CatBoost: unbiased boosting with categorical features

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, Andrey Gulin

September 3, 2021

Outline

Outline

Boosting

Given $\mathcal{D} = \{(x_k, y_k)\}_{k=1\ldots n}$, where $x_k = (x_k^1, \ldots, x_k^m)$ is a data point with m features and y_k is the target, we want to learn a model $F : \mathbb{R}^m \to \mathbb{R}$ that minimizes the expected loss function $\mathcal{L}(F) := \mathbb{E}[L(y, F(x))]$.

- Using a sequence of *weak* models to produce a *strong* model
- Approximate F iteratively.
  We build a sequence of models F^t $(t = 0, 1, \ldots)$ s.t.

  $$F^t = F^{t-1} + \alpha h^t$$

  $h : \mathbb{R}^m \to \mathbb{R}$ is a function/base predictor chosen from a family of functions H.

Gradient Boosting

- $L^t(y, F^t(x)) := \frac{1}{2}\sum_{k=1}^{n}(y_k - F^t(x_k))^2$
- $\frac{\partial L^t(y, F^t(x))}{\partial F^t(x)} = y - F^t(x)$ the **residual**

Gradient Boosting

- $L^t(y, F^t(x)) := \frac{1}{2}\sum_{k=1}^{n}(y_k - F^t(x_k))^2$
- $\frac{\partial L^t(y,F^t(x))}{\partial F^t(x)} = y - F^t(x)$ the **residual**
- To minimize this loss function, we choose h^t s.t.

$$h^t(x) \approx -\frac{\partial L^t(y,s)}{\partial s}|_{s=F^{t-1}(x)} = y - F^{t-1}(x)$$

Gradient Boosting

- $L^t(y, F^t(x)) := \frac{1}{2}\sum_{k=1}^{n}(y_k - F^t(x_k))^2$
- $\frac{\partial L^t(y, F^t(x))}{\partial F^t(x)} = y - F^t(x)$ the **residual**
- To minimize this loss function, we choose h^t s.t.

$$h^t(x) \approx -\frac{\partial L^t(y,s)}{\partial s}|_{s=F^{t-1}(x)} = y - F^{t-1}(x)$$

- In practice, h^t's are decision trees (CART for Gradient Boosting)

Outline

Problem: Categorical Features

- Categorical Variables has a **discrete** set of values(categories) that are not comparable to each other. E.g. Gender, Major, User ID

Problem: Categorical Features

- Categorical Variables has a **discrete** set of values(categories) that are not comparable to each other. E.g. Gender, Major, User ID
- High dimensionality $\rightarrow$ **target statistics**(TS)

Problem: Categorical Features

- Categorical Variables has a **discrete** set of values(categories) that are not comparable to each other. E.g. Gender, Major, User ID
- High dimensionality $\rightarrow$ **target statistics**(TS)
- A Typical TS is the expected target value conditioned on the category.

$$\hat{x}_k^i \approx \mathbb{E}[y \mid x^i = x_k^i]$$

- This changes a categorical feature to numerical feature that is easier for a decision tree to use.

Solution Attempt: Greedy TS

- For a category i, use the average target value of all samples within this category to estimate TS.
- Add smoothing:

$$\hat{x}_k^i := \frac{\sum_{j=1}^{n} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + ap}{\sum_{j=1}^{n} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}$$

where $a > 0$ is a hyperparameter and p is the prior probability.
$p = \frac{1}{n} \sum_{k=1}^{n} y_k$

Target Leakage & Conditional Shift

- Target leakage: we use y_k (target/the ground-truth label) to compute a feature that is used in a model to predict y.
  - "leaking answers to testtakers" when training
  - BUT, no access to answer when testing
- Conditional shift: Train $\hat{x}_k^i | y_k \Rightarrow$ Test $\hat{x}^i | y$

Target Leakage & Conditional Shift

- Target leakage: we use y_k (target/the ground-truth label) to compute a feature that is used in a model to predict y.
  - "leaking answers to testtakers" when training
  - BUT, no access to answer when testing
- Conditional shift: Train $\hat{x}_k^i | y_k \Rightarrow$ Test $\hat{x}^i | y$
- 2 Problems:
  1. $\hat{x}_k^i$ is a **biased** estimator of $\mathbb{E}[y \mid x^i = x_k^i]$
  2. overfitting, poor generalization

Solutions

- Holdout TS: Partition the training set into 2 parts $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_1$. One for calculating the TS and the other for training.

