RAPPORT DATA CHALLENGE

Frikha Myriam

November 13, 2023

1 Nettoyage des données

On commence l'étude par l'observation d'une grille de graphiques qui établit des relations entre chaque paire des éléments du data frame (Voir *Notebook pt1 - Data cleaning + models comparison*- I).

La plupart des variables présentent des distributions et des graphiques de dispersions assez cohérentes; A priori, il faudra les exploiter avec un modèle. Cependant, les colonnes *way* et *composition* montrent une forte corrélation avec les autres variables. On remarque de plus que la colonne *composition* présente une valeur qui n'existe pas dans le test set: $composition = 1$. On commence par enlever la colonne *way* : en effet, cette dernière ne présente aucune information supplémentaire par rapport au data set. De plus, on enlève les lignes où $composition = 1$. Ces valeurs pourraient perturber l'étape de prédiction et décroitre la précision du modèle.

Enfin, on repère les données manquantes : Il y a 7 colonnes qui contiennent des données manquantes à proportions différentes : p0q3, p0q0, p3q0, p2q0, hour, p0q1 et p1q0. On assigne des flags à ces dernières pour signaler au modèle leur nature.

2 Choix du modèle

2.1 Modèles de Scikit-Learn

J'effectue un premier test de comparaison entre les modèles de Scikit-Learn (Voir *Notebook pt1 - Data cleaning + models comparison* - II) en les évaluant selon l'erreur en valeur absolue (mae). Les deux meilleurs modèles sont : Random Forest et XG-boost. De plus, j'effectue un deuxième test en mélangeant les deux meilleurs modèles. J'effectue un premier mélange en faisant une moyenne des prédictions et un deuxième avec stacking : Le mélange avec stacking semble prometteur. On peut donc énumérer plusieurs stratégies :

- Stratégie 1 : Optimiser les hyperparamètres d'un des deux meilleurs modèles.
- Stratégie 2 : Optimiser les hyperparamètres d'un modèle mélange.

Je choisis d'optimiser les hyperparamètres du XG-boost dans un premier temps. Je le préfère au modèle de Random Forest car il propose plus de paramètres à optimiser. Le modèle est assez robuste et propose une perte assez faible.

J'ai testé ensuite plusieurs modèles de mélange : Mélange de XG-boost et Forests avec stacking (avec plusieurs tests de meta learners) ainsi qu'un mélange de XG-boost et Forests avec une moyenne. Mes tests n'ont pas apporté une différence significative par rapport à mon modèle de base. En effet, vu le nombre conséquent de paramètres à optimiser, ils sont souvent susceptibles à du overfitting.

2.2 Autres modèles

L'idée d'implémenter d'autres modèles spécifiquement conçus pour les séries temporelles me semblait judicieuse. En particulier, j'ai essayé de faire une première phase d'optimisation des hyperparamètres du modèle *LSTM* de la librairie Torch (voir *etape explorative LSTM*). La particularité de celui-ci est qu'il exige un reordonnement des données selon plusieurs familles : échantillons, pas de temps et caractéristiques. J'ai donc fait appel à des tenseurs

qui trient les données selon les jours. Les résultats d'optimisation ne sont pas concluants, donc j'abandonne également cette approche.

Bilan : Le modèle XG-boost optimisé semble être le plus performant et robuste (si on exclut le modèle du leak).

3 Modèle final

3.1 XG-boost

Le modèle est évalué sur l'ensemble de validation en utilisant l'erreur absolue moyenne (MAE) comme métrique sur ce problème de régression. J'optimise les hyper-paramètres grâce à Optuna. J'effectue plusieurs tests préalables en choisissant des intervalles de recherche assez larges pour ces derniers. Ensuite je les affine petit à petit en fonction des résultats obtenus (voir *pt2 - meilleur notebook*). Pour une prédiction plus robuste, j'utilise une validation croisée. La manipulation directe des intervalles des hyperparamètres dans la fonction objective, ainsi que les outils de visualisation d'Optuna permettent une meilleure compréhension de leur impact sur la prédiction. Par exemple, on voit que le choix de la profondeur maximale de chaque arbre de décision ainsi que le nombre d'arbres a plus d'impact que les autres paramètres étudiés. Il faut donc accorder une attention particulière à l'optimisation de ces deux derniers.

3.2 Leak - classement académique public : 6/26 , classement académique privé : 6/26

Il s'agit cette fois d'identifier des patterns qui permettent d'attribuer les bonnes valeurs de prédictions (voir *pt3 - meilleur notebook*). Je propose un code assez simple mais qui performe assez bien sur le benchmark. Tout d'abord on sépare les fausses données des vraies; il s'agit de celles ayant quatre décimales après la virgule. Ensuite on établit, grâce aux observations, des fonctions qui exploitent le leak. Enfin, on remplace les fausses données par les prédictions du XG-boost effectuées précédemment.

4 Obstacles

La difficulté majeure que j'ai rencontrée consistait à réduire le temps d'éxécution des modèles. C'est une problématique d'autant plus importante quand on n'a pas accès à un GPU. Voici les solutions que j'ai trouvées :

- Remplacer l'optimisation avec Gridsearch par Optuna (2239488 fits à faire pour optimiser 12 paramètres avec Gridsearch!)

- Implémenter des techniques de parallélisme (avec la commande n jobs=-1) afin d'utiliser tous les cœurs de processeur disponibles sur la machine

- Introduire un Pruner spécifique à XG-boost *XGBoostPruningCallback* qui applique l'élagage pendant l'entraînement du modèle XGBoost.

La complexité de certains modèles est aussi une autre difficulté; Les mélanges de modèles avec stacking nécessitent une optimisation simultanée de 3 modèles différents (les deux modèles de prédiction et le meta learner), ce qui augmente la complexité du code et rend son

amélioration particulièrement difficile. La manipulation des tenseurs est également sensible pour le modèle LSTM, notamment lors de l'extraction des données finales de prédictions.

Finalement, opter pour un modèle de prédiction simple semble être le meilleur compromis entre compléxité et robustesse.