

XGBoost: A Scalable Tree Boosting System

01

XGBOOST NEDİR?

Açılımı Extreme Gradient Boosting'dir.

Makale genelinde algoritmanın avantajları farklı alt başlıklarla anlatılmış ve kullanım örnekleri verilmiştir. Bunlar sırasıyla seyrek veriye duyarlılık(sparsity-aware), ağırlıklı nicelik çizimi(weighted quantile sketch), önbellek duyarlılığı(cache-aware) ile veri sıkıştırma ve parçalama(compression and sharding).

02

Neden Önemli?

Kaggle'ın 2015'deki 28 yarışmanın 17 tanesi XGBOOST ile, 11 tanesi Deep Neural Nets ile kazanılmıştır.

Hem classification hem regression problemlerini gerçekçi şekilde çözebiliyor. Tüm senaryolarda ölçülebilir, tek makinede çalışan veya dağıtılmış data ile çalıştırılan algoritmaların yanında on kat daha hızlı çalışıyor.

GRADIENT TREE BOOSTING

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

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

Hataları tahminleyen karar ağacı kurulur. Burada amaç hataları öğrenip doğru tahmine yaklaşmaktadır. Ağacın her bir dalı için benzerlik skoru(similarity score) hesaplanır. Dalların ne kadar iyi oluşturulduğunu gösterir. İkinci denklem ise ağacın kalitesini ölçmek amacıyla oluşturulmuştur.

Tree Building

Algoritmada amaç optimal kazancı bulmaktadır. Bunun için ağaçlar oluşturulur. Ağaçlarda dalların kazançları hesaplanır ve buna göre score güncellenir.

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 $\text{sorted}(I, \text{ by } \mathbf{x}_{jk})$ **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

Approximate Algorithm

Exact Greedy Algorithm başarılı olsa da gerçek hayatta hafıza tüm datayı işleyebilecek güçte olmayacağıdır. Bu durumda algoritmanın yavaşlaması yüksek ihtimaldir. Özetle, algoritma yüzdelik feature bölme noktaları belirler. Featurelar bucketlara işlenir. İstatistikler toplanır ve buna göre en iyi çözüm belirlenir.

Shrinkage ve Column Subsampling

Shrinkage hangi feature'ın daha etkin hangisinin daha az etkili olmasını kararlaştırmak için kullanılan bir parametredir.

Column Subsampling Random Forest'a da kullanılmaktadır. Row Subsampling(batch)'den daha iyi overfiti engelliyor. Ayrıca daha hızlı çözüm üretiliyor.

