

APPLICATIONS DE L'ALGORITHME GBM ET TUNING

Les librairies R utilisées dans le cadre de ce TP sont les suivantes :

gbm, *xgboost* (ou *mboost*), *rsample*, *ggplot2*, *pdp*, *devtools*, *vip*, *caret*.

L'objectif du TP est d'implémenter l'algo GBM sur un cas concret, et plus précisément les Gradient Tree Boosting. Nous essaierons de comprendre comment manipuler les principaux paramètres (de tuning) des fonctions R utilisées, afin d'éviter le sur-apprentissage.

Considérons les données `cu.summary`, contenu dans la librairie `rpart` du logiciel R. Nous allons commencer par travailler avec la librairie `gbm`, pour expliquer le prix des voitures en fonction des autres variables.

Sur ce jeu de données :

- (1) Charger la librairie `gbm`, charger les données. Grâce aux fonctions `initial_split`, `training` et `testing`, isoler des données d'apprentissage représentant 70% de l'échantillon initial.
- (2) Consulter l'aide de la fonction `gbm`. Quels sont ses arguments et leur signification ?
- (3) Estimer un premier modèle GBM avec les paramètres suivants : 10000 itérations, composé d'arbres feuilles, de taux d'apprentissage égal à 0.001, et 5 champs pour la validation croisée.
- (4) Accéder aux résultats de ce modèle. Quels en sont les attributs ? Consulter et comprendre les. Quelle est la RMSE du modèle ?
- (5) Accéder à la performance par la fonction `gbm.perf`. Que remarquez-vous ?
- (6) On se propose maintenant d'optimiser le modèle en faisant varier ses paramètres de tuning afin de déterminer un bon jeu de paramètres. A ce titre, créer une grille de paramètres par la fonction `expand.grid` : on propose de jouer sur
 - le taux d'apprentissage,
 - la profondeur des arbres,
 - le nombre d'observations minimal par feuille,
 - le rééchantillonnage.Créer la grille correspondante, en évitant qu'elle ne contienne trop de cas pour ne pas que la suite prenne trop de temps (50 cas environ suffisent).
- (7) Pour gagner du temps, on abandonne la validation croisée et on crée seulement un échantillon test pour le calibrage et la mesure d'erreur du modèle (cf argument `train.fraction`, par exemple 75%). Estimer chacun des modèles associés aux jeux de paramètres de votre grille.
- (8) Accéder aux résultats et interpréter. Dans quelle(s) zone(s) semblent se trouver les paramètres optimaux ? Affiner maintenant votre grille en fonction de ces premiers résultats. Relancer l'estimation des modèles correspondants. Afficher les 10 meilleurs modèles obtenus : avez-vous amélioré vos résultats ?
- (9) Estimer maintenant un GBM final en fixant vos paramètres de tuning égaux au jeu de paramètre optimal déterminé précédemment, en réintroduisant la validation croisée. Quelle est la performance de ce modèle ?

- (10) Visualisation des résultats et de l'importance des variables :
 - avec la librairie `pdp` et la fonction `summary`, déterminer la variable explicative la plus importante. A quoi est liée cette mesure d'importance. Y'a t'il d'autres mesures accessibles ?
 - idem avec la librairie `vip` et la fonction `vip`.
 - Avec la fonction `partial` de la librairie `pdp`, faire un graphique "partial dependence plot" qui illustre l'importance de la variable explicative `Mileage`.
 - Remarque : la librairie `lime` peut également aider à l'interprétation du modèle GBM et l'impact des variables explicatives.
- (11) Effectuer des prévisions du modèle GBM construit sur votre échantillon test isolé au départ. En termes de RMSE, quelle est la qualité de votre modèle ?

Exercice 2 : avec la librairie `xgboost`

la librairie `xgboost` est certainement la plus connue des librairies pour l'implémentation pratique des GBM. En effet, elle est optimisée pour le calcul parallèle et va donc plus vite que les autres librairies la plupart du temps. Elle dispose également de fonctionnalités d'accélération bien pratiques !

- (1) Mettez en forme les données tel que la librairie `xgboost` le requiert.
- (2) Accéder à l'aide des fonctions `xgboost` et `xgb.train` pour en comprendre les arguments et les sorties. Quels sont les paramètres de tuning disponibles ici ?
- (3) A l'aide de la fonction `xgb.cv`, estimer un premier modèle GBM avec les caractéristiques suivantes :
 - 1000 itérations,
 - 5 champs de validations croisées,
 - les autres arguments avec leur valeur par défaut.
- (4) Lister les attributs de l'objet retourné et comprenez les. Accéder au résumé de la qualité du modèle via l'attribut `evaluation-log` : a priori combien d'itérations devraient suffire ? Faire le graphique correspondant avec `ggplot`.
- (5) On veut accélérer le calibrage du modèle. Quel argument de la fonction `xgb.cv` utiliser ? Fixer le à 10. Faire un graphique des erreurs MSE en fonction du nombre d'arbres, que ce soit pour l'apprentissage ou le test. Que remarquez-vous ?
- (6) On vous demande maintenant de tuner le modèle. Procédez comme dans la partie précédente en créant une grille de paramètres. Quels sont les paramètres de tuning pertinents à utiliser ici (relativement à la partie précédente) ?
- (7) Estimer l'ensemble des modèles correspondants et afficher les 5 meilleurs modèles obtenus. Quelles sont les zones pertinentes pour ces paramètres. Affiner votre grille afin d'optimiser encore mieux le modèle.
- (8) Estimer maintenant un modèle final avec la fonction `xgboost`, tenant compte des résultats obtenus lors de la recherche des paramètres de tuning optimaux. Quelle est la performance de ce modèle ? Comparer à celui de la partie précédente.
- (9) Visualisation des résultats et de l'importance des variables :
 - A l'aide de la fonction `xgb.plot.importance`, accéder à l'importance des variables explicatives. Quelles sont les différentes mesures d'importance disponibles ? Que représentent-elles ?
 - De la même manière que dans la partie précédente, utiliser la librairie `pdp` pour faire un graphique "partial dependence plot" qui illustre l'importance de la variable explicative `Mileage`.
- (10) Effectuer des prévisions du modèle GBM construit sur votre échantillon test isolé au départ. En termes de RMSE, quelle est la qualité de votre modèle (utiliser la librairie `caret` et sa fonction `RMSE`) ?