Natacha GOUGEON
Gatien CHOPARD

APST1
Projet Data Challenge

Predicting odor compound concentrations

I. Introduction

La plateforme *Challenge Data*, gérée par la *Data team* de l'ENS de Paris, par le Collège de France et le *DataLab* de l'institut Louis Bachelier, propose chaque année des défis de machine learning. On se propose ici de participer à l'un des challenges soumis en 2022.

Le challenge choisi est intitulé *Predicting odor compound concentrations*. Il est proposé par l'entreprise Veolia, un leader global dans le management optimisé de ressources. Cette entreprise propose des solutions pour la gestion des services publics d'eau, d'assainissement et de traitement des déchets. En particulier, Veolia cherche des solutions pour réduire les nuisances autour des sites industriels, notamment les nuisances olfactives.

Ainsi, pour ce challenge, il s'agit de prédire les concentrations en dioxyde de soufre (SO2), un gaz toxique et irritant, à l'odeur pugnace, produit par de nombreux procédés industriels. Plus exactement, le but de ce projet est donc d'entraîner un modèle de régression pour prédire les concentrations en SO2 pour 12 heures sur une station en particulier, à partir des données géographiques, météorologiques et de concentration en SO2 sur les 48 heures précédentes d'un réseau de 6 stations alentour. L'enjeu est donc aussi de montrer que les prédictions sont cohérentes alors que les concentrations précédentes de la station test ne sont pas dans le jeu d'entraînement.

Pour participer à ce challenge, on a travaillé dans un premier temps sur des outils non partagés, c'est-à-dire des notebooks Jupyter et des scripts python sous Spyder, puis nous avons mis en commun notre travail dans un même notebook sous Colab, notebook final rendu avec ce rapport.

II. Jeu de données

Les données pour ce challenge sont présentées sous la forme de mesures réalisées heure par heure pendant 48 heures pour 6 stations différentes. On appellera donc 'feature' l'ensemble des 48 mesures effectuées pour une grandeur physique donnée. Par exemple, 'windSpeedKph' est une feature constituée des mesures 'windSpeedKph-i' pour $1 \leq i \leq 48$. La variable à prédire est la concentration en SO2 à la station MAS (en jaune sur la carte) pendant les 12 prochaines heures. Les features peuvent être distinguées en 3 catégories :

- Tout d'abord, les concentrations en SO2 mesurées sur les 6 stations qui se trouvent autour de la station MAS. Ne disposant pas de mesures de concentrations en SO2 pour la station MAS dans les données d'entraînement, ce sont les seules qui sont homogènes à la quantité à prédire. On s'attend donc à ce qu'elles aient un rôle majeur dans l'apprentissage.
- Nous disposons également de données météorologiques communes à toutes les stations, comme la température, la direction du vent ou encore sa vitesse. Comme expliqué dans la vidéo de présentation du sujet, si le lien entre la concentration en dioxyde de soufre à un point fixé et celles à proximité est évident, l'impact des données météo est plus difficile à mesurer. Un objectif du projet sera donc d'étudier la corrélation potentielle entre les données météos et la concentration en SO2, et dans quelle mesure elles permettent une meilleure prédiction par rapport aux seules mesures de concentration.
- Enfin, nous disposons d'informations relatives aux stations, à savoir leur position géographique et le type d'environnement où elles se trouvent (port, zone urbaine, forêt…).

Avant d'entraîner un modèle, nous pouvons effectuer des transformations sur les données afin de faciliter l'apprentissage. Les 'land cover class' en particulier, sont des chaînes de caractère décrivant, comme mentionné ci-dessus, l'environnement entourant la station. On en retrouve 4 différentes dans le dataset ('Continuous urban fabric', 'Discontinuous urban fabric', 'Green urban areas', 'Port areas'). Ce sont donc des classes qu'on peut représenter par un entier entre 1 et 4, ou bien en réalisant un one-hot encoding.

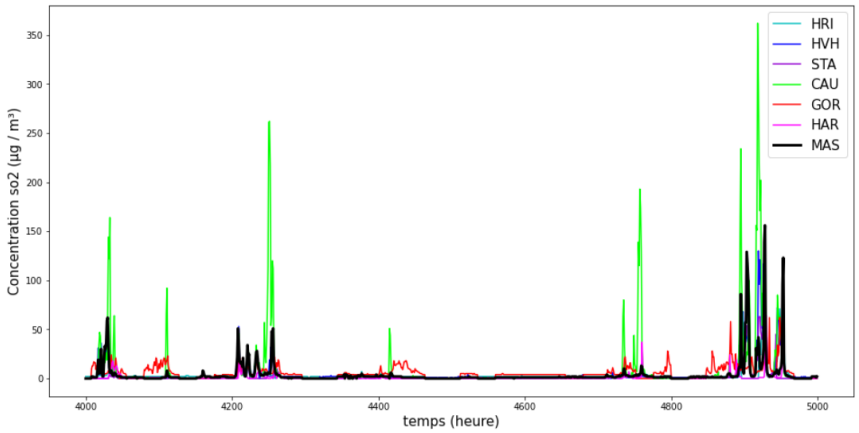
La direction du vent, exprimée en degrés, pourra également être transformée. Elle prend des valeurs de 0 à 360 degrés, ce qui peut créer un ordre qui n'a pas lieu d'être entre les différentes directions : Un vent avec un angle de 2° ne sera pas forcément traité de la même façon qu'un vent de 358° alors que leur direction est quasiment identique. De plus, le vent est le seul paramètre physique pouvant interagir avec l'information de la position géographique. Nous testerons dans la partie sur le deep learning une combinaison de ces différentes informations permettant de simplifier les données.