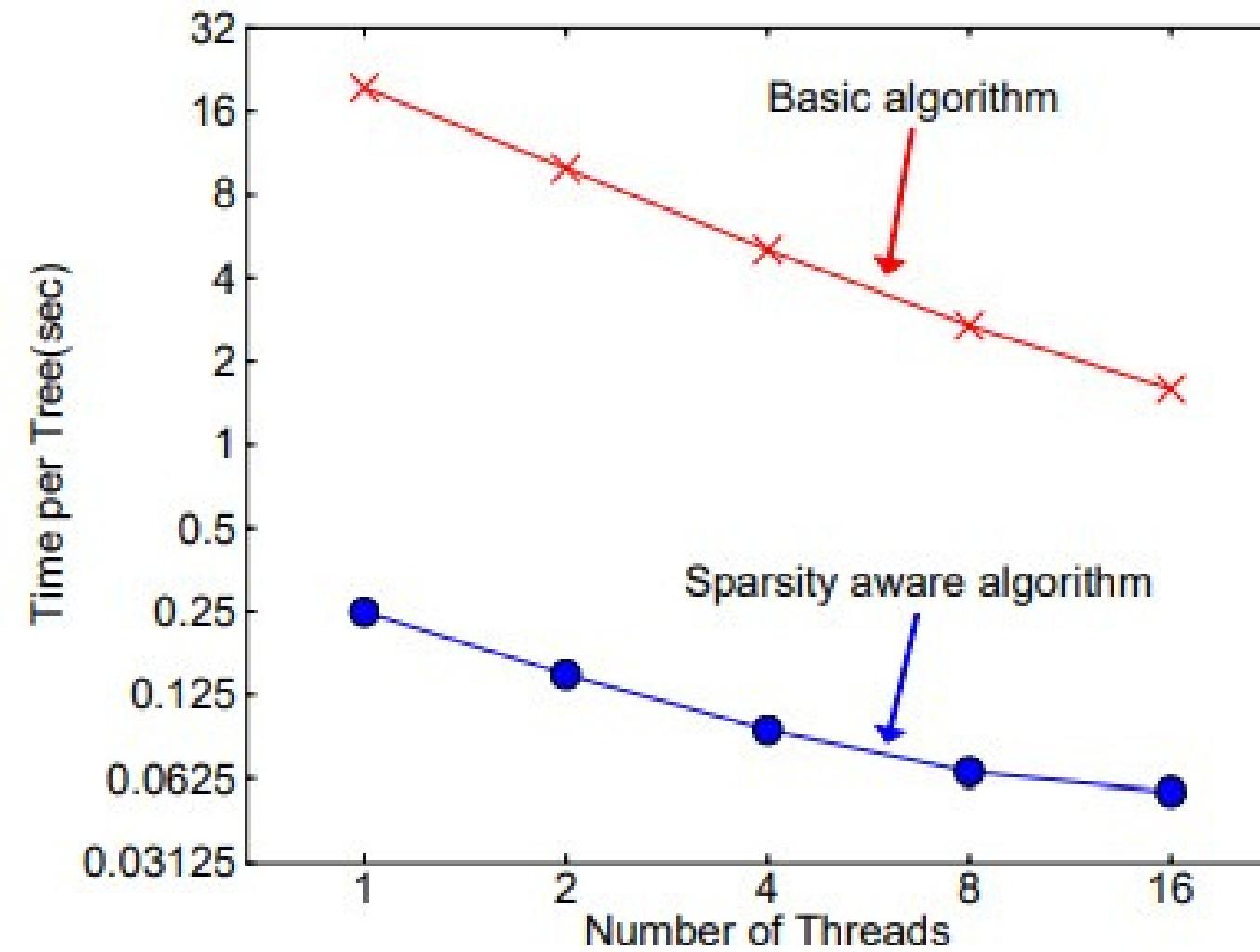
07

Ağırlıklı Nicelik Çizimi

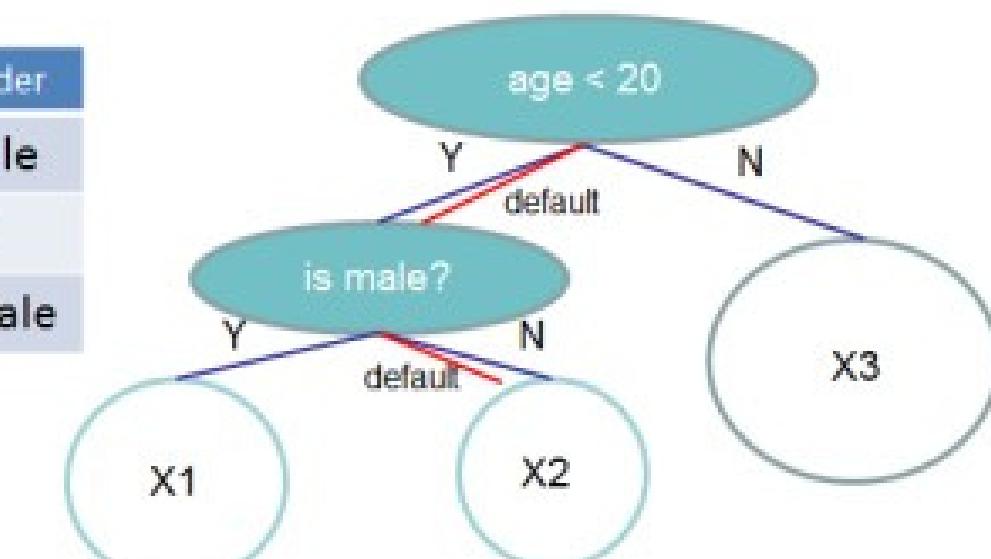
XGBoost, verideki her değeri incelemek yerine veriyi parçalara(quantile) böler ve bu parçalara göre çalışır. Parça miktarı artırıldıkça algoritma daha küçük aralıklara bakacak ve daha iyi tahminleme yapacaktır. Tabi ki bu durum modelin öğrenme süresi de artacaktır.

Sorunu aşmak için “Sketches” adı verilen algoritma kullanılır. Amacı parçaları bulmak için yakınsama yapmasıdır.

