Défi IA 2022

Organisé par Météo France

« Prédictions des précipitations quotidiennes accumulées sur des stations d'observations au sol »

Master Valdom

Alvarez Eloïse, Berjon Pierre, Daurat Jason, Pradeau Rémi

Table des matières

I) Introduction

a) Cadre

De nos jours, il est difficile de prédire tous les événements météorologiques et force est de constater que leur nombre croît continuellement dû notamment au dérèglement climatique. Afin de garantir la sécurité des personnes et des biens, il est crucial de pouvoir prédire avec précision ces événements qui peuvent avoir des conséquences plus ou moins importantes (tempêtes, tornades, orage, brouillard, grêle…).

Le défi IA proposé par Météo France nous place dans le cadre où nous devons aider un ingénieur prévisionniste dans sa prise de décision à l'aide d'algorithmes d'intelligence artificielle. Météo France a déjà testé certaines techniques de Machine Learning comme les forêts aléatoires et le but est de les améliorer en proposant de nouvelles méthodes de traitement des données ainsi que de nouveaux algorithmes d'apprentissage.

Concrètement, l'objectif est de prédire la pluie cumulée sur 24 heures sur des points bien précis en France. Il s'agit de stations d'observations placées au sol ou dans la mer. Pour le défi, Météo France a restreint le cadre uniquement au quart nord-ouest de la France.

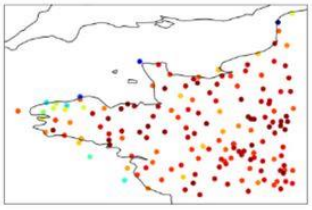


Figure 1 : Localisation des stations d'observations

b) Sources de données et variables

b.1) Mesures des stations d'observations

Les stations d'observations sont munies de nombreux équipements de mesure permettant ainsi d'obtenir des informations comme l'humidité, la température, le vent … Ces paramètres vont nous aider à entraîner notre modèle de prédiction.

Fichier X_train_station.csv : **ANNEXE 1** pour la description des variables.

Ainsi pour chaque jour et à chaque heure, les stations fournissent des mesures de ces différents paramètres avec parfois des valeurs manquantes. En effet, des capteurs peuvent tomber en panne. De plus les stations sont éphémères, certaines sont présentes qu'à partir de 2018, et d'autres de 2016 à 2017.

b.2) Systèmes prévisionnistes de Météo France

En plus de ces données de mesures, nous possédons également les données des systèmes prévisionnistes de Météo France *Arpege (maillage grossier de points discrétisés en latitude et longitude) et Arome (maillage plus fin sur une grille de discrétisation 3D longitude x latitude x height).* Il s'agit de fichiers netCDF au format .nc dans le dossier X_forecast et sur /meteonet.umr-cnrm.fr.

Pour la description des variables : **ANNEXE 1**

Comme pour les stations d'observations, il est à noter la présence de valeurs manquantes à certaines dates.

II) Pré-traitement et premier modèle prédictif

Avant d'appliquer un algorithme de Machine Learning pour réaliser des prédictions, il est nécessaire de travailler les datasets en effectuant un certain nombre de transformations. Le but est ainsi d'obtenir un jeu de données plus cohérent et avec une gestion intelligente des valeurs manquantes *NaN*. Dans la suite du rapport, les dataframes seront surlignés en gras pour faciliter la lecture.

Les parties a) et b) présentent *la pipeline* que nous avons utilisée pour préparer le X_train et y_train. La partie c) présente les premiers résultats qui ont été soumis sur Kaggle.

a) Extraction des données AROME et ARPEGE

Une première jointure est faîte entre le dataframe stockant le fichier *X_station_train.csv* et *station_coordinates.csv* sur la colonne 'number_sta'. Cela permet ainsi d'ajouter 3 nouvelles features à notre dataset : 'lat', 'lon', 'height_sta' en plus des variables 'ff', 't', 'td', 'hu', 'precip'.

Puis une nouvelle jointure entre ce dataframe et les données d'ARPEGE est créée. Le travail se fait alors sur une copie de ce dataframe résultant nommé **df**.

Le dataframe stockant les données du fichier *Y_train.csv* avec les targets 'Ground_truth' est nommé **y**.