  Inefficient data usage

Solutions

- Holdout TS: Partition the training set into 2 parts $\mathcal{D} = \mathcal{D}_0 \cup \mathcal{D}_1$. One for calculating the TS and the other for training.

  Inefficient data usage
- **Ordered TS**: No using y_k for $\hat{x}_k^i$
  - (randomly) order the data
  - compute TS only using the preceding examples

Let σ be a random permutation of the indices $1, \ldots, n$.
For the k-th training example, use $\mathcal{D}_k = \{x_j : \sigma(j) < \sigma(k)\}$ for computing TS.
For a testing example, use $\mathcal{D}$.

This ensures $\mathbb{E}[\hat{x}_k^i | y_k] = \mathbb{E}[\hat{x}^i | y]$

Outline

Prediction Shift & Ordered Boosting

- Similar Problem when doing gradient boosting: $h^t(x_k, y_k) \approx -g^t(x_k, y_k)$ is computed using x_k
- Train $g^t(x_k, y_k) \mid x_k \Rightarrow$ Test $g^t(x, y) \mid x$
- Ordered boosting:

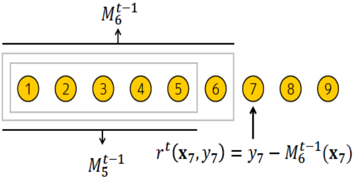

Figure 1: Ordered boosting principle, examples are ordered according to σ.

Algorithm 1: Ordered boosting

input : $\{(\mathbf{x}_k, y_k)\}_{k=1}^n$, I;

$\sigma \leftarrow$ random permutation of $[1, n]$;
$M_i \leftarrow 0$ for $i = 1..n$;
for $t \leftarrow 1$ **to** I **do**
  for $i \leftarrow 1$ **to** n **do**
    $r_i \leftarrow y_i - M_{\sigma(i)-1}(\mathbf{x}_i)$;
  for $i \leftarrow 1$ **to** n **do**
    $\Delta M \leftarrow$ $LearnModel((\mathbf{x}_j, r_j) : \sigma(j) \leq i)$;
    $M_i \leftarrow M_i + \Delta M$;
return M_n

Outline

CatBoost

- CatBoost: using *ordering principle* when (1) dealing with categorical features and (2) computing gradient in boosting

Algorithm 2: Building a tree in CatBoost

input : $M, \{(\mathbf{x}_i, y_i)\}_{i=1}^n, \alpha, L, \{\sigma_i\}_{i=1}^s, Mode$

$grad \leftarrow CalcGradient(L, M, y)$;
$r \leftarrow random(1, s)$; // sample an index for the permutation
if $Mode = Plain$ **then**
  $G \leftarrow (grad_r(i)$ for $i = 1..n)$;
if $Mode = Ordered$ **then**
  $G \leftarrow (grad_{r,\sigma_r(i)-1}(i)$ for $i = 1..n)$;
$T \leftarrow$ empty tree;
foreach *step of top-down procedure* **do** // choose the optimal tree structures
  foreach *candidate split c* **do**
    $T_c \leftarrow$ add split c to T;
    if $Mode = Plain$ **then**
      $\Delta(i) \leftarrow \text{avg}(grad_r(p)$ for $p: leaf_r(p) = leaf_r(i))$ for $i = 1..n$;
    if $Mode = Ordered$ **then**
      $\Delta(i) \leftarrow \text{avg}(grad_{r,\sigma_r(i)-1}(p)$ for $p: leaf_r(p) = leaf_r(i), \sigma_r(p) < \sigma_r(i))$ for $i = 1..n$;
    $loss(T_c) \leftarrow \cos(\Delta, G)$
  $T \leftarrow \arg\min_{T_c}(loss(T_c))$
if $Mode = Plain$ **then**
  $M_{r'}(i) \leftarrow M_{r'}(i) - \alpha \,\text{avg}(grad_{r'}(p)$ for $p: leaf_{r'}(p) = leaf_{r'}(i))$ for $r' = 1..s, i = 1..n$;
if $Mode = Ordered$ **then** // update the predictions
  $M_{r',j}(i) \leftarrow M_{r',j}(i) - \alpha \,\text{avg}(grad_{r',j}(p)$ for $p: leaf_{r'}(p) = leaf_{r'}(i), \sigma_{r'}(p) \le j)$ for $r' = 1..s$, $i = 1..n, j \ge \sigma_{r'}(i) - 1$;
return T, M

Thank You!