APPRENTISSAGE STATISTIQUE - MS ENSAI
TP3 : Boosting, Bagging, Stacking. 2025-2026

Objectifs du TP

Nous allons mettre en application les algorithmes relatifs aux méthodes d'agrégation et de convexité (en classification et régression), et les comparer aux autres méthodes d'apprentissage supervisé (k-ppv, arbres de décision, régression logistique, machines à vecteurs de support et réseaux de neurones).

Comme dans les TP précédents, pour chaque jeu de données réelles, il s'agit d'effectuer rapidement une première exploration descriptive des données et une analyse statistique simple des variables d'intérêt. Il faut ensuite utiliser les classifieurs et régresseurs Boosting, Bagging et Stacking, tester les différentes fonctions et leurs options, puis comparer les performances avec toutes les méthodes vues précédemment.

N'hésiter pas à aller chercher de la documentation en ligne et à utiliser Chat-GPT ou autre IA pour générer ou corriger des parties de votre code (en tâchant de comprendre ce qu'ils font !).

1 Comparaison d'un arbre seul avec Adaboost en régression

1°. Construire un échantillon d'une centaine de points en deux dimensions à partir d'une fonction sinusoïde perturbée par un bruit aléatoire.

2°. Utiliser les fonctions `DecisionTreeRegressor` et `AdaboostRegressor` pour ajuster deux modèles et les comparer sur un même graphique (on pourra prendre initialement 200 arbres avec une profondeur de 4).

3°. Étudier les paramètres de `AdaboostRegressor` (type d'estimateur, nombre, taux d'apprentissage, perte) et les attributs (en particulier l'importance des variables) de cette fonction. Les faire varier et observer graphiquement l'effet.

2 Comparaison d'Adaboost et Gradient Boosting en classification

4°. À l'aide de la fonction `make_hastie_10_2`, créer un échantillon de taille 1000 et le séparer en deux sous-échantillons d'apprentissage et de test (utiliser `train_test_split`). À l'aide des fonctions `GradientBoostingClassifier`, `AdaBoostClassifier` et `DecisionTreeClassifier`, ajuster un classifieur selon l'algorithme Adaboost (dont les classifieurs faibles seront des arbres de décision) et un autre selon l'algorithme Gradient Boosting, avec initialement $n = 200$ arbres d'une profondeur maximale de 3. Comparer leurs performances.

5°. reprendre la question précédente en faisant varier le taux d'apprentissage de 0.1 à 1.9. Représenter graphiquement les performances des deux classifieurs en fonction du taux d'apprentissage. Représenter ensuite les performances de chaque algorithme sur son échantillon d'apprentissage et son échantillon de test. Détecter l'apparition du phénomène de sur-apprentissage.

6°. En jouant sur les paramètres des arbres, essayer de corriger ce sur-apprentissage.

3 Gradient Boosting en régression

7°. Construire deux échantillons (un pour l'apprentissage et un pour le test) d'une centaine de points en deux dimensions à partir d'une fonction sinusoïde perturbée par un bruit aléatoire.

8°. Utiliser la fonction `DecisionTreeRegressor` pour ajuster un modèle d'arbre de décision et tracer sur un même graphique les données, ainsi que plusieurs arbres correspondants à des profondeurs différentes.

9°. À l'aide de la fonction `GradientBoostRegressor`, ajuster un modèle de Boosting contenant 1000 arbres et de profondeur maximale 1. Tracer la fonction de régression sur le même graphique que précédemment.

10°. Étudier les paramètres de `GradientBoostRegressor` et les faire varier pour en étudier l'effet.

11°. Écrire une fonction qui calcule l'erreur de régression et afficher cette quantité en fonction du nombre d'arbres, à la fois pour les données d'entraînement et de test.

4 Comparaison de tous les algorithmes de Boosting en classification

12°. Étudier les fonctions Python permettant d'effectuer du Boosting (Gradient Boosting, XGBoost, AdaBoost, CatBoost et LightGBM) et leurs paramètres respectifs.