Avant de commencer la phase d'apprentissage, nous visualisons l'évolution des features afin de repérer d'éventuels motifs dans leur évolution ou une corrélation avec la quantité à prédire. Bien que les mesures du jeu de données soient réalisées sur 48h, elles ne sont pas indépendantes entre elles : en effet, les mesures ont été réalisées sur un an, et deux mesures successives ne sont en fait décalées que d'une heure.

SO2_GOR-16	SO2_GOR-15	SO2_GOR-14	SO2_GOR-13	SO2_GOR-12	SO2_GOR-11	SO2_GOR-10	SO2_GOR-9	SO2_GOR-8	SO2_GOR-7
6	14	15	13	12	6	5	14	15	41
14	15	13	12	6	5	14	15	41	1
15	13	12	6	5	14	15	41	1	1

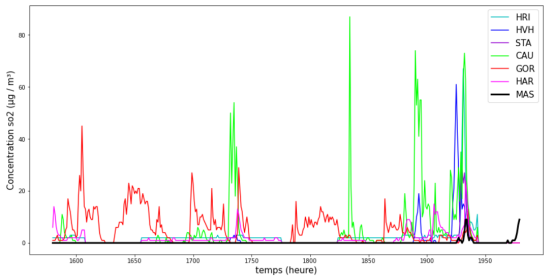
On peut donc reconstituer l'évolution de chaque feature sur une durée allant jusqu'à un an, et faire des observations sur les quantités étudiées, dont voici quelques exemples :

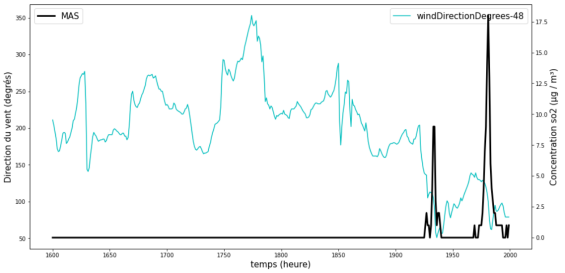
On s'intéresse d'abord aux concentrations en SO2, en particulier aux moments de leurs pics :



On remarque que les pics de SO2 à la station MAS sont généralement associés à des pics pour les autres stations, mais que ceux-ci ne sont pas nécessairement différés, ce qui pourrait s'avérer problématique si un pic survient pendant les douze heures à prédire; mais la corrélation ne fait aucun doute.

Il y a en revanche des pics mesurés sur les six stations d'entrée qui ne sont pas retrouvés sur la station MAS :





On constate d'autre part que sur cette même période, le vent souffle plutôt entre 180 et 360°, c'est-à-dire qu'il souffle globalement d'Ouest en Est. Les stations GOR (en rouge) et CAU (en vert) étant elle même situées à l'Est, on peut donc supposer que le vent a poussé les polluants dans la direction opposée à celle de la station MAS. On retrouve d'ailleurs un pic à la station MAS dès que le vent commence à tourner (autour de h=1930).

Ces visualisations nous ont permis de nous rendre compte de l'interaction entre les stations et de la corrélation avec d'autres paramètres physiques, notamment le vent.

III. Méthodes utilisées et résultats

On a mis en place différents modèles différents de machine learning, de méthodes basées sur des arbres à des réseaux de neurones. Les performances de ces modèles sont mesurées par calcul de l'erreur quadratique moyenne (MSE) contre le jeu de données de test. Pour donner un ordre de grandeur, une prédiction aléatoire donne un score MSE=89,9176. Un benchmark à également été établi pour ce challenge. Il s'agit d'un réseau LSTM mis en place avec une perte *'mse'*, l'optimiseur *Adam* et un taux d'apprentissage de 1e-3 depuis *Keras*. Les autres paramètres sont donnés dans la description du challenge. Ce modèle obtient un score MSE=46,3378. Un des objectifs est donc d'obtenir de meilleures performances que ce benchmark.

On peut noter de plus qu'un essai en régression linéaire à été effectué au départ, mais sans succès à cause de la qualité multi-sorties de nos données (il n'y a pas une variable cible mais 12, une variable par heure). Par la suite, on a trouvé comment ajuster des modèles à ce genre de sortie grâce au module *multioutput* de *scikit-learn* (cf partie Boosting), mais on a pas repris ces modèles assez simple de régression linéaire.