Méthode Merge avec Arpege en prenant le plus proche voisin :

Sur le serveur de météonet, plusieurs fichiers sont mis à notre disposition pour enrichir le csv *X_station_train.csv* et tenter d'améliorer notre score. Ils contiennent des prédictions météorologiques faites par les modèles AROME et ARPEGE. Ces prédictions sont faites à chaque point d'une grille correspondant à des coordonnées GPS.

Les différentes données (voir annexe) dans les fichiers 2D sont organisées suivant trois paramètres, l'heure, la latitude et la longitude pour laquelle la prédiction est faite. On a décidé de prendre, pour chaque heure, la prédiction la plus proche de chaque station afin de la rajouter au csv.

Comme il y avait un fichier par jour, il a fallu optimiser le code afin d'ouvrir qu'une seule fois chaque fichier. On a donc fait en sorte de prendre les données pour toutes les stations, pour toutes les heures de la journée en une seule ouverture de fichier avant de les ajouter à un nouveau dataframe. Pour une station, 24 lignes étaient créées représentant les 24 heures de la journée. Enfin on a merge ce nouveau dataframe à l'ancien en faisant correspondre le numéro de station, le jour et l'heure.

b) Transformations et gestion des NaN

Dans cette partie nous présentons comment le jeu de données a été transformé proprement pour pouvoir ensuite entraîner nos modèles.

Travail sur le dataframe **df** :

La première étape est de modifier l'identifiant 'Id' pour seulement avoir le numéro de station et le jour :

```
[90] df['Id'] = df['Id'].apply(lambda x: x.split('_')[0] + '_' + x.split('_')[1])
```

La colonne ‘month’ est créée au préalable à partir de la colonne ‘date’. Il peut en effet plus ou moins pleuvoir selon la saison, il s’agit d’une feature potentielle supplémentaire qui est également présente dans le fichier de test : *X_station_test.csv.*

Avant de gérer les NaN, les données sont rassemblées par jour et par station à l’aide de la méthode Dataframe.groupby() et du nouvel ‘Id’ que l’on vient de créer.

Au début, un simple groupby(‘Id’).sum() était utilisé pour toutes les variables mais le fait de sommer par jour n’avait pas forcément de sens. Par exemple pour ‘ff’ la vitesse du vent, il est préférable de prendre la moyenne sur la journée plutôt que de sommer. Par conséquent, un dictionnaire est défini au préalable pour préciser la méthode à appliquer au cas par cas :

```
df['month'] = df['month'].astype('int32')
dict_agg = {'ff':'mean', 't':'mean', 'td':'mean', 'hu':'mean', 'dd':'max', \
            'precip':'sum', 'lat':'min', 'lon':'min', 'height_sta':'min', \
            'month':'mean' ,'ws':'mean', 'p3031':'mean', 'u10':'mean', \
            'v10':'mean', 't2m':'mean', 'd2m':'mean', 'r':'mean', 'tp':'sum', \
            'msl':'min'}
df = df.groupby('Id').agg(dict_agg).reset_index()
df.head()
```

Les données sont donc groupées par numéro de station et par jour en fonction de la méthode définie dans ce dictionnaire puis, triées par indice de station et de jour.

```
df.sort_values(by='Id', axis=0, inplace=True)
df = df.fillna(method='ffill')
```

	Id	ff	t	td	hu	dd	precip	lat	lon	height_sta	month	ws	p3031	u10	v10	t2m	d2m	r	tp	msl
0	14066001_0	3.913750	280.333750	278.514583	88.591667	200.0	0.2	49.334	-0.431	2.0	1	5.896024	154.818732	-2.813736	4.891051	278.944803	277.507755	90.810500	5.134943	100462.060
1	14066001_1	8.041250	282.936667	279.997500	82.300000	241.0	3.4	49.334	-0.431	2.0	1	11.719456	213.644654	5.813690	7.905785	283.038778	280.671205	85.270216	41.192624	99930.560
2	14066001_10	5.408750	279.557917	277.497917	86.750000	249.0	6.0	49.334	-0.431	2.0	1	7.756676	219.109915	4.577999	5.469955	280.441820	278.345820	86.642426	7.516652	98268.055
3	14066001_100	4.296250	282.112917	277.944583	76.408333	199.0	11.6	49.334	-0.431	2.0	4	5.786060	127.964630	-4.696062	3.143035	281.193274	278.435246	83.250162	9.534248	100076.750
4	14066001_101	1.754583	282.805000	281.003333	88.745833	286.0	5.6	49.334	-0.431	2.0	4	2.348911	144.967537	-1.026263	0.728844	283.019172	281.082953	88.303185	14.615968	100257.940

