



ТЕХНОСФЕРА

Лекция 3 XGBoost

Владимир Гулин

16 сентября 2019 г.

План лекции

Напоминание

Регуляризация структуры деревьев

XGBoost

DART

Задача обучения с учителем

Постановка задачи

Пусть дан набор объектов

$\mathcal{D} = \{(\mathbf{x}_i, y_i)\}$, $\mathbf{x}_i \in \mathcal{X}$, $y_i \in \mathcal{Y}$, $i \in 1, \dots, N$, полученный из неизвестной закономерности $y = f(\mathbf{x})$. Необходимо построить такую $h(\mathbf{x})$, которая наиболее точно аппроксимирует $f(\mathbf{x})$.

Будем искать неизвестную

$$h(\mathbf{x}) = C(a_1(\mathbf{x}), \dots, a_T(\mathbf{x}))$$

$a_i(\mathbf{x}) : \mathcal{X} \rightarrow \mathcal{R}$, $\forall i \in \{1, \dots, T\}$ - базовые модели

$C : \mathcal{R} \rightarrow \mathcal{Y}$ - решающее правило

Алгоритмические композиции

Simple Voting

$$h(\mathbf{x}) = \frac{1}{T} \sum_{i=1}^T a_i(\mathbf{x})$$

Weighted Voting

$$h(\mathbf{x}) = \frac{1}{T} \sum_{i=1}^T b_i a_i(\mathbf{x}), \quad b_i \in \mathcal{R}$$

Mixture of Experts

$$h(\mathbf{x}) = \frac{1}{T} \sum_{i=1}^T b_i(\mathbf{x}) a_i(\mathbf{x}), \quad b_i(\mathbf{x}) : \mathcal{X} \rightarrow \mathcal{R}$$

AdaBoost(Freund & Shapire 1995)

1. Инициализировать веса объектов $w_j = 1/N, j = 1, 2, \dots, N$.
2. Для всех i от 1 до T :

(a) Построить классификатор $a_i(\mathbf{x})$, используя веса w_j

(b) Вычислить

$$err_i = \frac{\sum_{j=1}^N w_j I(y_j \neq a_i(\mathbf{x}_j))}{\sum_{j=1}^N w_j}$$

(c) Вычислить

$$b_i = \frac{1}{2} \log \frac{1 - err_i}{err_i}$$

(d) Присвоить $w_j \rightarrow w_j \cdot \exp[b_i \cdot I(y_j \neq a_i(\mathbf{x}_j))], j = 1, \dots, N$.

(e) Нормируем веса объектов

$$w_j \rightarrow \frac{w_j}{\sum_{k=1}^N w_k}, j = 1, \dots, N.$$

3. $h(\mathbf{x}) = \text{sign} \left[\sum_{i=1}^T b_i a_i(\mathbf{x}) \right]$

Gradient boosting algorithm

1. Инициализировать $h_0(\mathbf{x}) = \operatorname{argmin}_{\gamma} \sum_{j=1}^N L(y_j, \gamma)$
2. Для всех i от 1 до T :
 - (a) Для всех $j = 1, 2, \dots, N$ вычислить

$$g_{i,j} = - \left[\frac{\partial L(y_j, h(\mathbf{x}_j))}{\partial h(\mathbf{x}_j)} \right]_{h=h_{i-1}}$$

- (b) Построить базовую модель a_i на ответах $g_{i,j}$

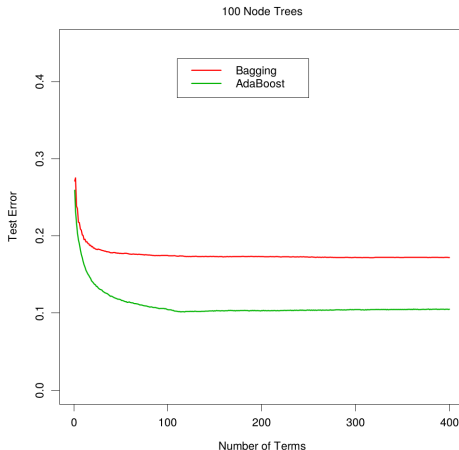
$$a_i = \operatorname{argmin}_a \sum_{j=1}^N (g_{i,j} - a(\mathbf{x}_j))^2$$

- (c) Определить вес b_i

$$b_i = \operatorname{argmin}_b \sum_{j=1}^N L(y_j, h_{i-1}(\mathbf{x}) + b \cdot a_i(\mathbf{x}_j))$$

