Modèles XGBoost

5. Modèles – Boosting – XGBoost

Extreme Gradient Boosting (2014) est une implémentation avancée du Gradient Boosting.

XGBoost est un algorithme d'arbre de décision boosté par le gradient. Il existe depuis 2014 et a fini par dominer Kaggle et la communauté des sciences des données. XGB a introduit le boosting par gradient, où de nouveaux modèles sont adaptés aux résidus des modèles précédents, puis ajoutés ensemble, en utilisant un algorithme de descente de gradient pour minimiser la perte.

Paramètres généraux : Ils définissent la fonctionnalité globale de XGBoost.

booster [default=gbtree] Sélectionnez le type de modèle à exécuter à chaque itération. Il y a 2 options : gbtree : modèles basés sur des arbres gblinear : modèles linéaires

silent [default=0] : Le mode silencieux est activé est fixé à 1, c'est-à-dire qu'aucun message d'exécution ne sera imprimé. Il est généralement bon de garder la valeur 0 car les messages peuvent aider à comprendre le modèle.

nthread [par défaut, le nombre maximum de threads disponibles si non défini]. Ceci est utilisé pour le traitement parallèle et le nombre de cœurs dans le système doit être entré.Si vous souhaitez exécuter sur tous les cœurs, la valeur ne doit pas être saisie et l'algorithme le détectera automatiquement.

Paramètres de la tâche d'apprentissage : ces paramètres sont utilisés pour définir l'objectif d'optimisation et la métrique à calculer à chaque étape.

objectif [default=reg:linear] : Ceci définit la fonction de perte à minimiser. Les valeurs les plus utilisées sont :
binary:logistic -régression logistique pour la classification binaire, renvoie la probabilité prédite (et non la classe).
multi:softmax -classification multiclasse utilisant l'objectif softmax, renvoie la classe prédite (pas les probabilités)
Vous devez également définir un paramètre supplémentaire num_class (nombre de classes) définissant le nombre de classes uniques.

5. Modèles – Boosting – XGBoost

Paramètres du Booster :

eta [default=0.3] / Analogue au taux d'apprentissage dans GBM Rend le modèle plus robuste en réduisant les poids à chaque étape. Valeurs finales typiques à utiliser : 0.01-0.2

min_child_weight [default=1]/ Définit la somme minimale des poids de toutes les observations requises dans un enfant.
Ceci est similaire à min_child_leaf dans GBM mais pas exactement. Cela fait référence à la "somme des poids" minimale des observations alors que GBM a un "nombre d'observations" minimal.
Utilisé pour contrôler l'over-fitting. Des valeurs plus élevées empêchent un modèle d'apprendre des relations qui pourraient être très spécifiques à l'échantillon particulier sélectionné pour un arbre.
Des valeurs trop élevées peuvent conduire à un sous-ajustement et doivent donc être réglées à l'aide de CV.

max_depth [default=6]/ La profondeur maximale d'un arbre, comme GBM. Utilisé pour contrôler l'excès d'ajustement car une profondeur plus élevée permettra au modèle d'apprendre des relations très spécifiques à un échantillon particulier.
Doit être réglé à l'aide de CV. Valeurs typiques : 3-10

max_leaf_nodes : Le nombre maximum de nœuds terminaux ou de feuilles dans un arbre. Peut être défini à la place de max_depth. Comme les arbres binaires sont créés, une profondeur de 'n' produirait un maximum de 2^n feuilles.
Si cette valeur est définie, GBM ignorera max_depth.

gamma [default=0] : Un nœud est divisé uniquement lorsque la division qui en résulte donne une réduction positive de la fonction de perte. Gamma spécifie la réduction minimale des pertes requise pour effectuer un fractionnement.
Rend l'algorithme conservateur. Les valeurs peuvent varier en fonction de la fonction de perte et doivent être ajustées.

max_delta_step [default=0] : Le pas delta maximum que nous autorisons pour l'estimation du poids de chaque arbre. Si la valeur est fixée à 0, cela signifie qu'il n'y a pas de contrainte. S'il est défini à une valeur positive, cela peut aider à rendre l'étape de mise à jour plus conservatrice.
Habituellement, ce paramètre n'est pas nécessaire, mais il peut être utile dans la régression logistique lorsque la classe est extrêmement déséquilibrée.
Il n'est généralement pas utilisé mais vous pouvez l'explorer davantage si vous le souhaitez.