Les *NaN* sont ensuite remplacés par la dernière observation valide précédente pour chacune des colonnes du dataframe **df** grâce à la méthode ‘ffill’ :

```
df.isna().sum()
```

```
number_sta          0
date                0
ff            1750817
t              231013
td            1428352
hu            1425877
dd            1752650
precip         310298
Id                  0
lat                 0
lon                 0
height_sta          0
ws             567077
p3031          567077
u10            567077
v10            567077
t2m            567077
d2m            567077
r              567077
tp             727179
msl            567077
month               0
dtype: int64
```

```
df.isna().sum()
```

```
Id          0
ff          0
t           0
td          0
hu          0
dd          0
precip      0
lat         0
lon         0
height_sta  0
month       0
ws          0
p3031       0
u10         0
v10         0
t2m         0
d2m         0
r           0
tp          0
msl         0
ind_sta     0
ind_day     0
dtype: int64
```

Le nombre de lignes de **df** est ainsi réduit de 4 409 474 à 183 747.

Concernant le fichier *y_train.csv* stocké dans le dataframe **y_train** :

```
y_train = pd.read_csv(path_to_train + 'Y_train.csv')
y_train
```

	date	number_sta	Ground_truth	Id
0	2016-01-02	14066001	3.4	14066001_0
1	2016-01-02	14126001	0.5	14126001_0
2	2016-01-02	14137001	3.4	14137001_0
3	2016-01-02	14216001	4.0	14216001_0
4	2016-01-02	14296001	13.3	14296001_0
...	...	...	...	...
183742	2017-12-31	86137003	5.0	86137003_729
183743	2017-12-31	86165005	3.2	86165005_729
183744	2017-12-31	86272002	1.8	86272002_729
183745	2017-12-31	91200002	1.6	91200002_729
183746	2017-12-31	95690001	1.2	95690001_729

183747 rows × 4 columns

Nous décidons de supprimer les NaN dans la target que l'on cherche à prédire 'Ground_truth' ce qui réduit le nombre de lignes de 183 747 à 162 107.

```
[27] y_train.isna().sum()
```

```
date              0
number_sta        0
Ground_truth    21640
Id                0
dtype: int64
```

```
y = y_train[['Id', 'Ground_truth']]
# Droping rows where Ground_truth is nan
y = y[y['Ground_truth'].notna()]
y.shape
```

Une jointure est ensuite réalisée sur **df** et **y** sur les identifiants pour qu'ils soient dans le même ordre.

```
[130] # Merging X_train and y['Ground_truth'] to have them in the same order
      df = df.merge(y[['Id', 'Ground_truth']], how='left', left_on='Id', right_on='Id')
```

Dorénavant, **df** a le même nombre de lignes que **y** (en ayant supprimé les NaN du 'Ground_truth') : 162 107.

Les mêmes transformations sont appliquées au jeu de données de validation. On note **X_val** le dataframe stockant les données du fichier *X_station_test.csv.* Il est à noter tout de même que la méthode 'bfill' a été utilisée pour le fillna car le fichier de validation a des NaN présents dans les premières lignes.

Après avoir terminé les transformations, nous générons les datasets qui serviront à entraîner nos modèles **: X_train** et **y_train**.

Important : Toutes les valeurs manquantes ont désormais été traitées et il est par conséquent possible de calculer le score MAPE.