Sparsity-aware



Data		
Example	Age	Gender
X1	?	male
X2	15	?
X3	25	female

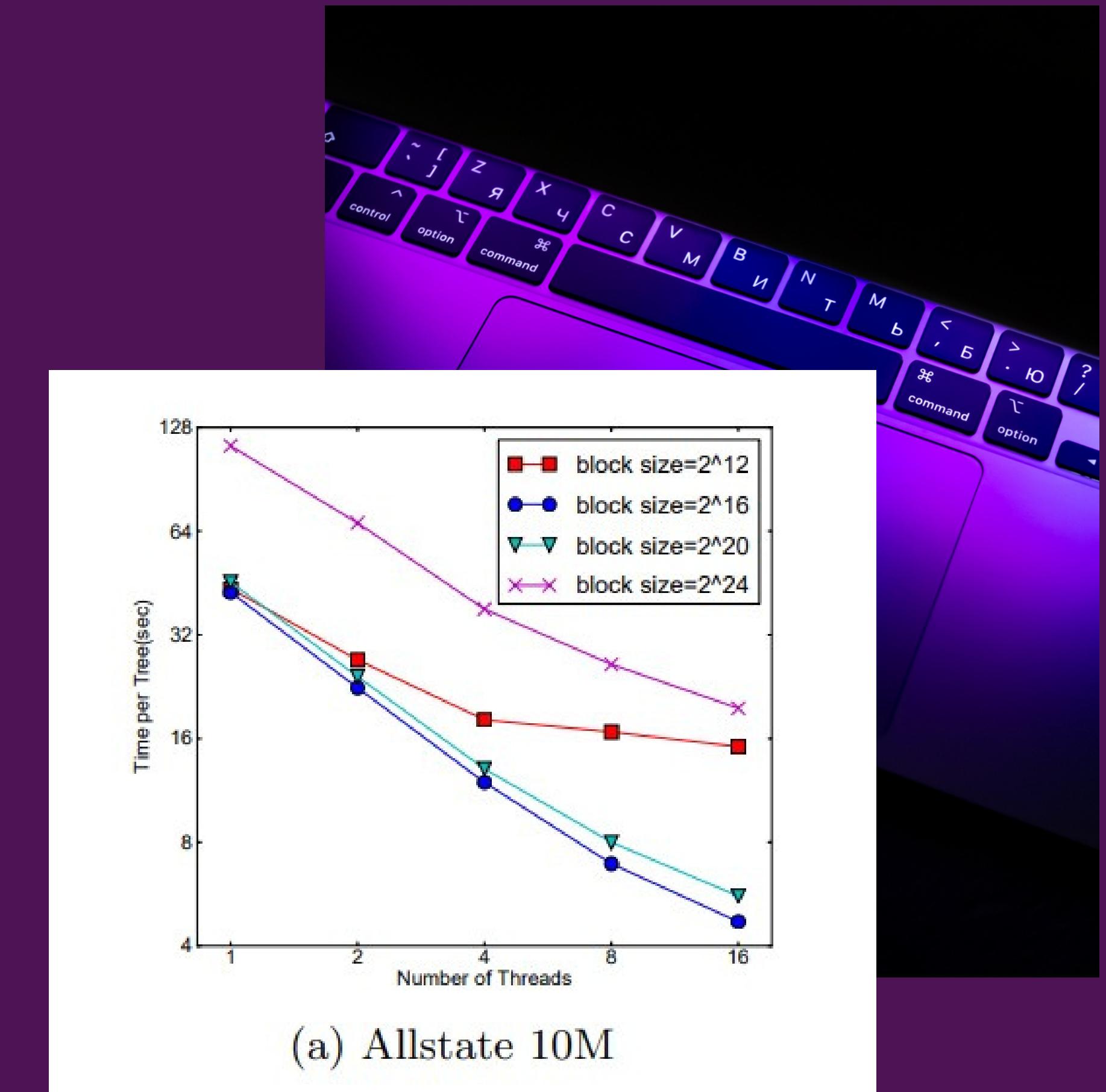


Column Block For Parallel Learning

Data blocklar halinde hafıza saklanır. Eğitimden önce yalnızca bir kez hesaplanması gerekir ve sonraki yinelemelerde yeniden kullanılabilir.

Cache-aware Access

Temel halinde okumak yazma işlemleri belleğe bağımlılık ortaya koyar. CPU'ya sığmama durumunda sistem yavaşlar. Cache-aware ile sistem hafifler.



Blocks for out-of-Core

Block Compression

Columnları sıkıştırıp, thread içinde hafızaya yüklenirken ters işlem uygulanır.
Zaman karmaşıklığını küçültür.

Block Sharding

Multidisklerde çalıştırılır ve threadlere yüklenir.
Verimlilik bu yöntemle artar.

12

Kullanım

Birçok ekosistemde kullanabilmek mümkün.

R, Python, Julia

Dağıtım sürümünün kullanılabildiği yerler:

Hadoop, MPI Sun Grid engine, Flink, Spark, Tianchi

13 Örnek

Table 3: Comparison of Exact Greedy Methods with 500 trees on Higgs-1M data.

Method	Time per Tree (sec)	Test AUC
XGBoost	0.6841	0.8304
XGBoost (colsample=0.5)	0.6401	0.8245
scikit-learn	28.51	0.8302
R.gbm	1.032	0.6224

Table 4: Comparison of Learning to Rank with 500 trees on Yahoo! LTRC Dataset

Method	Time per Tree (sec)	NDCG@10
XGBoost	0.826	0.7892
XGBoost (colsample=0.5)	0.506	0.7913
pGBT [22]	2.576	0.7915

Sonuç

Sware-aware ve approximate algoritması için doğrulanmış ağırlıklı nicel çizimle birlikte cache-aware ile compression ve sharding konuşuldu.