

Градиентный бустинг

CatBoost vs LightGBM vs XGBoost

Егор Горюнов
yegor_goryunov@vk.com

МТУСИ

14.11.2023

Идея градиентного бустинга

Запишем ансамбль, как линейную комбинацию базовых алгоритмов:

$$f_T(x) = \sum_{t=1}^T \alpha_t b_t(x), \quad x \in X$$

Критерий качества с заданной гладкой функцией потерь $L(f(x), y)$:

$$Q(f, X^N) = \sum_{i=1}^N L(f(x_i), y_i) \rightarrow \min_f$$

Можем получить рекуррентную формулу для ансамбля:

$$f_T(x) = f_{T-1}(x) + \alpha_T b_T(x)$$

Пусть $f_{T,i} = f_T(x_i)$, тогда $f_T = (f_{T,i})_{i=1}^N$ - вектор значений ансамбля по каждому объекту выборки.

Идея градиентного бустинга

Будем решать задачу оптимизации критерия качества методом градиентного спуска:

$$f_{T,i} = f_{T-1,i} - \alpha g_i, \quad g_i = L'(f_{T-1,i}, y_i)$$

Это соответствует рекуррентной формуле для нашего ансамбля

$f_{T,i} = f_{T-1,i} + \alpha b(x_i)$, если приблизить антиградиент функции потерь новой базовой моделью.

Идея: будем искать такой базовый алгоритм $b(x)$, чтобы он хорошо приближал вектор антиградиента в каждой точке выборки:

$$b_T := \arg \min_b \sum_{i=1}^N \text{Loss}(b(x_i), -g_i)$$

В качестве меры отклонения можно использовать, например, квадрат ошибки:

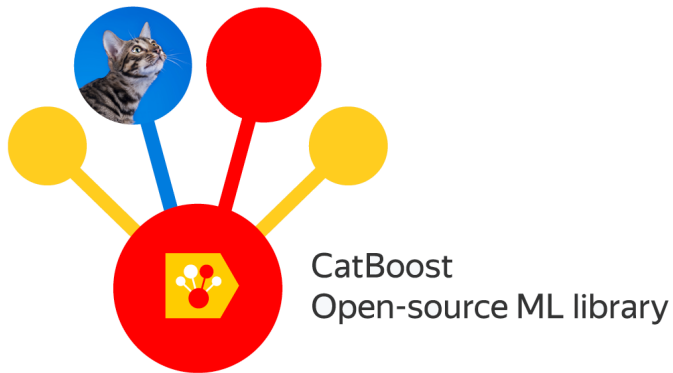
$$b_T := \arg \min_b \sum_{i=1}^N (b(x_i) + g_i)^2$$

При этом α можно брать, например, по методу *наискорейшего спуска*.

Некоторые особенности градиентного бустинга

1. Используют стохастическую оптимизацию:
Для обучения очередной базовой модели *сэмплируют* набор объектов из выборки и обучают по нему.
2. Используют *learning rate*:
Для избежания переобучения вклад каждой базовой модели во всю композицию искусственно снижается. Это позволяет брать больше деревьев в композицию и обучаться более плавно и точно.
3. Используют деревья решений (CART) в качестве базовых моделей:
Деревья решений являются "*слабыми*" моделями, которые сильно зависят от данных. Именно такие модели лучше всего работают в ансамблях благодаря разнообразию в ответах базовых моделей.

Где CatBoost?



Проблема смещённости бустинга

Переобучение на градиентах:

Градиенты $g_i = L'(f_{T-1}(x_i), y_i)$ вычисляются на тех же данных, на которых обучался ансамбль f_{T-1} .

Отсюда получаем смещённые оценки градиентов и переобучение.

