RAPPORT DU PROJET « DÉFI-IA 2022 »

ÉLABORÉ PAR :

Thomas FRAMERY

Yoann MAAREK

Dorian VOYDIE

Dans le cadre du Mastère Spécialisé VALDOM

Table des matières

[I. Introduction 3](#_Toc92646732)

[II. Analyse exploratoire des données 3](#_Toc92646733)

[Les jeux de données 3](#_Toc92646734)

[Statistiques du jeu X\_train 3](#_Toc92646735)

[Statistiques du jeu Y\_train (target feature = « Ground\_truth ») 4](#_Toc92646736)

[III. Transformation des datasets 4](#_Toc92646737)

[Preprocessing 4](#_Toc92646738)

[IV. Modélisation 6](#_Toc92646739)

[V. Introduction des datasets AROME et ARPEGE (site MeteoNet) 7](#_Toc92646740)

[VI. Le Deep Learning 10](#_Toc92646741)

[Le modèle 10](#_Toc92646742)

[Comment éviter le surentrainement 11](#_Toc92646743)

[VII. Problèmes rencontrés et pistes d’amélioration 11](#_Toc92646744)

[NaNs : 11](#_Toc92646745)

[Passer en dessous 29.48242 de MAPE 11](#_Toc92646746)

[Surentrainement : 11](#_Toc92646747)

[Équilibrage du jeu de données 12](#_Toc92646748)

[VIII. Autres pistes explorées 12](#_Toc92646749)

[La classification binaire 12](#_Toc92646750)

# 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>

# 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*

Une image contenant table

Description générée automatiquement

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 ».

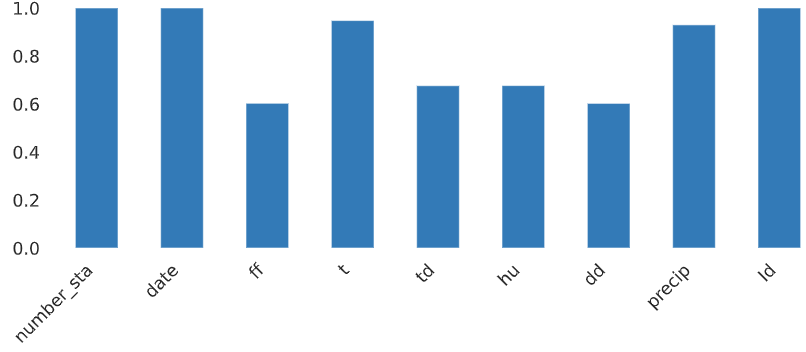
*Figure 2 : Les 5 premières lignes du dataset Y\_train*

|  |  |
| --- | --- |
| *Une image contenant table  Description générée automatiquement* | 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) |

# 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*

Une image contenant texte

Description générée automatiquement

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

Mise en ordre des colonnes et au format int

Création d’une colonne pour merge avec Y\_train

Tri des lignes chronologique

*Figure 4 : Application du preprocessing sur X\_train*

Attribut de jointure

Une image contenant table

Description générée automatiquement

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*

Une image contenant texte

Description générée automatiquement

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*

Une image contenant table

Description générée automatiquement

Attribut de jointure

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

*Figure 7 : Création du trainset*

Jointure des datasets

Une image contenant texte

Description générée automatiquement

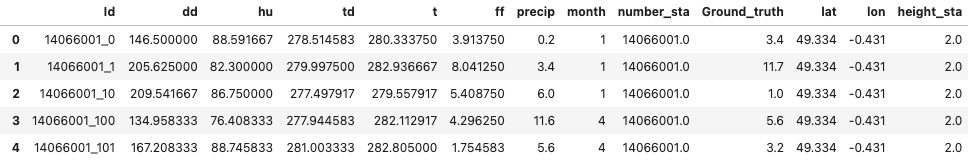
Jointure des coordonnées de stations

Moyenne des attributs de X\_train sur 24h car X\_train est décomposé en heures et Y\_train en jours.

Moyenne des features sur la journée

Somme des précipitations sur un jour

*Figure 8 : Visualisation du trainset*



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*

|  |  |
| --- | --- |
| Une image contenant texte  Description générée automatiquement | 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 |