A. méthodes à base d'arbres

On a entraîné différents modèles de machine learning à base d'arbres. Ces méthodes ont l'avantage d'être plutôt facilement interprétable par leur forme d'arbre de décision.

Préparation du dataset

La première étape est le choix des features conservées dans le dataset d'entraînement.

On a éliminé les features qu'on ne trouvait pas physique, comme l'identifiant de l'échantillon ou le jour de la semaine. Pour les données météorologiques, on a conservé les plus simples : les températures à la surface, l'humidité, la pression atmosphérique à la surface, la vitesse du vent, les précipitations, l'irradiance normale, les tombées de neige, les rafales de vents. L'idée était de simplifier le dataset en éliminant les features trop corrélées (la température ressentie par rapport à la

température réelle et au vent par exemple). On a également éliminé les données géographiques, difficilement mises en valeur par la forme du dataset. En effet, une ligne renseigne toutes les valeurs pour les 6 stations, il est donc difficile d'isoler les valeurs d'intérêt d'une station à l'autre. Pour pouvoir les utiliser, il aurait fallu remanier le dataset en profondeur, et on ne voulait pas se plonger dans cette entreprise. Enfin, on a bien sûr conservé toutes les mesures de concentration en SO2. Ces choix diminuent le nombre de covariables de 2065 pour le jeu de données au départ, à 672.

On a également dû s'occuper des données manquantes. Sur les 6089 échantillons du dataset, il y avait 49 échantillons présentant des données manquantes. C'est très peu de perte d'information, on a donc décidé de simplement éliminer ces échantillons du dataset. Enfin, il n'y avait pas de données manquantes dans le dataset de test.

On a utilisé ce dataset pour l'ensemble des méthodes de cette partie.

Arbres de décision de régression

Le premier modèle testé est un simple arbre de décision pour la régression. On utilise la fonction *DecisionTreeRegressor* de *scikit-learn*. On peut remarquer que le multi-output est déjà implémenté dans cette méthode. En effet, chaque feuille peut stocker plusieurs valeurs de sorties, et le splitting se fait sur l'ensemble des variables cibles.

Pour ajuster le modèle, on a joué sur deux paramètres : *max_leaf_nodes* le nombre maximal de feuilles de l'arbre et *max_features* le nombre de features maximal à considérer pour diviser les données à chaque nœud de l'arbre. Pour optimiser ces paramètres, on a mis en place une recherche par grille avec la fonction *GridSearchCV* de *scikit-learn* (*Fig1*). On a exécuté plusieurs fois cette recherche. On obtient des résultats assez variés selon les exécutions. Cela montre notamment le caractère instable d'un modèle formé par un unique arbre de décision, comme on avait déjà pu le voir en comparant la forme des arbres (cf TP3).

	exécution i			
(max_leaf_nodes, max_features)	(4,200)	(10,20)	(4,672)	(4,10)

Figure 1 : table des résultats de GridSearchCV pour 4 exécutions.
La grille correspondante est max_leaf_nodes = [4,10,15,30,50,100] , max_features = [10,20,50,200,672]

On a finalement décidé de tester les prédictions sur le dataset de test pour deux arbres : (max_leaf_nodes=5, max_features=672) et (max_leaf_nodes=10, max_features=20). On peut donc comparer les performances entre un arbre présentant peu de feuilles et beaucoup de features, et un autre présentant peu de features et plus de feuilles (*Fig2*). On trouve une MSE plus faible, donc une meilleure performance pour le deuxième modèle. On peut penser que limiter le nombre de features permet de limiter le surapprentissage, et donne donc de meilleures performances au test.

modèle	MSE
Arbre décisionnel 1 (max_leaf_nodes = 5, max_features = 672)	62,1532
Arbre décisionnel 2 (max_leaf_nodes = 10, max_features = 20)	55,7373

Figure 2 : table des MSE sur le dataset de test

Pour autant, un unique arbre ne permet pas de dépasser le benchmark, et l'instabilité d'un tel modèle le rend incertain. On va mettre en place des méthodes plus avancées.

Bagging

Le deuxième modèle testé est un modèle de *bagging*. L'idée est d'agréger les prédictions de différents estimateurs de base, ici des arbres décisionnels de régression et chaque arbre est entraîné sur un subset aléatoire des échantillons du dataset originel, la prédiction finale est obtenue en moyennant les prédictions des arbres de l'ensemble. Plus précisément, on utilise la fonction *BaggingRegressor* de *scikit-learn* pour le bagging et encore la fonction *DecisionTreeRegressor* pour les estimateurs de base. Ce modèle supporte donc toujours le multi-output.