- (d) Присвоить $h_i(\mathbf{x}) = h_{i-1}(\mathbf{x}) + b_i \cdot a_i(\mathbf{x})$
3. Вернуть $h(\mathbf{x}) = h_T(\mathbf{x})$

Boosting vs Bagging



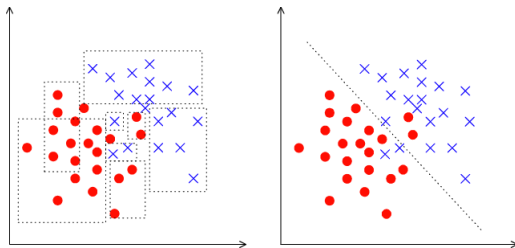
В общем случае:

BagBoo & Boobag > Gradient Boosting > Bagging > Single Tree

Недостатки градиентного бустинга

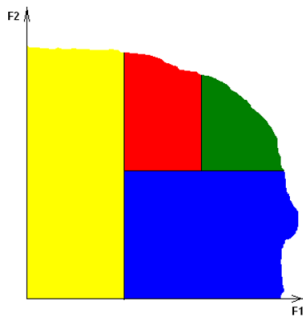
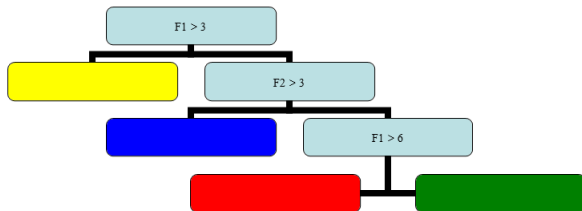
- ▶ Переобучается на “малом” количестве данных
- ▶ Медленно сходится к точке оптимума, из-за shrinkage
- ▶ Строит неоптимальный набор базовых моделей

Регуляризация структуры деревьев

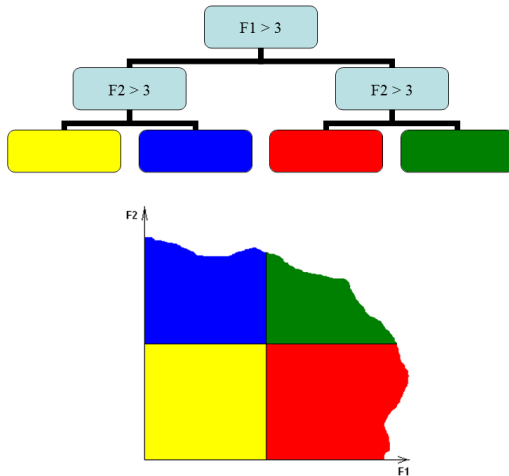


- ▶ Предположим, что у нас есть n точек в листе
- ▶ Предложение Фридмана: не будем строить поддереву, если $n < p$
- ▶ Как еще можно добиться “кашерности” дерева?

Decision Tree

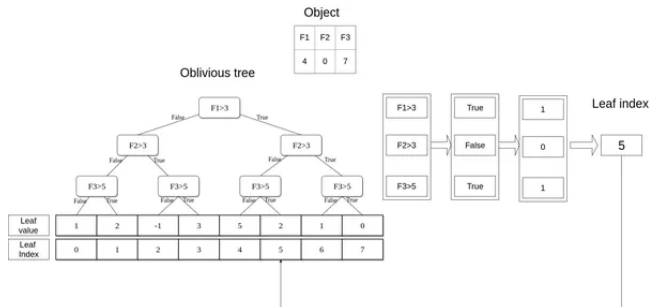


Oblivious Decision Tree



- ▶ Как построить такое дерево?
- ▶ Как хранить?

Oblivious Decision Tree



- Как эффективно применять?

Oblivious Decision Tree

Применялка

```
1  int index = 0;
2  for (int depth = 0; depth < tree.ysize(); ++depth) {
3      index |= binFeatures[tree[depth]] << depth;
4  }
5  result += Model.LeafValues[treeId][resultId][index];
```

- ▶ В продакшене > 10x быстрее обычных
- ▶ Откуда???