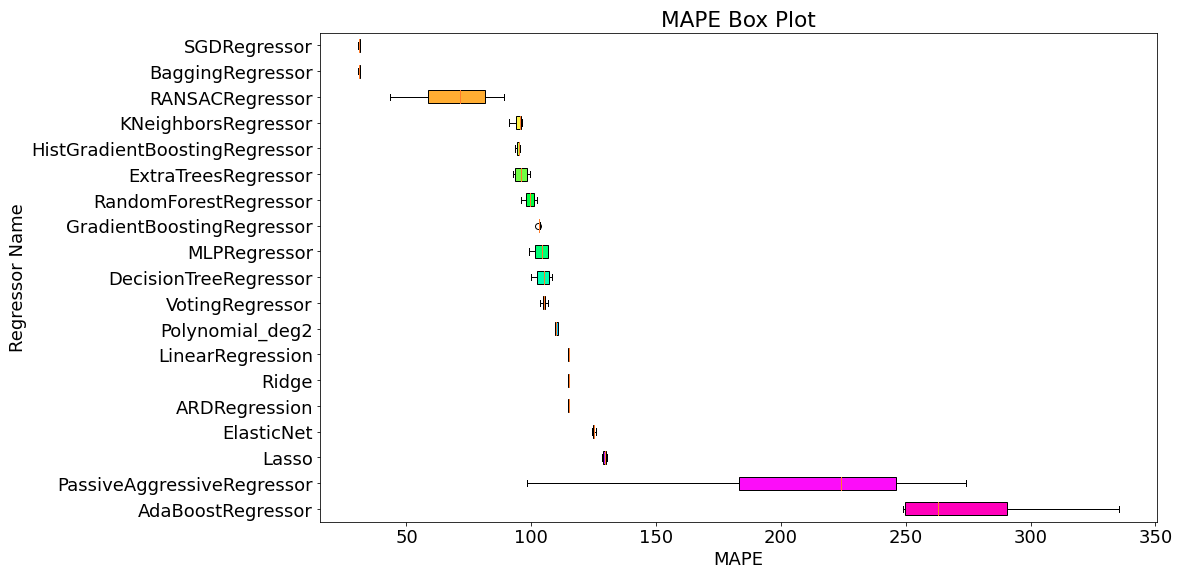
*Figure 10 : Encodage*

|  |  |
| --- | --- |
| Une image contenant texte  Description générée automatiquement | Encodage de la seule colonne qualitative : number\_sta |

# 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).

# 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).

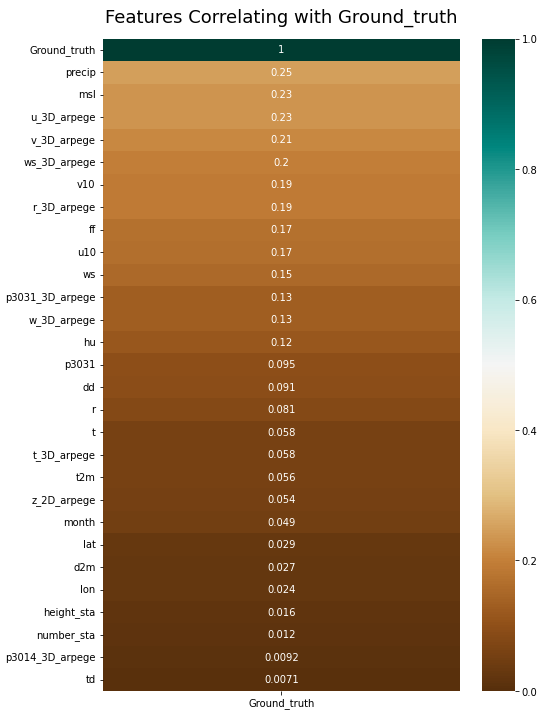
Afin de récupérer toute ces valeurs il nous a fallut sélectionner les latitudes et longitudes les plus proches de chaque station. Une fois les coordonnées sélectionner nous faisons la moyenne de chaque donnée sur 24h pour chaque journée de l’année (il y a un fichier par jour).

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 :





