XGBoost : A Scalable Tree Boosting System

Résumé fait par Thomas Binet

Sup Galilée

Un algorithme important

De nos jours, de plus en plus d'algorithme de machine learning en apprentissage supervisé sont crées pour répondre aux besoins de vitesse d'apprentissage et de calcul. Malgré cela, il est rare de voir des algorithmes sachant gérer les grandes bases de données pourtant très courantes dans le monde de l'industrie. C'est ce que l'algorithme de boosting : XGBoost à voulu faire. Implémenter un algorithme de boosting optimisé et simplifié au maximum pour gérer les très grosses bases de données. Les majeurs contributions de l'algorithme sont les suivantes :

- créer un algorithme optimisé pour la création extensible d'arbre.
- Proposition d'une justification de l'utilisation de quantiles pondérés.
- Algorithme permettant l'apprentissage avec NaN en calcul parallèle.
- Proposition d'une structure pour optimiser l'espace mémoire.

Modèle

Comme dans chaque algorithme de machine learning utilisant la méthode de boosting en apprentissage supervisé, le but va être d'optimiser une certaine fonction de perte (loss fonction) en ajoutant des arbres peu élaborés (peu de feuilles). L'expression de la prédiction de la cible y_i par la somme d'arbres $(f_k)_{k\in K}est$:

$$\hat{y}_i = \phi(x_i) = \sum_{k=1}^{K} f_k(x_i)$$

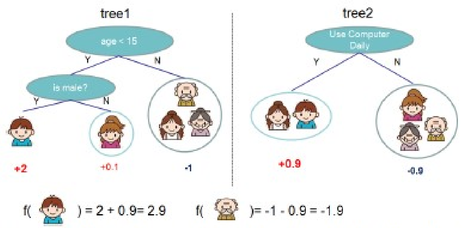

Figure 1: Tree Ensemble Model. The final prediction for a given example is the sum of predictions from each tree.

Fig. 1: Arbres candidats

Pour choisir entre les deux arbres de décisions, on va vouloir minimiser la fonction suivante :

$$\mathcal{L}(\phi) = \sum_i l(\hat{y}_i, y_i) + \sum_k \Omega(f_k)$$

Avec $\Omega(f) = \gamma T + \frac{1}{2}\lambda||\omega||^2$

En se situant à l'étape (t) de notre algorithme, i.e après avoir ajouté $(t-1)$ arbres de décisions à notre modèle, on va ajouter un t ème arbre tel que la fonction suivante sera à son minimum :

$$\mathcal{L}(\phi) = \sum_i l(y_i, \hat{y}_i^{(t-1)+f_t}) + \Omega(f_t)$$

Description de l'arbre minimisant

On peut alors pour simplifier l'algorithme (connaitre plus facilement la structure de l'arbre),en faisant un développement de Taylor à l'ordre 2, et en enlevant les termes contants on obtient :

$$\begin{aligned}\tilde{\mathcal{L}}(\phi) &= \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \Omega(f_t) \\ &= \sum_{i=1}^{n}[g_i f_t(x_i) + \frac{1}{2}h_i f_t^2(x_i)] + \gamma T + \frac{1}{2}\lambda\sum_{j=1}^{T}\omega_j^2 \\ &= \sum_{j=1}^{T}[(\sum_{i\in I_j} g_i)w_j + \frac{1}{2}(\sum_{i\in I_j} h_i + \lambda)w_j^2] + \gamma T\end{aligned}$$

Où les fonctions g_i et h_i sont respectivement les gradients et les hessiennes évalués en x_i, et w_j l'output de l'arbre T pour les $x_i \in I_j$. On a alors une équation du second ordre en w_j, convexe, dont on peut exhiber la valeur minimisant la fonction loss :

$$w_j^* = -\frac{(\sum_{i\in I_j} g_i)}{\sum_{i\in I_j} h_i + \lambda}$$

Dont la valeur sera :

$$\tilde{\mathcal{L}}^{(t)}(q) = -\frac{1}{2}\sum_{j=1}^{T}\frac{(\sum_{i\in I_j} g_i)^2}{\sum_{i\in I_j} h_i + \lambda} + \gamma T$$

On vient alors de donner les formules utiles pour la création de l'arbre optimal.