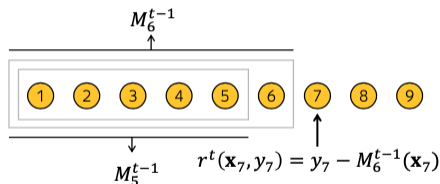
Идеи решения:

- ▶ Разделить выборку пополам
- ▶ k-fold
- ▶ leave-one-out
- ▶ ordered boosting (CatBoost)

Ordered Boosting

Суть подхода:

- ▶ Генерировать несколько случайных перестановок выборки, из них для каждой базовой модели выбирать случайную.
- ▶ Строить последовательно расширяющиеся обучающие подвыборки
- ▶ Вычислять g_i по модели f_{T-1} , которая не обучалась на x_i

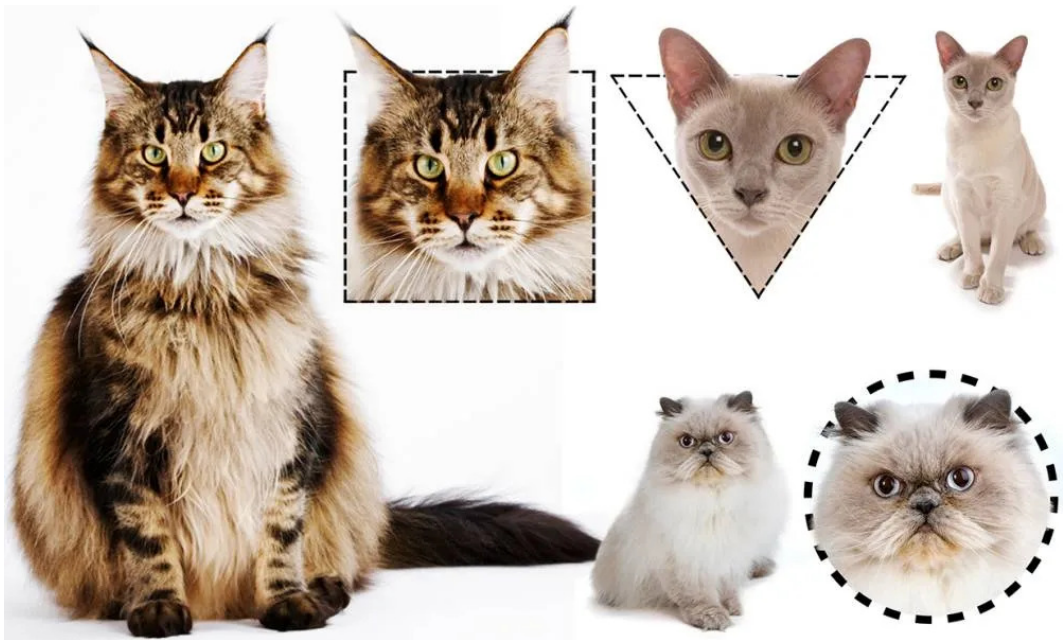


Алгоритм CatBoost с Ordered Boosting

input : $\{(x_i, y_i)\}_{i=1}^n, I, \alpha, L, s$

- 1 $\sigma_r \leftarrow$ random permutation of $[1, n]$ for $r = 0 \dots s$;
- 2 $M_0(i) \leftarrow 0$ for $i = 1 \dots n$;
- 3 **for** $j \leftarrow 1$ **to** $\lceil \log_2 n \rceil$ **do**
- 4 $M_{r,j}(i) \leftarrow 0$ for $r = 1 \dots s, i = 1 \dots 2^{j+1}$;
- 5 **for** $t \leftarrow 1$ **to** I **do**
- 6 $T_t, \{M_r\}_{r=1}^s \leftarrow \text{BuildTree}(\{M_r\}_{r=1}^s, \{(x_i, y_i)\}_{i=1}^n, \alpha, L, \{\sigma_i\}_{i=1}^s)$;
- 7 $\text{leaf}_0(i) \leftarrow \text{GetLeaf}(x_i, T_t, \sigma_0)$ for $i = 1 \dots n$;
- 8 $\text{grad}_0 \leftarrow \text{CalcGradient}(L, M_0, y)$;
- 9 **foreach** $\text{leaf } j$ **in** T_t **do**
- 10 $b_j^t \leftarrow -\text{avg}(\text{grad}_0(i) \text{ for } i : \text{leaf}_0(i) = j)$;
- 11 **for** $i = 1 \dots n$ **do**
- 12 $M_0(i) \leftarrow M_0(i) + \alpha b_{\text{leaf}_0(i)}^t$
- 13 **return** $F(x) = \sum_{t=1}^I \sum_j \alpha b_j^t \mathbb{1}_{\text{GetLeaf}(x, T_t, \text{ApplyMode})=j}$;