Leaf value scaling

- ▶ Oblivious деревья - полные бинарные деревья
- ▶ Вместо ограничения на количество примеров штрафует “ненадежные” листья
- ▶ Домножим значения в листьях дерева на

$$\sqrt{\frac{n}{n+k}} \rightarrow 1, n \rightarrow \infty$$

$$\log \left(1 + \frac{n}{n+k} \right) \rightarrow 1, n \rightarrow \infty$$

$$\frac{n}{1+n+\sqrt{n}}$$

XGBoost (2016)

eXtreme Gradient Boosting

Крайне широко используется на Kaggle соревнованиях в настоящее время.

- ▶ Прост в использовании
 - ▶ Легко установить
 - ▶ Есть пакеты для R и python
- ▶ Эффективен
 - ▶ Автоматически параллелится на одной машине
 - ▶ Имеется возможность запуска на кластере
- ▶ Показывает хорошие результаты
 - ▶ В среднем, для большинства датасетов
- ▶ Обладает гибкостью
 - ▶ Можно задавать собственную целевую функцию для оптимизации
 - ▶ Можно тюнить внутренние параметры алгоритма

XGBoost

Идея:

Ошибка композиции на обучающей выборке для градиентного бустинга

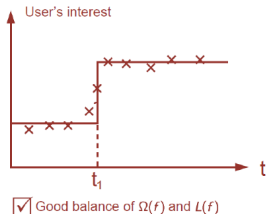
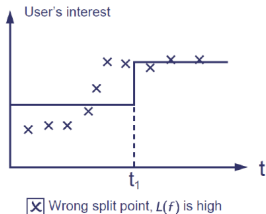
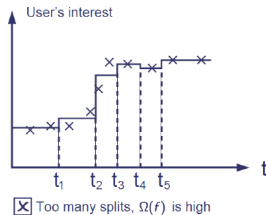
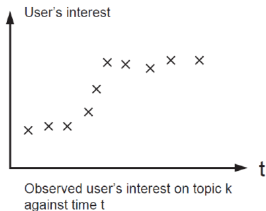
$$err(h) = \sum_{j=1}^N L(y_j, \sum_{i=1}^T b_i a_i(\mathbf{x}_j))$$

XGBoost явно добавляет регуляризацию в этот функционал

$$err(h) = \sum_{j=1}^N L(y_j, \sum_{i=1}^T b_i a_i(\mathbf{x}_j)) + \sum_{i=1}^T \Omega(a_i)$$

XGBoost

Компромисс между функцией потерь и регуляризатором



XGBoost

Задача оптимизации:

Как мы обучаем обычный градиентный бустинг

$$err(h) = \sum_{j=1}^N L(y_j, \sum_{i=1}^{T-1} b_i a_i(\mathbf{x}_j) + b \cdot a(\mathbf{x}_j)) \rightarrow \min_{b, a}$$

Соответственно для XGBoost задача оптимизации модифицируется следующим образом:

$$err(h) = \sum_{j=1}^N L(y_j, \sum_{i=1}^{T-1} b_i a_i(\mathbf{x}_j) + b \cdot a(\mathbf{x}_j)) + \sum_{i=1}^{T-1} \Omega(a_i) + \Omega(a) \rightarrow \min_{b, a}$$

XGBoost

Наша текущая цель научиться подбирать базовую модель на каждой итерации бустинга

$$\text{err}(h^{(T)}) = \sum_{j=1}^N L(y_j, h^{(T-1)} + a(\mathbf{x}_j)) + \Omega(a) + \text{const}$$

Воспользуемся разложение в ряд Тейлора

$$f(x + \delta x) \approx f(x) + f'(x)\delta x + \frac{1}{2}f''(x)\delta x^2$$

Обозначим

$$g_j = \left[\frac{\partial L(y_j, h(\mathbf{x}_j))}{\partial h(\mathbf{x}_j)} \right]_{h=h^{(T-1)}}, \quad s_j = \left[\frac{\partial^2 L(y_j, h(\mathbf{x}_j))}{\partial h(\mathbf{x}_j)^2} \right]_{h=h^{(T-1)}}$$

XGBoost

Таким образом

$$err(h^{(T)}) \approx \sum_{j=1}^N \left[L(y_j, h^{(T-1)}) + g_j a(\mathbf{x}_j) + \frac{1}{2} s_j a^2(\mathbf{x}_j) \right] + \Omega(a) + const$$

Величины, не зависящие от a , можно исключить из выражения

$$\sum_{j=1}^N \left[g_j a(\mathbf{x}_j) + \frac{1}{2} s_j a^2(\mathbf{x}_j) \right] + \Omega(a) \rightarrow \min_a$$

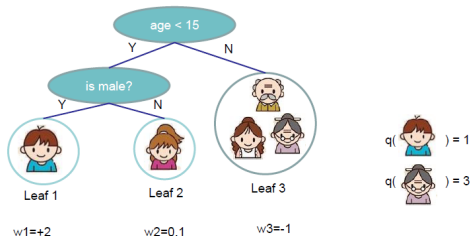
XGBoost

Что такое дерево решений?