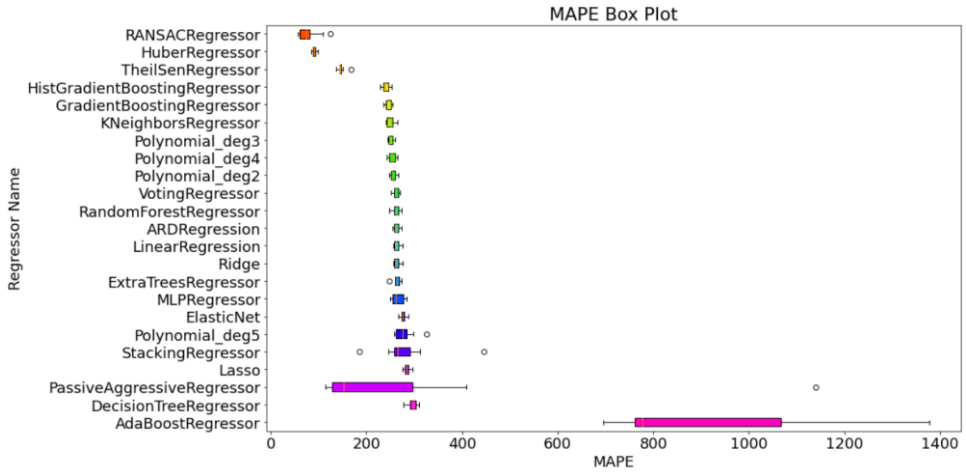
```
X_train_base = df.drop(columns=['Id', 'Ground_truth', 'ind_sta', 'ind_day'], axis=1)

y_train_base = df['Ground_truth']
```

```
X_train_base
```

	ff	t	td	hu	dd	precip	lat	lon	height_sta	month	ws	p3031	u10	v10	t2m	d2m	r	tp	msl
0	3.913750	280.333750	278.514583	88.591667	200.0	0.2	49.334	-0.431	2.0	1	5.896024	154.818732	-2.813736	4.891051	278.944803	277.507755	90.810500	5.134943	100462.060
1	8.041250	282.936667	279.997500	82.300000	241.0	3.4	49.334	-0.431	2.0	1	11.719456	213.644654	5.813690	7.905785	283.038778	280.671205	85.270216	41.192624	99930.560
2	5.408750	279.557917	277.497917	86.750000	249.0	6.0	49.334	-0.431	2.0	1	7.756676	219.109915	4.577999	5.469955	280.441820	278.345820	86.642426	7.516652	98268.055
3	4.296250	282.112917	277.944583	76.408333	199.0	11.6	49.334	-0.431	2.0	4	5.786060	127.964630	-4.696062	3.143035	281.193274	278.435246	83.250162	9.534248	100076.750
4	1.754583	282.805000	281.003333	88.745833	286.0	5.6	49.334	-0.431	2.0	4	2.348911	144.967537	-1.026263	0.728844	283.019172	281.082953	88.303185	14.615968	100257.940
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
183742	7.699167	279.041667	276.800000	85.587500	259.0	2.4	49.108	1.831	126.0	12	7.960984	203.161501	2.350976	6.917261	278.922062	277.482138	90.486717	69.282720	98747.510
183743	8.134167	277.274583	275.160417	86.275000	310.0	3.2	49.108	1.831	126.0	12	8.757726	252.789489	5.742630	0.401429	277.895645	275.938725	87.190832	88.978624	98083.400
183744	3.172500	275.019583	273.014583	86.816667	295.0	0.0	49.108	1.831	126.0	12	3.794898	246.215240	3.369965	1.223398	274.385175	273.130522	91.156639	0.000877	100070.695
183745	6.341250	277.452500	275.341250	86.637500	282.0	4.4	49.108	1.831	126.0	12	6.115135	218.250085	3.535934	3.890409	276.477835	275.077257	90.506057	71.976383	100102.625
183746	8.529583	285.330833	283.395417	88.175000	256.0	5.4	49.108	1.831	126.0	12	7.391012	228.892048	5.414123	4.513924	283.692524	282.766071	94.025580	134.490071	100092.516

162107 rows × 19 columns

Au début du projet, nous ne faisions pas forcément les bonnes transformations. Par exemple les données étaient rassemblées par jour et par station en prenant la somme ou le max de la variable. De plus on supprimait les 13 stations pour lesquelles nous avions remarqué que 'precip' valait toujours NaN ce qui nous faisait perdre des informations et des points en lesquels on voulait prédire. Enfin les NaN étaient remplacés par la moyenne de la variable. Par conséquent les scores MAPE des modèles explosaient pour la plupart :