On a optimisé l'hyper-paramètre *n_estimators*, le nombre d'estimateurs dans l'ensemble, avec une recherche par grille avec *GridSearchCV* (*Fig3*). On a exécuté plusieurs fois cette recherche, sur deux estimateurs de base (*DecisionTreeRegressor* avec max_leaf_nodes=5, max_features=672 dit estimateur 1 et *DecisionTreeRegressor* avec max_leaf_nodes=10, max_features=20 dit estimateur 2). On obtient de nouveau des résultats assez variés selon les exécutions. On a finalement conservé n_estimators = 40, moyenne pour l'estimateur 1 et n_estimators = 70, moyenne pour l'estimateur 2.

	exécution i									
n_estimators pour estimateur 1	70	100	30	10	30	10	10	90	30	50
n_estimators pour estimateur 2	60	100	30	100	30	100	100	60	100	60

Figure 3 : *table des résultats de GridSearchCV pour 10 exécutions.*
La grille correspondante est n_estimators = [30,60,100]

On a finalement décidé de tester les prédictions sur le dataset de test pour deux modèles de bagging : un ensemble avec 40 estimateurs 1 (modèle bagging 1) et un ensemble avec 70 estimateurs 2 (modèle bagging 2). De nouveau, on trouve des meilleures performances pour l'ensemble contenant des arbres avec peu de features et plus de feuilles pour empêcher le surapprentissage (*Fig4*). C'est cohérent, le bagging ne fait que la moyenne des résultats des différents estimateurs de l'ensemble. On remarque que le MSE du modèle bagging 2 est meilleur que celui du benchmark, ce qui est déjà un résultat appréciable pour le challenge.

modèle	MSE
Modèle bagging 1 n_estimators=40, max_leaf_nodes=5, max_features=672	51,5690
Modèle bagging 2 n_estimators=70, max_leaf_nodes=10, max_features=20	45,8447

Figure 4 : table des MSE sur le dataset de test

Forêt aléatoire

Le troisième modèle testé est une forêt aléatoire. On peut le voir comme une forme de *bagging*, mais où les différents arbres sont décorrélés. C'est-à-dire que chaque arbre est entraîné sur un subset aléatoire des échantillons du dataset originel, et sur un subset aléatoire des features, ce qui permet la décorrélation. La prédiction finale est encore obtenue en moyennant les prédictions des arbres de la forêt. On a utilisé la fonction *RandomForestRegressor* de *scikit-learn*. Ce modèle supporte aussi le multi-output.

On a optimisé les hyper-paramètres *n_estimators* le nombre d'estimateurs, *max_leaf_nodes* le nombre maximal de feuilles de l'arbre et *max_features* le nombre de features maximal à considérer pour diviser les données à chaque nœud de l'arbre. La recherche par grille par *GridSearchCV* étant assez longue pour évaluer un triplet de paramètre, on a choisi une grille réduite (n_estimators = [10,50,100,150], max_features = [10,50,100,300,672], max_leaf_nodes = [10,50,100,150]) et on a pas répéter l'exécution. On obtient n_estimators = 100, max_features = 10, max_leaf_nodes = 50.

On a testé trois modèles de forêt aléatoire sur le dataset de test :

- le modèle obtenu par recherche par grille, avec n_estimators=100, max_features=10, max_leaf_nodes=50
- un modèle plus complexe, avec n_estimators=100, max_features=100, max_leaf_nodes=100
- un modèle plus simple, avec n_estimators=100, max_features=4, max_leaf_nodes=100.

Logiquement, les meilleures performances sont données par le modèle obtenu par optimisation des hyper-paramètres. et elles dépassent de nouveau celles du benchmark (*Fig5*). Pour le modèle complexe, Forêt aléatoire 2, les performances plus faibles peuvent être expliquées par le surapprentissage alors que pour le modèle plus simple, Forêt aléatoire 3, ce modèle n'est pas capable d'expliquer la complexité du dataset.

modèle	MSE
Forêt aléatoire 1 n_estimators=100, max_features=10, max_leaf_nodes=50	44,3741
Forêt aléatoire 2 n_estimators=100, max_features=100, max_leaf_nodes=100	46,6336
Forêt aléatoire 3 n_estimators=100, max_features=4, max_leaf_nodes=100	50,0693

Figure 5 : table des MSE sur le dataset de test

Gradient Boosting

Le principe du *Boosting* est d'entraîner à la suite plusieurs estimateurs, en compensant à chaque itération les faiblesses de l'estimateur précédent. Plus exactement, on met ici en place du *Gradient Boosting* : les erreurs sont minimisées par l'algorithme de descente de gradient. On entraîne un premier modèle sur les données. Pour l'entraînement du deuxième modèle, ensuite, un second modèle est construit pour tenter de corriger les erreurs présentes dans le premier modèle. C'est à dire que plutôt que de prédire y la variable cible, il va prédire $y - \hat{y}$ *avec* $\hat{y}$ la prédiction du premier modèle. Cela permet de mettre un poids plus important au échantillons n'ayant pas été bien prédit par le premier modèle. Cette procédure est répétée pour obtenir le nombre de modèle désiré. La prédiction finale est la somme des prédictions des différents arbres.

