CatBoost Algorithm

umberto fontana

December 2022

CatBoost

Unlike other gradient boosting algorithms that require numeric data, CatBoost (categorical boosting) automatically handles categorical features.

One popular technique for dealing with categorical features in boosted trees is *one-hot encoding*. However, in the case of high cardinality features (like, e.g., "userID" feature), such technique leads to an infeasibly large number of features. To address this issue, one can group categories into a limited number of clusters and then apply one-hot encoding. A popular method is to group categories by *target statistics*(TS).

What is Target Statistics?

The main idea behind the target encoder is to encode the categories by replacing them for a measurement of the effect they might have on the target.

On a binary classifier, the simplest way to do that is by calculating the probability $P(t = 1|x = c_i)$ in which t denotes the target, x is the input and c_i is the i-th category. This means we will replace the category c_i for the value of the posterior probability of the target being 1 on the presence of that category.

Let's consider a database of movies. The categories are in the genre feature and the target is binary ($y_i \in \{0, 1\}$). For every single possible category, we need to count how many occurrences there are of the target 0 and the target 1. Then we compute:

$$encoding = (count\_of\_target\_1)/(total\_occurrences) \tag{1}$$

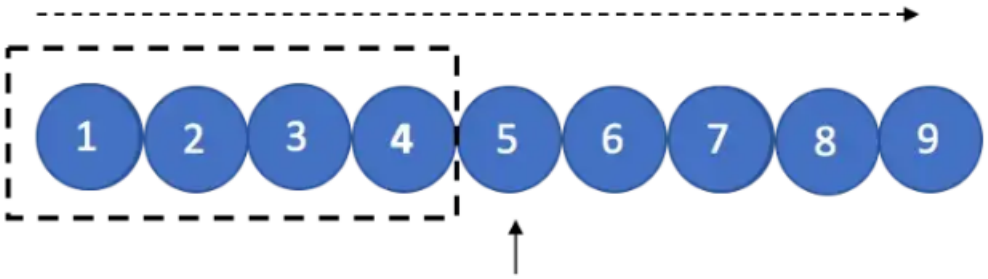
Since the target of interest is the value "1", this probability is actually the mean of the target, given a category. This is a very simple mean target statistic.

Other possible methods are possible, like the Greedy TS, Holdout TS and Leave-one-out TS.

Ordered TS

CatBoost uses a more effective strategy. It relies on the ordering principle and is inspired by online learning algorithms which get training examples sequentially in time. To adapt this idea to standard offline setting, Ordered TS introduce an

artificial time, i.e., a random permutation ω of the training examples. Then, for each example, we use all the available "history" to compute its TS. To simplify, it randomizes the rows and take running average of target label grouped by each category.



Running mean calculation.
Numbers are assigned randomly to each observation. Only 1-4 are used to find encoding for 5

Mathematically, the target estimate of the i-th categorical variable of the kth element of D can be represented as:

$$\hat{x}_k^i = \frac{\sum_{x_j \ inD_k} 1_{\{x_k^i = x_k^j\}} y_j + ap}{\sum_{x_j \in D_k} 1_{\{x_k^i = x_k^j\}} + a} \quad if \quad D_k = \{x_j \quad : \quad \omega(j) < \omega(k)\}$$

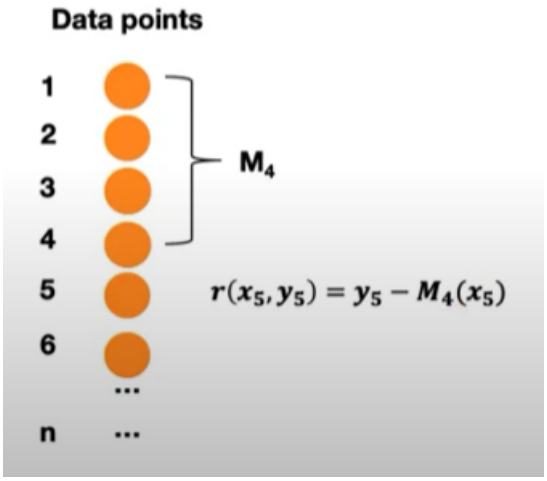
k is the k-th element according to the order we put on D with the random permutationω. The parameters a and p save the equation from underflowing.

When we use ordered statistics to encode categorical variables, the gradients of the loss function L with respect to the function F^{t-1} is also a random variable (because we use the random permutation).

There is a problem in taking group-wise running averages. For the initial few rows in the randomized data, the mean encoding will have high variance as it only saw a few points from the history. But, as it sees more data, the running average starts to get stable. This instability due to randomly chosen initial data points makes the initial part of dataset to have weak estimates. Catboost handles this problem by using more than random permutations. It trains several models simultaneously, each one is trained on its own permutation. All the models share the same tree structure (sequence of splitting features). But leaf values of these models are different. Before building the next tree, CatBoost selects one of the models, which will be used to select tree structure. The structure is selected using this model, and is then used to compute leaf values for all the models. CatBoost also uses ordered target encoding for categorical feature combinations, which is useful for some datasets.

Ordered Boosting

Assume the support model M_i was trained on the first i data points. We basically compute residuals at each data point i using model M_{i-1}



We need to have n separate trees to make this work, but this would not be feasible. IN practical application, we only work with trees at location 2^j where $j = 1, 2, ..., log_2(n)$.

At the start, CatBoost generates $s + 1$ independent random permutations of the training dataset. The permutations $\omega_1, ..., \omega_s$ are used for evaluation of splits that define tree structures (i.e., the internal nodes), while ω_0 serves for choosing the leaf values b_j of the obtained trees. Using only one permutation would increase the variance of the final model predictions.

In CatBoost, base predictors are oblivious decision trees. The term oblivious means that the same splitting criterion is used across an entire level of the tree.

In ordered boosting mode, during the learning process, we maintain the supporting models $M_{r,j}$, where $M_{r,j}(i)$ is the current prediction for the i-th sample based on the first j samples in the permutation ω_r. At each iteration t of the algorithm, we sample a random permutation ω_r from $\{\omega_1, ..., \omega_s\}$ and construct a tree T_t on the basis of it. First, for categorical features, all TS are computed according to this permutation.

Namely, based on $M_{r,j}(i)$, we compute the corresponding gradients $grad_{r,j}(i) = \frac{\partial L(y_i, M_{r,j}(i))}{\partial M_{r,j}(i)}$. At the candidate splits evaluation step, the leaf value $\Delta(i)$ for example i is obtained individually by averaging the gradients $grad_{r,\omega_r(i)-1}$ of the preceiding examples p lying in the same leaf $leaf_r(i)$ the example i belongs to. Note that $leaf_r(i)$ depends on the chosen permutation ω_r because ω_r can influence the values of ordered TS for example i. When the tree structure T_t is built, we use it to boost all the models $M_{r',j}$. One common tree structure T_t is used for all the models, but this tree is added to different $M_{r',j}$, with different sets of leaf values depending on r' and j.

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

https://www.youtube.com/watch?v=668Yj2sgQo4