Дерево решений состоит из набора предикатов и значений в листьях

$$a(\mathbf{x}) = w_{q(\mathbf{x})}, \quad w \in \mathcal{R}^Z, \quad q: \mathcal{R}^D \rightarrow \{1, \dots, Z\}$$

q - структура дерева, w - значения в листьях, Z - количество листьев



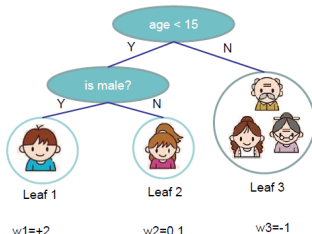
XGBoost

Сложность дерева

Определим сложность дерева как (например)

$$\Omega(a) = \gamma Z + \frac{1}{2} \lambda \sum_{i=1}^Z w_i^2$$

Средневзвешенная числа листьев и L_2 нормы значений в листьях



$$\Omega = \gamma 3 + \frac{1}{2} \lambda (4 + 0.01 + 1)$$

XGBoost

Обозначим факт попаданию примера в k -ый лист

$$I_k = \{j | q(\mathbf{x}_j) = k\}$$

Перепишем целевую функцию в следующем виде:

$$\begin{aligned} & \sum_{j=1}^N \left[g_j a(\mathbf{x}_j) + \frac{1}{2} s_j a^2(\mathbf{x}_j) \right] + \Omega(a) = \\ &= \sum_{j=1}^N \left[g_j w_{q(\mathbf{x}_j)} + \frac{1}{2} s_j w_{q(\mathbf{x}_j)}^2 \right] + \gamma Z + \lambda \frac{1}{2} \sum_{k=1}^Z w_k^2 = \\ &= \sum_{k=1}^Z \left[\left(\sum_{j \in I_k} g_j \right) w_k + \frac{1}{2} \left(\sum_{j \in I_k} s_j + \lambda \right) w_k^2 \right] + \gamma Z \end{aligned}$$

Получили сумму Z независимых квадратичных функций

- Как теперь получить значения, которые должны быть в листьях дерева?

XGBoost

Напоминание. Для квадратичной функции одной переменной

$$\operatorname{argmin}_x Gx + \frac{1}{2}Sx^2 = -\frac{G}{S}, \quad S > 0, \quad \min_x Gx + \frac{1}{2}Sx^2 = -\frac{1}{2} \frac{G^2}{S}$$

Обозначим $G_k = \sum_{j \in I_k} g_j$, $S_k = \sum_{j \in I_k} s_j$

$$\begin{aligned} \sum_{k=1}^Z \left[\left(\sum_{j \in I_k} g_j \right) w_k + \frac{1}{2} \left(\sum_{j \in I_k} s_j + \lambda \right) w_k^2 \right] + \gamma Z = \\ = \sum_{k=1}^Z \left[G_k w_k + \frac{1}{2} (S_k + \lambda) w_k^2 \right] + \gamma Z \end{aligned}$$

Таким образом, для фиксированной структуры дерева $q(\mathbf{x})$, оптимальными значениями для листьев будут

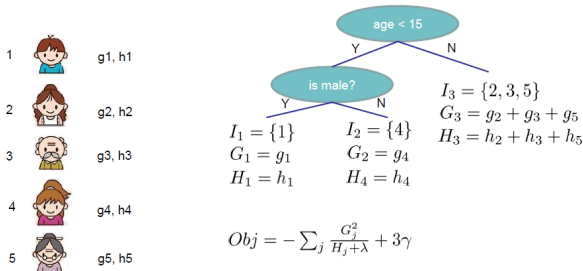
$$w_k^* = -\frac{G_k}{S_k + \lambda}, \quad k = 1, \dots, Z$$

XGBoost

При этом значение оптимизируемой функции, примет вид

$$err = -\frac{1}{2} \sum_{k=1}^Z \frac{G_k^2}{S_k + \lambda} + \gamma Z$$

Фактически, это мера того насколько “кошеровое” дерево мы получили



XGBoost

Как же все таки вырастить дерево?