Une image contenant table

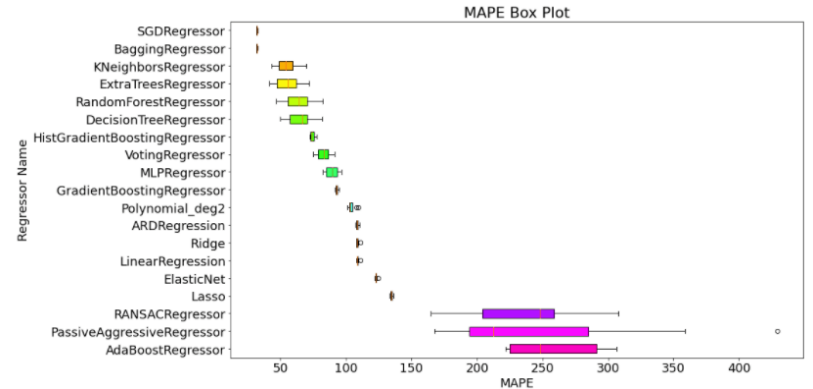
Description générée automatiquement

Une image contenant table

Description générée automatiquement

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 |  |

# 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 |  |

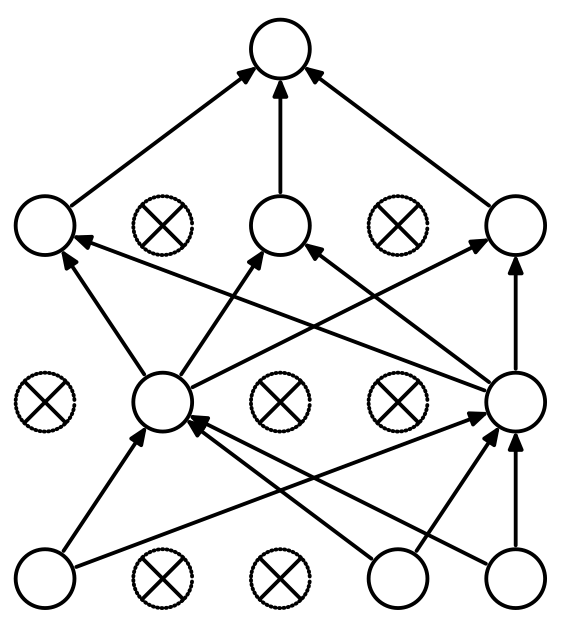
Un des gros avantages de ce réseau de neurones et qu’il arrive à discerner les pics de pluie parmi la masse de jours sans pluie. Voici un aperçu de 300 prédictions aléatoires effectuées sur le jeu de validation (données jamais encore vues) comparé à la labellisation :

Une image contenant texte, ciel, jour, plusieurs

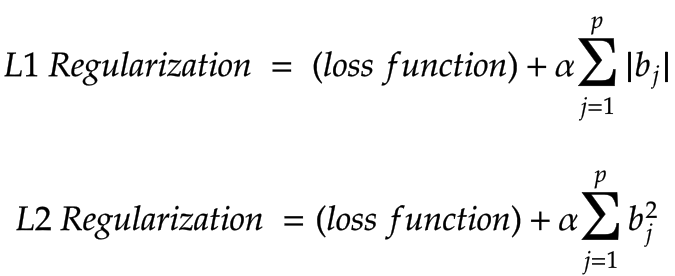
Description générée automatiquement

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.
   1. Pour la **L1-Regularization**la somme des poids en valeur absolue multiplié par une constante α
   2. Pour la **L2-Regularization** la somme des poids au carré multiplié par une constante



1. **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.

# 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.

# Autre piste explorées

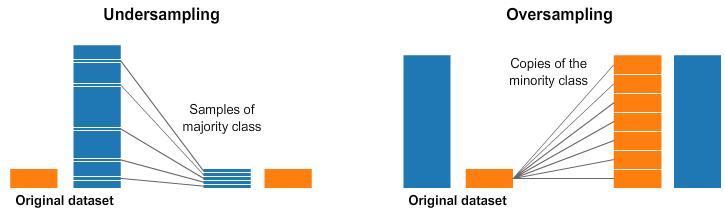
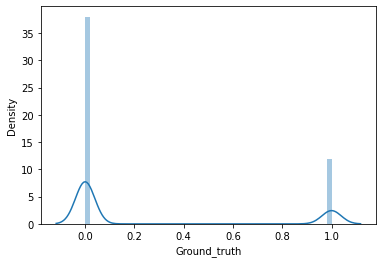
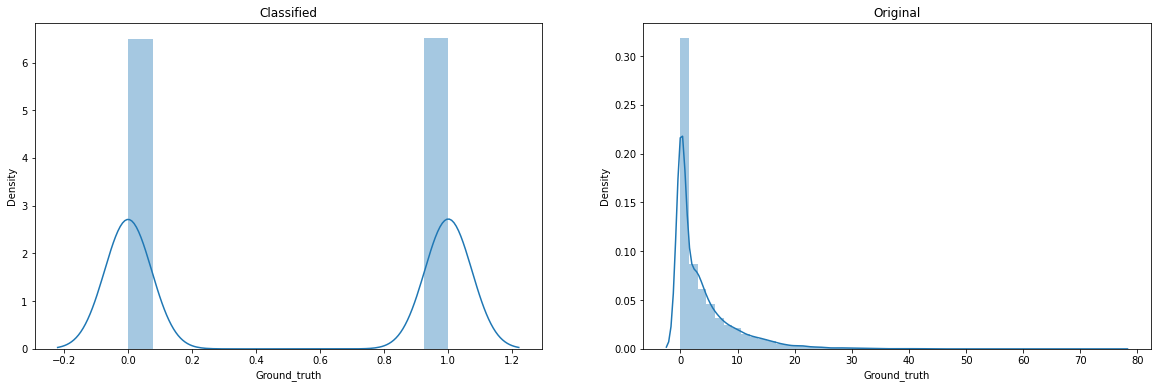
## La classification binaire

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 :

Une image contenant texte, clipart

Description générée automatiquement

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 |
|  |  |