On a utilisé la méthode *XGBoost*. Mais celle-ci ne marche que pour une unique variable cible, or ici on a 12 variable cibles, les concentrations pour les 12 heures suivantes. On a donc aussi utilisé la méthode *MultiOutputRegressor*, du module *multioutput* de *scikit-learn*, qui permet d'entraîner un régresseur par variable cible. On peut remarquer que ces régresseurs ne sont donc pas corrélés, ce qui n'est pas optimal pour un dataset de série temporelle comme le nôtre.

On a cherché à optimiser les hyper-paramètres *n_estimators* le nombre d'estimateurs, *max_leaf_nodes* le nombre maximal de feuilles des arbres. On a pas répéter l'exécution de la recherche par grille via *GridSearchCV*, car une exécution était déjà longue. On obtient n_estimators = 10, max_leaves = 10 pour une grille n_estimators = [10,50,100,150], max_leaves = [10,50,100,150]. Ce résultat est surprenant, il correspond au modèle le plus simple proposé par la grille choisie.

On a testé ce modèle de boosting sur le dataset de test, on obtient un MSE de 49,0569. Les performances sont donc moins bonnes que celles du benchmark et celles des modèles de Bagging et de Random Forest qu'on a mis en place.

On a également testé un modèle de boosting pour différents paramètres n_estimators et max_leaves (*Fig6*). On remarque que les performances semblent dépendre du nombre d'estimateurs et pas du nombre de feuilles par arbre. On pourrait sans doute améliorer les performances en optimisant plus précisément le nombre d'estimateurs, car si il semble déjà y avoir du surapprentissage pour 50 estimateurs, on peut espérer trouver un modèle plus performant avec un nombre d'estimateurs entre 10 et 50.

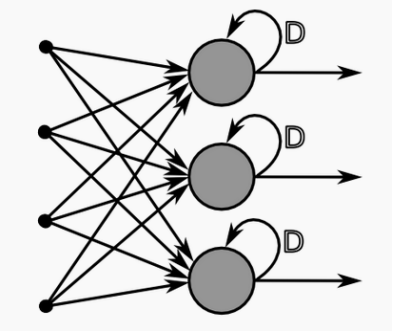
modèle	MSE
Boosting 1 n_estimators = 10, max_leaves = 10 (modèle optimisé par recherche par grille)	49,0569
Boosting 2 n_estimators = 10, max_leaves = 100	49,0569
Boosting 3 n_estimators = 50, max_leaves = 10	50,0707
Boosting 4 n_estimators = 50, max_leaves = 100	50,0707

Figure 6 : table des MSE sur le dataset de test

B. méthodes à base de réseaux de neurones

Nous utilisons dans cette partie des réseaux de neurones récurrents afin de prédire la concentration en dioxyde de soufre. Un réseau de neurone récurrent (ou RNN pour *recurrent neural network)*, tout comme un réseau de neurones 'classique', est constitué de couches avec un certain nombre de neurones, relié entre eux avec des poids, et s'entraîne en utilisant la rétropropagation du gradient.

La spécificité du RNN est qu'au lieu d'une simple propagation avant, certaines de ses couches comportent des cycles : La sortie d'un neurone sera ainsi utilisée en tant que donnée d'entrée de ce même neurone pour un nombre d'itérations défini.



source : wikipédia

Les RNN gardant en mémoire les événements passés dans leur apprentissage, ils sont tout à fait adaptés pour l'étude de séries temporelles. Ils se heurtent néanmoins au problème de disparition du gradient : pour des fonctions d'activation classiques, l'importance du gradient associé aux événements passés décroît exponentiellement, par conséquent le réseau aura de plus en plus de difficulté à tenir compte d'événements passés à mesure qu'ils s'éloignent de l'instant présent. Autrement dit, plus les informations s'accumulent, plus le RNN aura plus de mal à apprendre.

Les réseaux Long short-term memory (LSTM) sont un type de RNN adapté pour pallier ce problème. En plus des cycles récurrents, chaque neurone du LSTM comporte un triplet {forget gate, input gate, output gate}, dont le rôle consiste à filtrer l'information qui arrive dans le neurone et celle qui en ressort, et à en oublier une partie. les valeurs du triplet {forget gate, input gate, output gate} sont généralement des réels entre 0 et 1 indiquant la proportion d'information conservée ou oubliée.

Le réseau de neurone utilisé pour ce projet est constitué d'une couche LSTM ainsi que d'une couche dense en sortie. Nous avons commencé avec des paramètres proches de ceux du benchmark, puis les avons ajustés dans le but d'améliorer le score. Le premier réseau implémenté avec toutes les features et des paramètres proches de ceux du benchmark (epochs = 100, batch_size = 512, nb_LSTM_layers = 1, nb_units = 30, dropout_rate = 0.2) donne une MSE de 67.4.