- ▶ Рассмотрим все возможные структуры дерева...
- ▶ Для каждой структуры вычислим score

$$err = -\frac{1}{2} \sum_{k=1}^Z \frac{G_k^2}{S_k + \lambda} + \gamma Z$$

- ▶ Выбрав лучшую структура дерева, рассчитаем оптимальные значения для листьев

$$w_k^* = -\frac{G_k}{S_k + \lambda}, k = 1, \dots, Z$$

- ▶ Видите ли вы какие-нибудь проблемы в таком алгоритме?

Жадный рекурсивный алгоритм построения дерева

- ▶ Стартуем с дерева глубины 0
- ▶ Для каждого листа дерева пытаемся добавить новое разбиение

$$Gain = \frac{G_l^2}{S_l + \lambda} + \frac{G_r^2}{S_r + \lambda} - \frac{(G_l + G_r)^2}{S_l + S_r + \lambda} - \gamma$$

- ▶ Выбираем лучшее разбиение максимизируя $Gain$

XGBoost

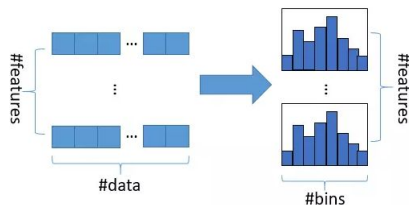
Регуляризация

- ▶ Однако, *Gain* может быть отрицательным, если $\gamma > \text{loss reduction}$

$$\text{Gain} = \frac{G_l^2}{S_l + \lambda} + \frac{G_r^2}{S_r + \lambda} - \frac{(G_l + G_r)^2}{S_l + S_r + \lambda} - \gamma$$

- ▶ Pre-stopping
 - ▶ Останавливаем ветвление, если лучшее разбиение имеет отрицательный *Gain*
 - ▶ Правильно ли так поступать?
- ▶ Post-Pruning
 - ▶ Растим дерево до упора, а затем подрезаем листовые ветвления отрицательным *Gain*

Histogram based tree building



Обозначим $\mathcal{D}_k = \{(x_{1k}, h_1), (x_{2k}, h_2) \dots (x_{nk}, h_n)\}$

$$r_k(z) = \frac{1}{\sum_{(x,h) \in \mathcal{D}_k} h} \sum_{(x,h) \in \mathcal{D}_k, x < z} h$$

Цель: Найти кандидатов для точек-сплитов $\{s_{k1}, s_{k2}, \dots s_{kl}\}$

$$|r_k(s_{k,j}) - r_k(s_{k,j+1})| < \epsilon, \quad s_{k1} = \min_i x_{ik}, \quad s_{kl} = \max_i x_{ik}$$

Approximate Split Finding Algorithm

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 $S_k = \{s_{k1}, s_{k2}, \dots, s_{kl}\}$ 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.

Approximate Split Finding Algorithm

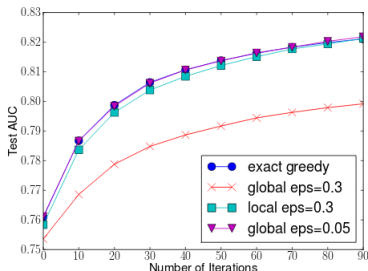


Figure 3: Comparison of test AUC convergence on Higgs 10M dataset. The ϵ s parameter corresponds to the accuracy of the approximate sketch. This roughly translates to $1 / \epsilon$ buckets in the proposal. We find that local proposals require fewer buckets, because it refine split candidates.

- Какие преимущества есть у гистограммного подхода с алгоритмической точки зрения?

Sparse data

- Data Source:
 - Movie ratings
 - Purchase history
- Feature engineering:
 - NLP: CountVectorizer, HashingTF
 - Categorical: OneHotEncoder
 - Image, video



Sparsity-aware Split Finding

Algorithm 3: Sparsity-aware Split Finding

Input: I , instance set of current node

Input: $I_k = \{i \in I | x_{ik} \neq \text{missing}\}$

Input: d , feature dimension

Also applies to the approximate setting, only collect statistics of non-missing entries into buckets

