeu de validation (données jamais encore vues) comparé à la labellisation :

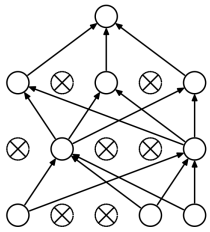




En revanche, l'un des problèmes de ce modèle et qu'il est légèrement surentrainé sur nos données d'entrainement, ce qui nous **empêche de passer la barre de 30 de MAPE sur nos soumissions Kaggle**.

Comment éviter le surentrainement

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.

VII. Problèmes rencontrés et pistes d'amélioration

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 29.48242 de MAPE

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 29.48242**. 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.

Surentrainement :

Nous avons rencontré un gros problème conséquent lorsque nous voulions soumettre une prédiction sur la compétition Kaggle, en effet si nous prenons l'exemple du Réseau de Neurone, **la MAPE calculée sur le validation-set était de 25.25**. Or une fois les prédictions ayant été effectuées avec le même modèle sur le test-set, la soumissions nous offrait **une MAPE autour de 30**. Cela vient très probablement d'un

surentrainement. Malgré les changements que nous avons opéré pour passer sous la barre des 30 de MAPE, le surentrainement était moindre mais notre score n'a pas franchi le cap.

Une des pistes d'amélioration aurait pu être de changer la pénalisation L1 en une combinaison de L1 et L2 afin d'apporter de la stabilité au modèle. Nous n'avons peut-être aussi simplement pas trouvé la meilleure architecture…

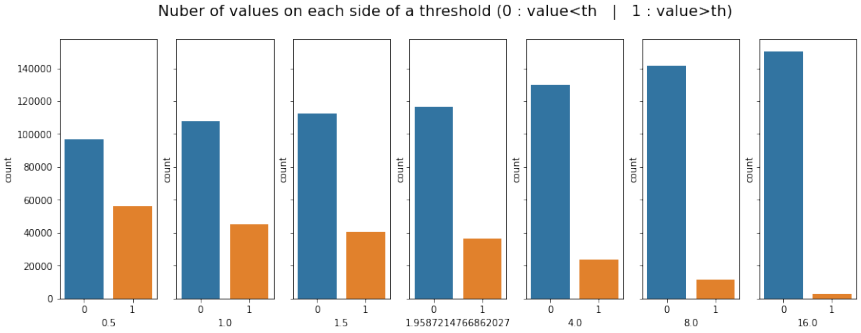
Équilibrage du jeu de données

Une des choses qui aurait pu nous aider à améliorer les performances de notre modèle aurait été d'équilibrer la distribution du jeu de données. En effet si on remonte en début de **page 4** et qu'on observe la distribution de Y_train, on voit qu'elle est centrée très proche de 0.

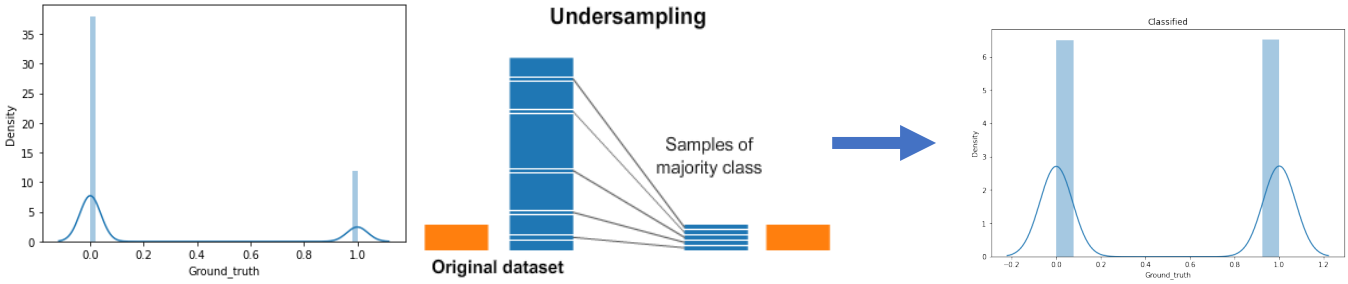
VIII. Autre piste explorées

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 :

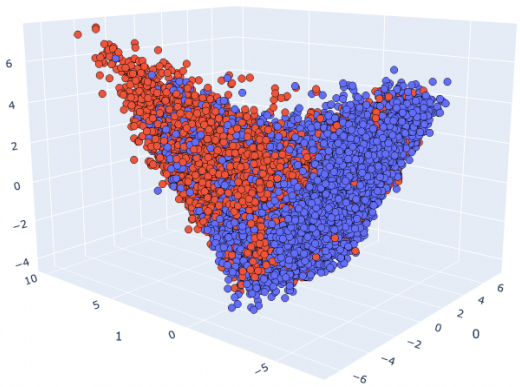


Ensuite, comme les classes étaient déséquilibrées, il a fallu faire de l'undersampling :



Nous avons testé plusieurs modèles de classification dont le meilleur (RandomForestClassifier) nous a donné une **<u>précision de 89%</u>**. 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