Voici les 3 axes d'amélioration explorés afin d'améliorer les performances du réseau :

Feature Selection/Transformation

Après avoir analysé le jeu de données, il apparaît que bon nombre de features peuvent être supprimées. D'une part, il est probable que certaines données météos n'aient pas d'impact sur la concentration à prédire, ou du moins un impact négligeable par rapport à d'autres features. D'autre part, certaines informations sont redondantes dans les mesure où il existe une corrélation entre les features. C'est le cas par exemple des coordonnées géographiques, que le réseau peut déduire de l'interaction des différentes stations entre elles.

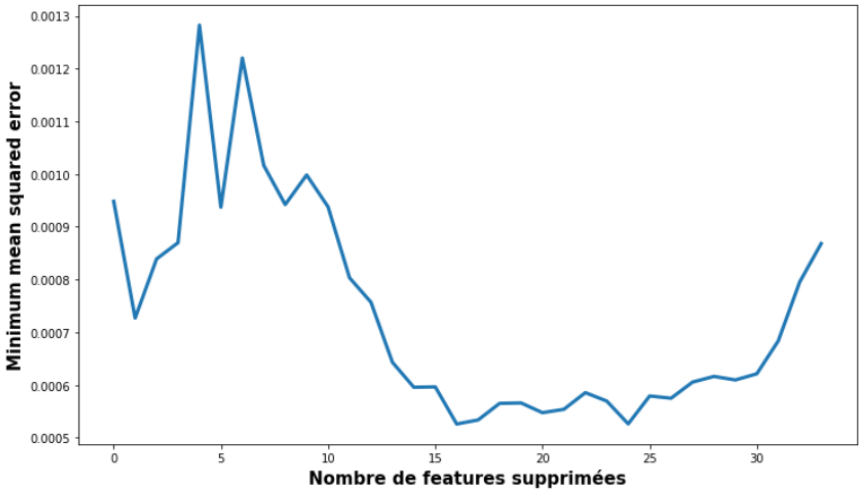
La première approche a été d'entraîner le réseau avec un nombre réduit de features. En particulier, entraîner le réseau avec les seules concentrations en SO2 dans les stations donne de très bons résultats, ce qui confirme l'hypothèse faite en introduction.

Pour aller plus loin, nous avons codé un algorithme de recursive feature selection (RSE) visant à trier les features par ordre d'importance. Son fonctionnement est le suivant :

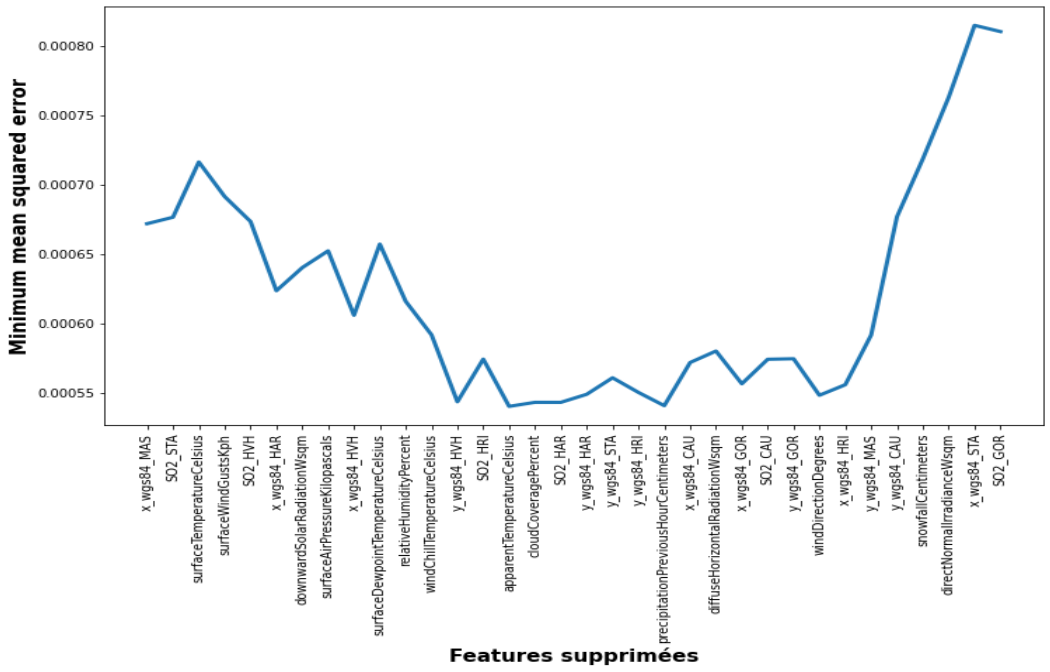
- On initialise le réseau avec toutes les features (n features)

- On entraîne le réseau en enlevant 1 feature (le réseau utilise donc n-1 features) et on calcule sa MSE. L'opération est répétée en retirant à chaque fois une feature différente.
- On compare les n MSE obtenues et on retire définitivement la feature pour laquelle l'erreur obtenue est la plus faible.
- On répète les deux étapes précédentes jusqu'à ce qu'il n'y ait plus de features.