$\text{gain} \leftarrow 0$

$G \leftarrow \sum_{i \in I} g_i, H \leftarrow \sum_{i \in I} h_i$

for $k = 1$ **to** m **do**

// enumerate missing value goto right

$G_L \leftarrow 0, H_L \leftarrow 0$

for j in sorted(I_k , ascent order by \mathbf{x}_{jk}) **do**

$G_L \leftarrow G_L + g_j, H_L \leftarrow H_L + h_j$

$G_R \leftarrow G - G_L, H_R \leftarrow H - H_L$

$\text{score} \leftarrow \max(\text{score}, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda})$

end

// enumerate missing value goto left

$G_R \leftarrow 0, H_R \leftarrow 0$

for j in sorted(I_k , descent order by \mathbf{x}_{jk}) **do**

$G_R \leftarrow G_R + g_j, H_R \leftarrow H_R + h_j$

$G_L \leftarrow G - G_R, H_L \leftarrow H - H_R$

$\text{score} \leftarrow \max(\text{score}, \frac{G_L^2}{H_L + \lambda} + \frac{G_R^2}{H_R + \lambda} - \frac{G^2}{H + \lambda})$

end

end

Output: Split and default directions with max gain

Sparse vs Dense

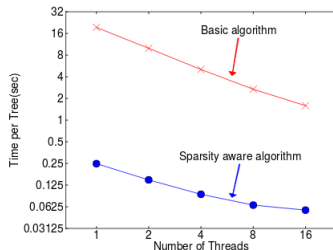
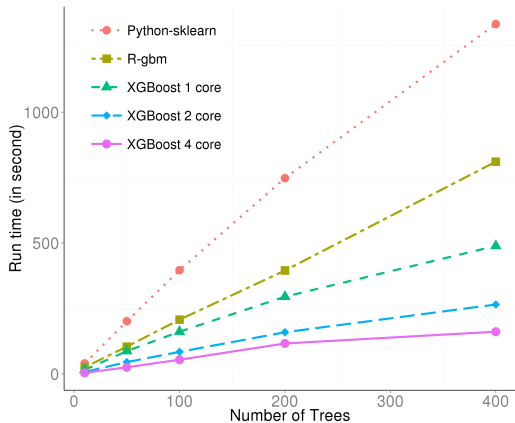


Figure 5: Impact of the sparsity aware algorithm on Allstate-10K. The dataset is sparse mainly due to one-hot encoding. The sparsity aware algorithm is more than 50 times faster than the naive version that does not take sparsity into consideration.

XGBoost

Хорошая реализация на C++



XGBoost

Результаты

- ▶ Kaggle CrowdFlower
- ▶ Kaggle Malware Prediction
- ▶ Kaggle Tradeshift
- ▶ Kaggle Higgs competition
- ▶ ...

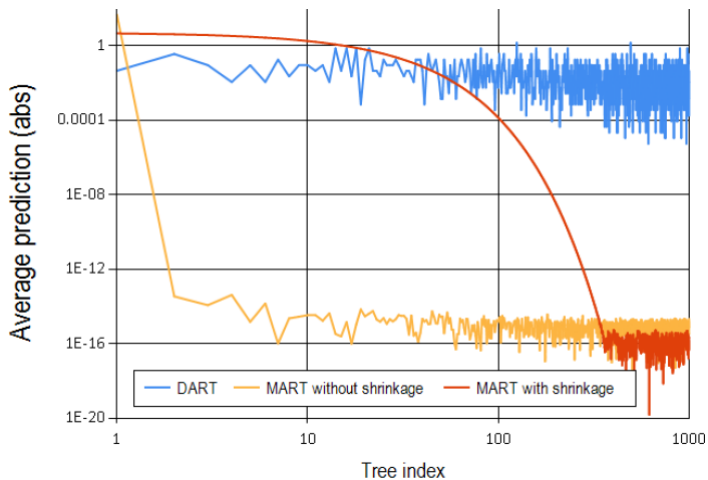
Остальное можно почитать на
<https://xgboost.readthedocs.org>

DART (2015)

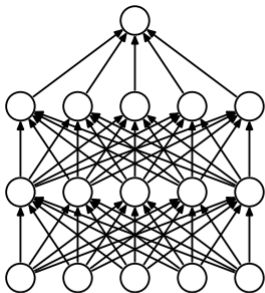
Dropouts meet Multiple Additive Regression Trees

- ▶ Идея состоит в применении техники dropout для GBM
- ▶ DART предотвращает over-specialization
- ▶ Деревья добавленные вначале имеют существенное влияние на результат
- ▶ Shrinkage фиксирует проблему, но не достаточно эффективно