CatBoost и при чём тут котики



Кодирование категориальных признаков

- ▶ One-Hot Encoding
- ▶ Label Encoding
- ▶ Hash Encoding
- ▶ Frequency Encoding
- ▶ Target Encoding

$$\tilde{f}(x) = \frac{\sum_{i=1}^{\ell} [f(x_i) = f(x)] y_i + \gamma p}{\sum_{i=1}^{\ell} [f(x_i) = f(x)] + \gamma}$$

- ▶ **Ordered Target Encoding**

$$\hat{x}_k^i = \frac{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} \cdot y_j + a p}{\sum_{\mathbf{x}_j \in \mathcal{D}_k} \mathbb{1}_{\{x_j^i = x_k^i\}} + a}.$$

Визуализация в CatBoost

```
In [11]: model.fit(
    X_train, y_train,
    cat_features=categorical_features_indices,
    eval_set=(X_validation, y_validation),
    # verbose=True, # you can uncomment this for text output
    plot=True
)
```

× ☒ --- Learn ☒ — Test

Logloss Accuracy

☒ current 19s

curr --- 0.2555652... — 0.3988282... 498

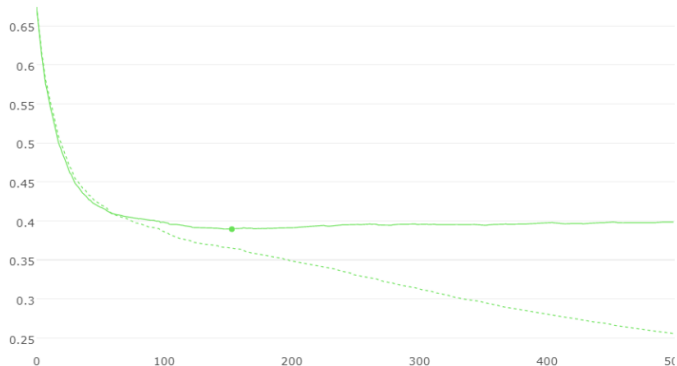
best — 0.3894004... 153

☐ Click Mode

☐ Logarithm

☐ Smooth

0.5



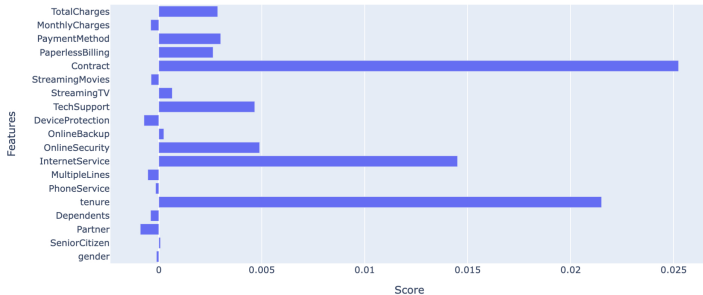
Текстовые признаки

1. Токенизация
2. Векторизация (Мешок слов)
 - 2.1 Частотная
 - 2.2 NaiveBayes
 - 2.3 BM25

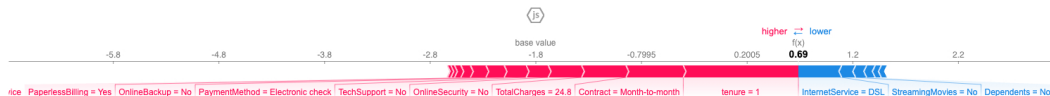
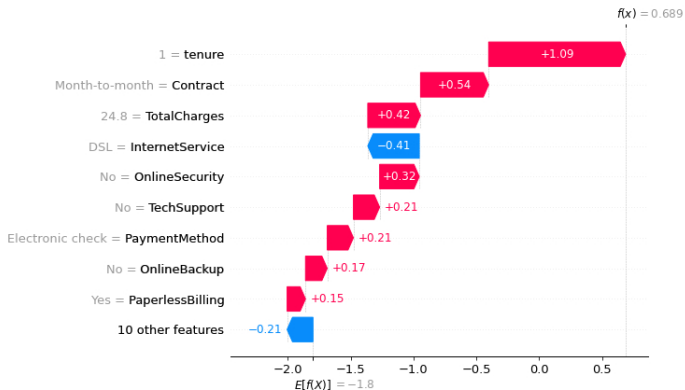
Отбор признаков