Dans notre cas, le nombre de features est de 35, en retirant avant d'exécuter l'algorithme les Land Cover Class, les jours de la semaine et les heures. L'algorithme MSE entraîne 35+34+....+2= 629 réseaux lors de son exécution. Chaque réseau pouvant prendre 1 minute pour s'entraîner, on devine facilement que cet algorithme est très coûteux en temps de calcul et en ressources. Après l'avoir exécuté deux fois, les résultats sont les suivants :



Lors de la première exécution, les features ne sont pas retirées par ordre croissant d'importance mais selon un ordre arbitraire. L'erreur est minimale pour un nombre de features compris entre 10 et 20 (15 à 25 features retirées).



Au cours de la 2ème exécution, les features les moins importantes sont supprimées en premier conformément à l'algorithme décrit précédemment. Le graphique ci-dessus montre que l'ordre obtenu est difficile à interpréter, à l'image des concentrations en SO2 qui sont à la fois parmi les premières et les dernières features retirées.
Une explication de ces résultats pourrait être qu'un seul réseau a été entraîné à chaque itération pour une feature donnée : l'écart non significatif entre les MSE associé à la variance liée à l'entraînement font que les features n'ont pas forcément été retirées par ordre d'importance.
Une piste d'amélioration serait donc d'entraîner plusieurs réseaux par feature testée, puis de moyenner la MSE, au prix d'un temps de calcul plus important.
Cette deuxième exécution confirme cependant l'observation faite plus haut, à savoir qu'il existe des nombres de features optimaux pour lesquels l'erreur est moindre.

Enfin, nous avons testé une transformation des données visant à traduire l'importance du vent sur le déplacement du SO2, à partir de la direction du vent et des coordonnées géographiques. Cette méthode n'a pas donné de résultats concluants, mais nous la détaillerons dans le notebook joint à ce rapport.

Modèle	MSE
LSTM, features SO2 uniquement	46.37
LSTM, features SO2 + données météos	50.83

Figure 7 : table des MSE sur le dataset de test

Optimisation des hyperparamètres

Une autre manière d'optimiser le réseau de neurones est de modifier ses hyperparamètres. Les hyperparamètres ici correspondent au batch size, au nombre d'epochs et à l'optimiseur utilisé; ce sont tous les paramètres qui définissent le fonctionnement du réseau. Nous utilisons pour cela la fonction GridSearch de scikit-learn, qui, à partir des valeurs que l'on souhaite tester pour chaque paramètre, teste toutes les combinaisons possibles puis les compare. Les résultats obtenus ne sont pas déterminants, puisque l'erreur est sensiblement la même pour tous les réseaux testés :

```
Best: -34.864988 using {'batch_size': 50, 'optimizer': 'rmsprop'}
-34.864988 (46.194897) with: {'batch_size': 50, 'optimizer': 'rmsprop'}
-34.937470 (46.166502) with: {'batch_size': 50, 'optimizer': 'adam'}
-34.961770 (46.184431) with: {'batch_size': 50, 'optimizer': 'SGD'}
-34.993158 (46.310963) with: {'batch_size': 100, 'optimizer': 'rmsprop'}
-35.117086 (46.399681) with: {'batch_size': 100, 'optimizer': 'adam'}
-34.907690 (46.281251) with: {'batch_size': 100, 'optimizer': 'SGD'}
-35.090605 (46.532003) with: {'batch_size': 150, 'optimizer': 'rmsprop'}
-34.926107 (46.628077) with: {'batch_size': 150, 'optimizer': 'adam'}
-35.021455 (46.660131) with: {'batch_size': 150, 'optimizer': 'SGD'}
```

Après de plus amples recherches, nous ajustons les hyperparamètres de la manière suivante :

- Le batch size est considérablement réduit par rapport au benchmark et aux premiers essais, passant de 512 à 64.

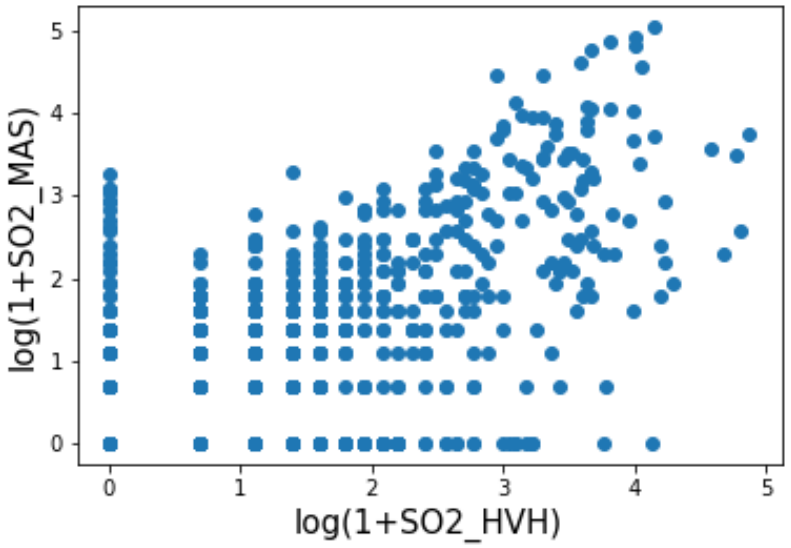
- On donne aux paramètres des valeurs correspondant à des puissances de 2 : 64 pour le batch size, et 32 units dans la couche LSTM. Bien que cela n'améliore pas drastiquement les performances, il s'agit d'une bonne pratique ayant un léger impact sur le temps de calcul.
- Ajout de l'Early stopping au réseau. L'Early stopping permet de comparer l'erreur à chaque epoch et de retenir quand celle-ci ne s'améliore plus. Après un certain nombre d'epochs sans améliorations, l'entraînement s'arrête, réduisant ainsi le temps d'exécution.
- Les kernels regularizer l1, l1l2 et l2 ont tous trois été testés, mais n'apportent pas de réduction de la MSE.