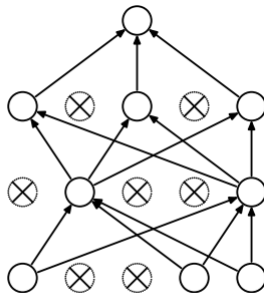
DART



Dropout

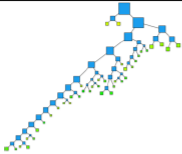
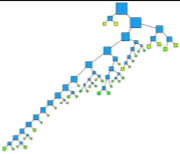
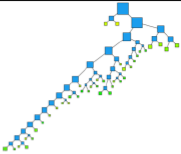
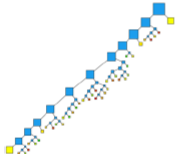


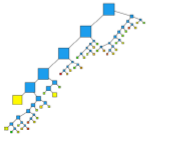
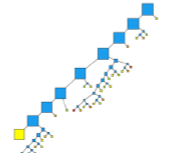
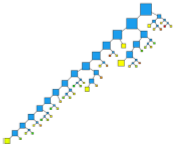


(a) Standard Neural Net



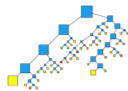
(b) After applying dropout.

DART

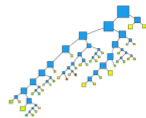
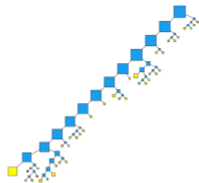
Index	MART without Shrinkage	MART with Shrinkage	DART
1			
100			
			

DART

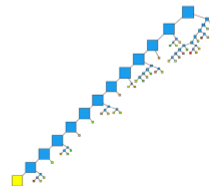
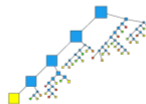
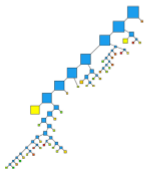
200



400



1000



DART

Algorithm 1 The DART algorithm

Let N be the total number of trees to be added to the ensemble

$S_1 \leftarrow \{x, -L'_x(0)\}$

T_1 be a tree trained on the dataset S_1

$M \leftarrow \{T_1\}$

for $t = 2, \dots, N$ **do**

$D \leftarrow$ the subset of M such that $T \in M$ is in D
with probability p_{drop}

if $D = \emptyset$ **then** $D \leftarrow$ a random element from M
 end if

$\hat{M} \leftarrow M \setminus D$

$S_t \leftarrow \left\{x, -L'_x\left(\hat{M}(x)\right)\right\}$

T_t be a tree trained on the dataset S_t

$M \leftarrow M \cup \left\{\frac{T_t}{|D|+1}\right\}$

for $T \in D$ **do**

 Multiply T in M by a factor of $\frac{|D|}{|D|+1}$

end for

end for

Output M

DART

Регрессия

Ensemble size	25	50	100	250	500	1000
MART	35.13	31.79	30.92	30.07	29.76	29.28
DART	32.50	30.50	29.66	28.14	28.11	27.98
Random Forest	32.76	33.21	32.88	32.36	32.66	32.33

L2

CT slices dataset

DART

Классификация

Ensemble size	50	100	250	500	1000
MART	0.9687	0.9699	0.9707	0.9704	0.9695
DART	0.9676	0.9692	0.9714*	0.9693	0.9699
Random Forest	0.9627	0.9629	0.9629	0.9630	0.9628

Accuracy

Pascal Large scale learning challenge

DART

Ранжирование

algorithm	Shrinkage	Dropout	Loss function parameter	Feature fraction	NDCG@3
MART	0.4	0	1.2	0.75	46.31
DART	1	0.03	1.2	0.5	46.70

NDCG

Yahoo! Learning to rank challenge

Задача

Дано: Имеется набор данных из системы поискового антиспама.

Требуется: Требуется сравнить ранее рассмотренные классификаторы с xgboost и dart.

Пошаговая инструкция

1. Скачать данные и запустить шаблон кода на python
`goo.gl/CCM2Yo`
2. Построить графики качества классификации в зависимости от параметров алгоритмов
3. Построить графики скорости обучения в зависимости от числа базовых моделей

```
$ python compos.py -h  
$ python compos.py -tr spam.train.txt -te spam.test.txt
```

Вопросы