- ▶ PredictionValuesChange
- ▶ LossFunctionChange
- ▶ InternalFeatureImportance
- ▶ **SHAP**

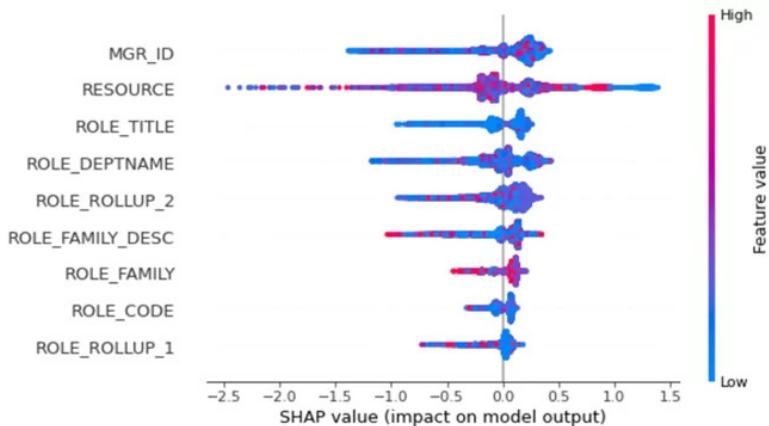
Feature Importance using LossFunctionChange technique



SHAP



SHAP



Сравнение точности

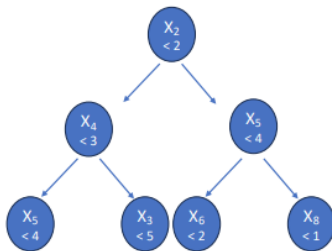
	CatBoost		LightGBM		XGBoost		H2O	
	Tuned	Default	Tuned	Default	Tuned	Default	Tuned	Default
Adult	0.26974	0.27298 +1.21%	0.27602 +2.33%	0.28716 +6.46%	0.27542 +2.11%	0.28009 +3.84%	0.27510 +1.99%	0.27607 +2.35%
Amazon	0.13772	0.13811 +0.29%	0.16360 +18.80%	0.16716 +21.38%	0.16327 +18.56%	0.16536 +20.07%	0.16264 +18.10%	0.16950 +23.08%
Click prediction	0.39090	0.39112 +0.06%	0.39633 +1.39%	0.39749 +1.69%	0.39624 +1.37%	0.39764 +1.73%	0.39759 +1.72%	0.39785 +1.78%
KDD appetency	0.07151	0.07138 -0.19%	0.07179 +0.40%	0.07482 +4.63%	0.07176 +0.35%	0.07466 +4.41%	0.07246 +1.33%	0.07355 +2.86%
KDD churn	0.23129	0.23193 +0.28%	0.23205 +0.33%	0.23565 +1.89%	0.23312 +0.80%	0.23369 +1.04%	0.23275 +0.64%	0.23287 +0.69%
KDD internet	0.20875	0.22021 +5.49%	0.22315 +6.90%	0.23627 +13.19%	0.22532 +7.94%	0.23468 +12.43%	0.22209 +6.40%	0.24023 +15.09%
KDD upselling	0.16613	0.16674 +0.37%	0.16682 +0.42%	0.17107 +2.98%	0.16632 +0.12%	0.16873 +1.57%	0.16824 +1.28%	0.16981 +2.22%
KDD 98	0.19467	0.19479 +0.07%	0.19576 +0.56%	0.19837 +1.91%	0.19568 +0.52%	0.19795 +1.69%	0.19539 +0.37%	0.19607 +0.72%

Сравнение скорости

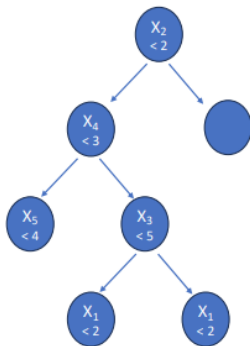
	CatBoost	XGBoost	LightGBM
CPU (Xeon E5-2660v4)	527 sec	4339 sec	1146 sec
GTX 1080Ti (11GB)	18 sec	890 sec	110 sec

Dataset Epsilon (400K samples, 2000 features). Parameters: 128 bins, 64 leafs, 400 iterations.