Grâce aux transformations plus 'propres' définies précédemment dans cette partie, nos scores ont été largement améliorés. (En dessous de 60)

c) Résultats

Comme l'illustre la figure précédente, nous avons testé plus d'une quinzaine de modèles tels que : LinearRegression, Ridge, Lasso, ElasticNet, RandomForestRegressor, KNeigborsRegressor, MLPRegressor, RANSACRegressor, LGBM, …

Après avoir défini notre propre fonction de score MAPE, chaque modèle était entraîné puis évalué par **validation croisée** afin d'avoir davantage confiance en nos résultats : KFold de sklearn.model_selection avec 10 folds.

C'est le modèle RANSAC qui a été choisi pour effectuer les soumissions Kaggle avec comme **features** : `['ff', 't', 'td', 'hu', 'dd', 'precip']`

Le modèle est normalisé lors de la création de l'instance :

```
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
```

```
ransac_model = make_pipeline(StandardScaler(),
                             RANSACRegressor(random_state=0))
ransac_model.fit(X_fit, Y_fit)
```

La moyenne du MAPE sur les 10 folds est pour l'entrainement de 37.148 +- 9.193 et pour la validation de 37.271 +- 9.413. Sur Kaggle nous avons obtenu le score de **39.8** ce qui est cohérent en termes d'ordre de grandeur.

```
Génère le seed 199 pour l'itération 0
Mape fit pour le fold 1 : 35.235
Mape val pour le fold fold 1 : 35.100
----------------------------

Génère le seed 395 pour l'itération 1
Mape fit pour le fold 2 : 31.000
Mape val pour le fold fold 2 : 31.008
----------------------------

Génère le seed 223 pour l'itération 2
Mape fit pour le fold 3 : 36.830
Mape val pour le fold fold 3 : 36.932
----------------------------

Génère le seed 869 pour l'itération 3
Mape fit pour le fold 4 : 62.233
Mape val pour le fold fold 4 : 62.991
----------------------------
...
Génère le seed 723 pour l'itération 7
Mape fit pour le fold 8 : 30.974
Mape val pour le fold fold 8 : 31.126
----------------------------

Génère le seed 876 pour l'itération 8
Mape fit pour le fold 9 : 31.107
Mape val pour le fold fold 9 : 31.076
----------------------------

Génère le seed 830 pour l'itération 9
Mape fit pour le fold 10 : 43.963
Mape val pour le fold fold 10 : 44.294
----------------------------

MAPE score (moyenne) pour le fit : 37.148 ± 9.193
MAPE score (moyenne) pour le val : 37.271 ± 9.413
```

Remarques :

Le fait d'ajouter les variables d'Arpege au features apporte de la complexité et le modèle donne de moins bons scores de prédictions. De plus, le modèle Ransac a des difficultés pour prédire les valeurs élevées de Ground_truth :

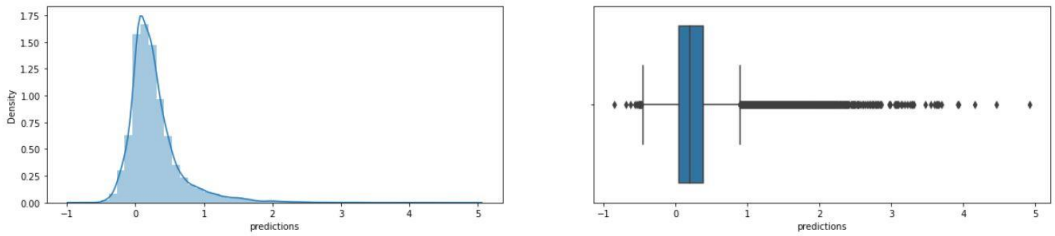


Figure 2 : Distribution des prédictions du Ransac

Il semble pourtant nécessaire d'avoir un modèle qui puisse prédire des valeurs parfois élevées selon la distribution du Ground_truth de **y** :