13°. À l'aide de la fonction `make_classification`, créer un échantillon de taille 1000 avec des variables de dimension 20 (`n_features`). Séparer l'échantillon en deux en conservant 30% pour la partie test.

14°. Créer une liste des 5 classifieurs et entraîner chacun d'eux sur l'échantillon. Évaluer leurs performances respectives en termes d'AUC et tracer sur un même graphique toutes les courbes ROC correspondantes. Comparer également les temps

d'entraînement de chaque algorithme dans un histogramme.

15°. En vous aidant de la littérature trouvée sur internet et des simulations précédentes, construire un tableau de comparaison des 5 algorithmes.

5 California Housing Dataset

Le jeu de données « California Housing Dataset » est une base de données très courante utilisée comme benchmark en apprentissage statistique. Il contient des données issues d'un ancien recensement effectué en Californie, relatives au prix des logements en fonction de la position géographique, du nombre de pièces, de l'âge de la construction, de la population du district où le logement est construit, etc. La variable cible est le prix médian du logement. Ce jeu de données est inclus dans la librairie `sklearn.datasets`. Nous avons déjà utilisé ce jeu de données lors des TPs précédents ; ici, il s'agit de prévoir le prix médian à l'aide des algorithmes de Boosting et de Bagging pour un problème de régression.

16°. À l'aide de `fetch_california_housing`, importer ce jeu de données et effectuer des statistiques descriptives permettant de visualiser la base de données et d'étudier ses caractéristiques.

17°. Séparer les données en un échantillon d'apprentissage et un échantillon de test. Afin de pouvoir se comparer avec les résultats donnés dans l'ouvrage "the Elements of Statistical Learning" de Hastie et ses collaborateurs, on choisira une proportion de 0.8 pour la taille de l'échantillon d'apprentissage et on mesurera l'erreur par la métrique de MAE (Mean Absolute Error).

18°. À l'aide de la fonction `GradientBoostingRegressor`, ajuster un modèle de Boosting aux données et mesurer ses performances (dans un premier temps, vous pourrez sélectionner 300 arbres de profondeur maximale 2, avec un taux d'apprentissage de 0.04 et une fonction perte de Huber).

19°. Afficher sous la forme d'un histogramme l'importance relative des variables (attribut `feature_importances` de l'estimateur).

20°. Il vaut mieux travailler avec le logarithme du prix médian. Une fois cette transformation effectuée, normaliser également les données avec `StandardScaler`. Utiliser l'instruction `RFE` pour effectuer une sélection des variables par la méthode des modèles imbriqués. Conserver les 5 variables les plus significatives et utiliser la fonction `GridSearchCV` pour déterminer les meilleurs paramètres (nombre de classifieurs faibles, taux d'apprentissage et profondeur maximale) du modèle de Gradient Boosting.

21°. Ajuster le modèle correspondant aux meilleurs paramètres avec l'échantillon et mesurer ses performances.

22°. Étudier les fonctions `RandomForestRegressor`, `BaggingRegressor`, `XGBRgressor` et `LGBMRegressor` en prêtant attention aux paramètres importants.

23°. Ajuster chacun des modèles à l'aide d'une validation croisée et déterminer les paramètres qui optimisent la prédiction sur les données de test.

6 Stacking

24°. Étudier les fonctions `StackingClassifier` et `VotingClassifier` de `scikit-learn`. Quelles sont leurs différences ?

25°. Charger le jeu de données des iris de Fisher.

26°. Initialiser 3 classifieurs (k-ppv, forêt aléatoire, classifieur bayésien naïf) et les intégrer dans un méta-modèle à l'aide de la fonction `StackingClassifier`.

27° Ajuster ces modèles à l'échantillon (les 3 classifieurs seuls ainsi que le méta-modèle), calculer leurs performances et visualiser les régions de décision.