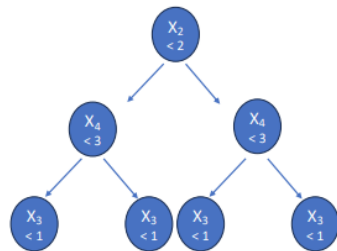
Способы построения деревьев



XGBoost



LightGBM



CatBoost

Сравнительная таблица

	CatBoost	LightGBM	XGBoost
Parameters to <u>tune</u>	<i>iterations</i> : number of trees <i>depth</i> : depth of tree <i>min_data_in_leaf</i> : control depth of tree	<i>num_leaves</i> : value should be less than $2^{\text{max_depth}}$ <i>min_data_in_leaf</i> : control depth of tree <i>max_depth</i> : depth of tree	<i>n_estimators</i> : number of trees <i>max_depth</i> : depth of tree <i>min_child_weight</i> : control depth of tree
Parameters for <u>better accuracy</u>		<i>max_bin</i> : maximum number of bins feature values will be bucketed in <i>num_leaves</i> Use bigger training data	
Parameters for <u>faster speed</u>	<i>subsample</i> : fraction of number of instances used in a tree <i>rsm</i> : random subspace method; fraction of number of features used in a split selection <i>iterations</i> <i>sampling_frequency</i> : frequency to sample weights and objects when building trees	<i>feature_fraction</i> : fraction of number of features used in a tree <i>bagging_fraction</i> : fraction of number of instances used in a tree <i>bagging_freq</i> : frequency for bagging <i>max_bin</i> <i>save_binary</i> : indicator to save dataset to binary file Use parallel learning	<i>colsample_bytree</i> : fraction of number of features used in a tree <i>subsample</i> : fraction of number of instances used in a tree <i>n_estimators</i>
Parameters to <u>control overfitting</u>	<i>early_stopping_rounds</i> : stop training after specified number of iterations since iteration with optimal metric value <i>od_type</i> : type of overfitting detector <i>learning_rate</i> : learning rate for reducing gradient step <i>depth</i> <i>l2_leaf_reg</i> : regularization parameter	<i>max_bin</i> <i>num_leaves</i> <i>max_depth</i> <i>bagging_fraction</i> <i>bagging_freq</i> <i>feature_fraction</i> <i>lambda_l1 / lambda_l2 / min_gain_to_split</i> Use bigger training data Regularization	<i>learning_rate</i> <i>gamma</i> : regularization parameter, higher gamma for more regularization <i>max_depth</i> <i>min_child_weight</i> <i>subsample</i>

Резюме

- ▶ Используем градиентный бустинг, когда имеем разнородные данные.
- ▶ Градиентный бустинг позволяет брать любые гладкие функции потерь.
- ▶ Ordered Boosting позволяет избежать смещения и переобучения.
- ▶ CatBoost (и др.) решает задачи: регрессии, классификации, ранжирования.
- ▶ Используем CatBoost, когда много категориальных признаков.
- ▶ В CatBoost можно работать с текстами (Bag of Words).
- ▶ CatBoost позволяет очень эффективно обучаться на GPU.
- ▶ CatBoost позволяет отлавливать начало переобучения.
- ▶ Для достижения лучшего результата стоит подбирать количество базовых моделей и learning rate.
- ▶ Если точка переобучения слишком рано (Overfitting), чтоит снизить learning rate. Если в самом конце (Underfitting) - повысить learning rate.

Ссылки

- ▶ catboost.ai - Документация CatBoost
- ▶ github.com/catboost/tutorials - CatBoost Tutorials
- ▶ <https://arxiv.org/abs/1706.09516> - CatBoost: unbiased boosting with categorical features.
- ▶ <https://arxiv.org/abs/2307.07771> - Enhanced Gradient Boosting for Zero-Inflated Insurance Claims and Comparative Analysis of CatBoost, XGBoost, and LightGBM.
- ▶ <https://towardsdatascience.com/catboost-vs-lightgbm-vs-xgboost-c80f40662924> - CatBoost vs. LightGBM vs. XGBoost.
- ▶ <https://ods.ai/hubs/catboost> - CatBoost ODS hub.



Обсудим?