subsample [default=1] : Identique au sous-échantillon de GBM. Dénote la fraction d'observations à échantillonner de manière aléatoire pour chaque arbre.
Des valeurs faibles rendent l'algorithme plus conservateur et empêchent l'ajustement excessif, mais des valeurs trop faibles peuvent conduire à un ajustement insuffisant.
Valeurs typiques : 0.5-1

colsample_bytree [default=1] (en anglais) : Similaire à max_features dans GBM. Indique la fraction de colonnes à échantillonner de manière aléatoire pour chaque arbre.
Valeurs typiques : 0.5-1

colsample_bylevel [default=1] : Indique le rapport de sous-échantillonnage des colonnes pour chaque division, dans chaque niveau.
Je ne l'utilise pas souvent car subsample et colsample_bytree feront le travail pour vous. Mais vous pouvez explorer davantage si vous le souhaitez.

lambda [default=1] : Terme de régularisation L2 sur les poids (analogue à la régression Ridge)
Ceci est utilisé pour gérer la partie régularisation de XGBoost. Bien que de nombreux data scientists ne l'utilisent pas souvent, il devrait être exploré pour réduire l'overfitting.

alpha [default=0] : Terme de régularisation L1 sur le poids (analogue à la régression Lasso)
Peut être utilisé en cas de dimensionnalité très élevée afin que l'algorithme s'exécute plus rapidement lorsqu'il est mis en œuvre.

scale_pos_weight [par défaut=1] : Une valeur supérieure à 0 devrait être utilisée en cas de fort déséquilibre des classes, car elle permet une convergence plus rapide.

5. Modèles – Boosting – XGBoost

Avantages (%GBM) :

- Régularisation : L'implémentation GBM standard n'a pas de régularisation comme XGBoost, ce qui permet de réduire l'overfitting. En fait, XGBoost est également connu comme une technique de "boosting régularisé".
- Traitement parallèle : XGBoost met en œuvre un traitement parallèle et est incroyablement plus rapide que GBM.

Mais attendez, nous savons que le boosting est un processus séquentiel, alors comment peut-il être parallélisé ? Nous savons que chaque arbre ne peut être construit qu'après le précédent, alors qu'est-ce qui nous empêche de faire un arbre en utilisant tous les cœurs ? J'espère que vous comprenez où je veux en venir. Consultez ce lien pour aller plus loin.

- XGBoost supporte également l'implémentation sur Hadoop.
- Grande flexibilité : XGBoost permet aux utilisateurs de définir des objectifs d'optimisation et des critères d'évaluation personnalisés. Cela ajoute une toute nouvelle dimension au modèle et il n'y a aucune limite à ce que nous pouvons faire.
- Traitement des valeurs manquantes : XGBoost dispose d'une routine intégrée pour gérer les valeurs manquantes. L'utilisateur doit fournir une valeur différente des autres observations et la passer comme paramètre. XGBoost essaie différentes choses lorsqu'il rencontre une valeur manquante sur chaque nœud et apprend quel chemin prendre pour les valeurs manquantes à l'avenir.
- Élagage de l'arbre : Un GBM arrête de diviser un nœud lorsqu'il rencontre une perte négative dans la division. Il s'agit donc plutôt d'un algorithme avide. XGBoost, quant à lui, effectue des scissions jusqu'à la profondeur maximale spécifiée, puis commence à élaguer l'arbre en arrière et supprime les scissions au-delà desquelles il n'y a pas de gain positif. Un autre avantage est que parfois, une division de perte négative, par exemple -2, peut être suivie d'une division de perte positive +10. GBM s'arrêtera lorsqu'il rencontrera -2. Mais XGBoost ira plus loin et verra un effet combiné de +8 de la division et gardera les deux.
- Validation croisée intégrée : XGBoost permet à l'utilisateur d'exécuter une validation croisée à chaque itération du processus de boosting et il est donc facile d'obtenir le nombre optimal exact d'itérations de boosting en une seule exécution. Contrairement à GBM, où nous devons effectuer une recherche sur grille et où seules quelques valeurs peuvent être testées.
- Continuer sur le modèle existant : L'utilisateur peut commencer à former un modèle XGBoost à partir de la dernière itération de l'exécution précédente. Cela peut être un avantage significatif dans certaines applications spécifiques. L'implémentation GBM de sklearn possède également cette fonctionnalité, ils sont donc à égalité sur ce point.
- Ancien : documentation et beaucoup d’exemples