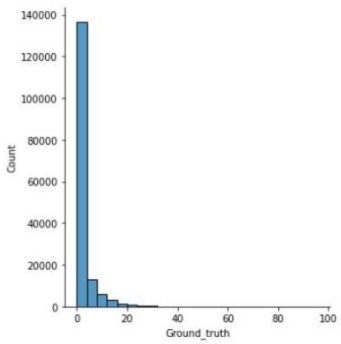


Figure 3 : Distribution du Ground_truth de Y_train

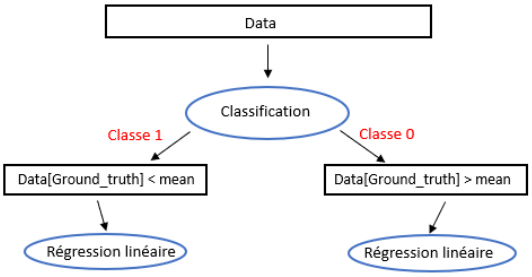
Le modèle Ransac prédit également des valeurs négatives de précipitations ce qui n'est pas logique. En prenant un échantillon de 20% du X_train, soit 32422 valeurs, il prédit : 5489 négatives et 26933 positives.

Une nouvelle soumission est faîte en remplaçant simplement les valeurs négatives par leur valeur absolue. Le score Kaggle obtenu est de **37.10**. Puis en remplaçant par 0 au lieu de la valeur absolue, le score est de **35.09**.

III) Autres tests

Comme nos modèles de Machine Learning ont des difficultés pour prédire les valeurs élevées de Ground_truth, nous avons eu l'idée de faire une classification binaire des valeurs du Ground_truth par rapport à la moyenne. On a également testé de classer par rapport à la valeur 1 ou à la médiane mais les scores de classification étaient moins bons. De cette façon, le dataset est séparé en deux classes et un régresseur est alors appliqué sur chacune de ces parties. Le principe est de "*Diviser pour régner*". Le meilleur classifieur retenu parmi ceux testés est le RandomForestClassifier avec un score de 84% de classifications justes.

Les deux régressions linéaires retenues pour les régresseurs sont des LinearRegression classiques.



Ainsi pour le **y_train** on a sur 183 747 lignes 139 575 dont le Ground_truth est inférieure à la moyenne et 44 173 où il est supérieur à la moyenne. En séparant les données en un jeu d'entraînement et un jeu de test à hauteur de 20% *(train_test_split),* le score MAPE sur les régressions pour la classe 1 est environ de 20 et pour la classe 0 autour de 60. Cependant lorsque l'on passe au dataset complet, les régressions donnent de moins bons scores avec un MAPE dépassant les 100. Cela vient notamment du fait que les régressions sont entraînées sur une partie seulement du dataset et par conséquent ont du mal à généraliser.

IV) Recherche de feature

Dans cette partie, nous allons voir les features que nous avons cherché afin d'améliorer le résultat de nos prédictions. Pour cela, nous nous sommes concentrés sur la signification des données et l'impact qu'elles pouvaient avoir sur les précipitations.

1) Température négative et positive

Dans un premier temps, nous nous sommes intéressés aux conditions météorologiques qui provoquent la pluie. Nous avons donc décidé de séparer les températures négatives (qui provoque des précipitations de type neige et grêle) et les températures positives (avec des précipitations de type pluie). Pour cela, nous avons ajouté une colonne et nous avons utilisé la donnée température "t" qui est en Kelvin que nous avons comparé à 273,6K. Si elle était inférieure alors on ajoutait un 0 dans la nouvelle colonne, sinon 1.

2) Pression atmosphérique

Lors de nos recherches, nous avons appris que la pression atmosphérique est utilisée pour prédire le risque de précipitation : lorsqu'elle diminue il y a plus de risque de pluie que

lorsqu'elle augmente. Nous avons donc ajouté une colonne pour la calculer à partir de la formule suivante :

$$P = P_0 . e^{-\frac{\mu.g.h}{R.T}}$$

Avec :

- P0 = la pression atmosphérique au niveau de la mer "msl"
- h = l'altitude de la station "height_sta"
- T = la température au niveau de la station "t"
- μ = Constante, masse molaire moyenne de l'air : M = 0.0289644 kg/mol
- g : Constante de la pesanteur terrestre, g = 9.80665 m/s2
- R : Constante des gaz parfaits R = 8.31432 J/K.mol

3) La température au point de rosée

La température au point de rosée correspond à la température à laquelle, pour une pression donnée, air devient saturé en vapeur d'eau. Lorsque l'air atteint cette température, l'eau se condense et cela forme des nuages ce qui peut augmenter le risque de précipitation. Nous avons créé une colonne supplémentaire qui permet de comparer ces deux températures : 1 si la température est supérieure à la température au point de rosée, 0 sinon.