Algorithme approximé

Comme le but va être d'aller vite lors de la création d'arbre de décision, pour trouver quelle est la meilleure séparation des données, on ne va pas traiter tout les cas un à un (car pour de grand data set cela serait trop long), mais utiliser les quantiles de la distribution pour avoir des bons candidats de séparation dès le début. Il restera plus qu'à comparer les différents gain en fonction des quantiles.

Algorithm 1: Exact Greedy Algorithm for Split Finding

Input: I, instance set of current node
Input: d, feature dimension
$gain \leftarrow 0$
$G \leftarrow \sum_{i\in I} g_i$, $H \leftarrow \sum_{i\in I} h_i$
for $k = 1$ *to* m **do**
  $G_L \leftarrow 0$, $H_L \leftarrow 0$
  for j *in sorted(I, by* $\mathbf{x}_{jk}$*)* **do**
    $G_L \leftarrow G_L + g_j$, $H_L \leftarrow H_L + h_j$
    $G_R \leftarrow G - G_L$, $H_R \leftarrow H - H_L$
    $score \leftarrow \max(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
  end
end
Output: Split with max score

Algorithm 2: Approximate Algorithm for Split Finding

for $k = 1$ *to* m **do**
  Propose $S_k = \{s_{k1}, s_{k2}, \cdots s_{kl}\}$ by percentiles on feature k.
  Proposal can be done per tree (global), or per split(local).
end
for $k = 1$ *to* m **do**
  $G_{kv} \leftarrow= \sum_{j\in\{j|s_{k,v}\geq \mathbf{x}_{jk} > s_{k,v-1}\}} g_j$
  $H_{kv} \leftarrow= \sum_{j\in\{j|s_{k,v}\geq \mathbf{x}_{jk} > s_{k,v-1}\}} h_j$
end
Follow same step as in previous section to find max score only among proposed splits.

Fig. 2: Algorithme approximé.

La complexité en temps est alors de $\mathcal{O}(K||x||_0 \log q)$. Ou q est le nombre de candidat proposé pour le data set (entre 32 et 100).

Optimisations

Une fois l'algorithme global fait, XGBoost à implémenté beaucoup de moyens pour optimiser le temps de calcul, comme l'utilisation des plusieurs unité de calcul pour séparer le data set et calculer les quantiles pondérés en simultané. Mais aussi l'incrémentassions d'une voie par défaut dans chaque arbre maximisant le gain, qui prendra en compte les valeurs non définis dans l'algorithme.

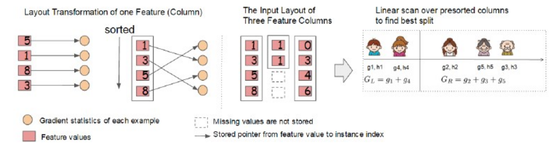

Fig. 3: Structure pour apprentissage en parallèle .

De plus XGBoost utilise des méthodes de Cache-aware Access et de Blocks pour l'Out-of-core Computation pour optimiser au mieux son algorithme sur un ordinateur.

Comparaison

Pour finir, on souhaite comparer XGBoost avec différents algorithmes. On va d'abord regarder l'implémentation des différentes méthodes de XGBoost sur les autres algorithmes classiques de boosting via arbre de décision. On remarque que XGBoost est de loin le seul à utiliser autant de techniques avancées pour optimiser son temps de calcul.

Table 1: Comparison of major tree boosting systems.

System	exact greedy	approximate global	approximate local	out-of-core	sparsity aware	parallel
XGBoost	yes	yes	yes	yes	yes	yes
pGBRT	no	no	yes	no	no	yes
Spark MLLib	no	yes	no	no	partially	yes
H2O	no	yes	no	no	partially	yes
scikit-learn	yes	no	no	no	no	no
R GBM	yes	no	no	no	partially	no

Fig. 4: Comparaison des différents algorithmes de TreeBoosting .

Et enfin une comparaison du temps de calcul d'un arbre pour de très grand data set selon la méthode utilisée, on remarque que les algorithmes classiques s'arrêtent très tôt tandis que XGBoost peut traiter énormément d'informations.

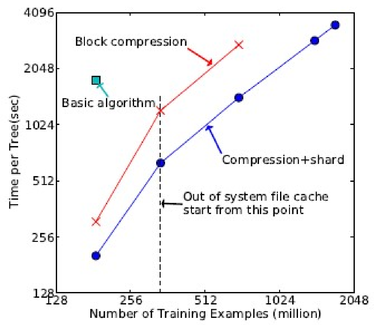

Fig. 5: Comparaison du temps de calcul sur différentes méthodes.