Modèle	MSE
LSTM, 32 units, features SO2, batch size =1024	48.20
LSTM, 32 units, features SO2, batch size =64	46.37

Figure 8 : table des MSE sur le dataset de test

Méthode Hybride

Une autre observation qui ressort des visualisations décrites en introduction est le comportement de la concentration que l'on souhaite prédire : celle-ci atteint des valeurs élevées par pics, mais est presque nulle le reste du temps.



Ce graphique met en lumière que pour des valeurs faibles de la concentration à prédire (en ordonnées), la concentration dans les autres stations prend un large éventail de valeurs et ne semble pas corrélée. Pour des valeurs de concentration plus importantes en revanche, une relation linéaire se dégage (entre les log des concentrations). Ce constat nous incite à nous concentrer sur les pics dans l'apprentissage du réseau de neurones. La prédiction des pics est essentielle car c'est de là que provient la plus grosse partie de l'erreur MSE, du fait des grandes valeurs de concentration élevées au carré. D'un point de vue pratique, les pics sont importants car ce sont eux qui représentent une nuisance voire un danger potentiel pour les habitants, et qui par conséquent motivent cette étude.

La première composante de la méthode hybride consiste donc à séparer les données en deux classes grâce à une régression logistique. D'une part, les instants où la concentration est nulle, et d'autre part, ceux où il y a un pic. Pour la définir brièvement, la régression logistique consiste à déduire une variable binaire à partir de nombreuses observations. Elle s'apparente à un modèle linéaire dans son expression, à la différence que la variable Y est un logarithme, d'où son nom.

Une fois les deux classes séparées, on entraîne un réseau de neurones LSTM (ou n'importe quelle autre méthode présentée dans ce rapport) sur les données où il y a des pics.
La prédiction est ainsi réalisée en deux étapes : On sépare d'abord les données selon les 2 classes 'pic' et 'pas pic'. S'il n'y a pas de pic, on assigne la concentration à 0. Sinon, on prédit avec le modèle LSTM (ou autre) entraîné spécialement sur les pics. La méthode offre finalement de bons résultats avec le réseau LSTM. Avec une forêt aléatoire, il n'y a pas de réelle amélioration par rapport à une forêt aléatoire seule :

Modèle	MSE
LSTM, batch size = 64, units = 32, features SO2	46.37
Régression logistique + LSTM, features SO2	46.16
Régression logistique + forêt aléatoire n_estimators=100, max_features=10, max_leaf_nodes=50 features SO2 + météo	44.42

Figure 9 : table des MSE sur le dataset de test

IV. Conclusion

A l'issue de cette étude, plusieurs modèles ont donné des performances supérieures à celle du benchmark : un modèle de bagging, un modèle de random forest et un modèle hybride associant une régression logistique à un réseau LSTM. Les meilleurs résultats sont obtenus avec une random forest, pour une MSE de 44,3741. Étonnement, le meilleur modèle est donc un modèle qui ne prend pas en compte le caractère séquentiel des données.

Toutefois, plusieurs pistes d'amélioration sont possibles. On pourra notamment traiter différemment les données afin de tirer un meilleur parti de leur caractère séquentiel et géographique. La prédiction des concentrations en polluants est un sujet largement abordé dans la littérature scientifique. Plusieurs études ont montré l'efficacité des réseaux de neurones, mais également de modèles adaptés aux séries temporelles comme les ARMA ou ARIMA. Explorer ces modèles serait une continuation possible de ce projet à condition de traiter la non-stationnarité des données.

L'objectif sous-jacent de toutes ces méthodes est de parvenir à caractériser les pics de SO2 qui constituent le cœur du problème, tant au niveau de l'erreur que des nuisances. Le Havre, ville dont proviennent les données, a vu sa centrale à charbon fermer définitivement l'année dernière. Étant une source majeure d'émissions de dioxyde de soufre, on peut donc espérer une réduction du nombre et de l'intensité des pics dans les années à venir.