4) Taux d'humidité

Nous nous sommes ensuite intéressés au taux d'humidité dans l'air ambiant. S'il est important nous pouvons penser qu'il impactera le risque de précipitation. Nous avons donc ajouté une colonne qui permettra de seuiller le taux d'humidité : 1 s'il est supérieur à 90%, 0 s'il est inférieur.

5) Précipitation du jour précédent

Dans les données, nous avons une information concernant le taux de précipitation du jour précédent. Nous nous sommes rendu compte que lorsque nous avons enlevé cette information des données, le modèle RANSAC que nous utilisons jusqu'à présent était beaucoup plus stable et plus précis. Nous avons donc fait le choix de ne plus utiliser cette donnée mais de la transformer pour qu'elle impacte positivement le modèle. Pour cela, nous avons ajouté une colonne qui indique "il a plu la veille ou non" : 1 si le taux de précipitation du jour précédent est supérieur à 0, 0 sinon.

6) Impact de la saison

Le taux de précipitation peut varier en fonction de la période de l'année. En effet, il y a plus de risque de pleuvoir en automne qu'en été. Nous avons donc séparé les données en quatre colonnes afin de représenter les 4 saisons. Chaque colonne sera représentative d'une seule saison et contiendra donc 0 ou 1 qui dépendra si le mois fait partie de cette saison ou non. Nous avons classé de la façon suivante :

- Décembre, Janvier, Février
- Mars, Avril, Mai
- Juin, Juillet, Août
- Septembre, Octobre, Novembre

7) Impact de la direction du vent

Lors de nos recherches, nous avons appris que le vent était aussi un indicateur du risque de précipitation. En effet, lorsque le vent vient de l'Est il y a plus de risque de précipitation que lorsqu'il vient de l'Ouest. Pour cela, nous avons utilisé la colonne donnant l'information sur la direction du vent et ajouté une nouvelle colonne tel que : si la direction du vent est comprise entre 90 et 270° alors c'est un vent de l'ouest donc 0 sinon c'est un vent de l'est donc 1.

8) Conclusion des features

Toutes les features n'ont pas eu le même impact sur nos prédictions. Certaines ont empiré nos prédictions, c'est le cas notamment pour la feature sur la répartition des mois dans les saisons, et d'autres n'ont pas eu de réel effet comme le taux d'humidité. Finalement, nous avons pu trouver une combinaison qui nous a permis d'avoir notre meilleure prédiction : température, pression atmosphérique, pluie du jour précédent et direction du vent (est ou ouest) :

```
MAPE score pour le fit : 31.223 ± 0.434
MAPE score pour le val : 31.210 ± 2.015
```

Le score Kaggle était alors de **30.2**.

ANNEXE 1

Cette annexe présente les listes des variables dans les différents fichiers de données utilisés.

Fichier X_station_train.csv :

- ff : vitesse du vent en $m.s^{-1}$
- t : température en Kelvin
- td : température du point de rosée ou de condensation : **dew point temperature**
- hu : humidité en pourcentage
- dd : direction du vent en degré
- precip : précipitations relevées durant la période en kg.m2 (~mm)
- Id : identifiant : n°station + jour + heure
- number_sta : n°station
- date : une par heure au format 'YYYY-MM-DD HH:mm:ss'

Fichiers netCDF : AROME et ARPEGE 2D

- ws : wind speed $m.s^{-1}$

- p3031 : direction du vent en degrés

- u10 : composante horizontale U de la vitesse du vent à 10m

- v10 : composante verticale V de la vitesse du vent à 10m

- t2m : température (K) à 2m

- d2m : dew point temperature (K) à 2m

- r : humidité relative (%)

- tp : précipitation totale ($kg.m^{-2}$) relevée depuis le début de l'exécution du modèle

- msl : pression moyenne au niveau de la mer (Pa)