PARTIE 1: FONDEMENTS THÉORIQUES DES MODÈLES

Decision Tree (Arbre de Décision)

Principe :
— Divise récursivement l'espace des features en régions rectangulaires
— À chaque nœud, choisit la meilleure variable et seuil pour minimiser l'erreur
— Prédiction = moyenne des valeurs dans chaque feuille
Avantages :
— Interprétable et visualisable
— Capture les relations non-linéaires
— Pas besoin de normalisation
— Très rapide
Inconvénients :
— Tendance au sur-apprentissage
— Instable (petits changements = grands impacts)
— Moins performant que les ensembles
Hyperparamètres clés :
— max_depth : Profondeur maximale (contrôle la complexité)
— min_samples_split : Minimum d'échantillons pour diviser un nœud
— min_samples_leaf : Minimum d'échantillons par feuille

Random Forest (Forêt Aléatoire)

Principe :
— Ensemble de nombreux arbres de décision
— Chaque arbre entraîné sur un échantillon aléatoire (bootstrap)
— Prédiction = moyenne des prédictions de tous les arbres
— "Wisdom of the crowd" réduit la variance
Avantages :
— Très robuste au sur-apprentissage
— Gère bien les données bruitées
— Fournit l'importance des features
— Parallélisable (rapide avec multi-threading)
Inconvénients :
— Moins interprétable qu'un seul arbre
— Plus lent que les arbres simples
— Consomme plus de mémoire
Hyperparamètres clés :
— n_estimators : Nombre d'arbres
— max_depth : Profondeur des arbres
— max_features : Nombre de features considérées par split
— min_samples_leaf : Régularisation

Gradient Boosting

Principe :
— Construit les arbres séquentiellement
— Chaque nouvel arbre corrige les erreurs du précédent
— Minimise une fonction de perte par descente de gradient
— Combine des “weak learners” en un “strong learner”
Avantages :
— Souvent le plus performant sur des données tabulaires
— Capture des relations complexes
— Moins sensible aux outliers que Random Forest
— Flexibilité dans la fonction de perte
Inconvénients :
— Sensible au sur-apprentissage si mal paramétré
— Plus lent (séquentiel, non parallélisable)
— Nécessite un tuning minutieux
Hyperparamètres clés :
— n_estimators : Nombre d’arbres
— learning_rate : Taux d’apprentissage (plus petit = plus robuste)
— max_depth : Profondeur
— subsample : Fraction d’échantillons par arbre

XGBoost (eXtreme Gradient Boosting)

Principe :
— Version optimisée et régularisée du Gradient Boosting
— Techniques avancées : régularisation, pruning, parallélisation
— Algorithme de splitting plus efficace
— Gestion native des valeurs manquantes
Avantages :
— Souvent le meilleur en compétitions
— Plus rapide que Gradient Boosting classique
— Régularisation L1/L2 intégrée
— Gestion automatique des valeurs manquantes
Inconvénients :
— Beaucoup d'hyperparamètres à tuner
— Peut être "overkill" pour des problèmes simples
— Nécessite compréhension approfondie
Hyperparamètres clés :
— n_estimators : Nombre d'arbres
— learning_rate : Taux d'apprentissage
— max_depth : Profondeur des arbres
— colsample_bytree : Fraction de features par arbre
— reg_alpha/reg_lambda : Régularisation L1/L2

MLP (Réseau de Neurones)

Principe :
— Réseau de neurones artificiels organisés en couches
— Chaque neurone applique : activation(weighted_sum(inputs) + bias)
— Apprentissage par rétropropagation du gradient
— Peut approximer n'importe quelle fonction
Avantages :
— Peut capturer des relations très complexes
— Flexible et adaptable
— Bonne généralisation avec assez de données
Inconvénients :
— Nécessite beaucoup de données
— Boîte noire (difficile à interpréter)
— Sensible au scaling des features
— Lent à entraîner
— Instable (résultats variables)
Hyperparamètres clés :
— hidden_layer_sizes : Architecture (nombre et taille des couches)
— activation : Fonction d'activation (relu, tanh)
— alpha : Régularisation L2
— learning_rate_init : Taux d'apprentissage initial