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:

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

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

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

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 хороши приближал вектор антиградиента в каждой точке выборки:

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

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*.

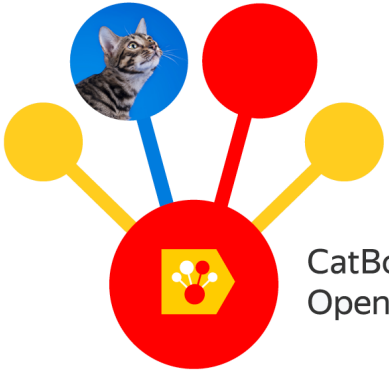
Некоторые особенности градиентного бустинга

1. Используют стохастическескую 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 искуственно 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.

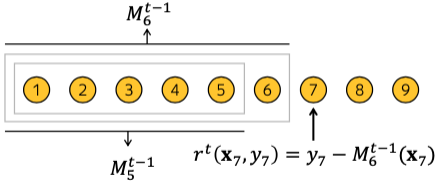
Идеи решения:

- 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



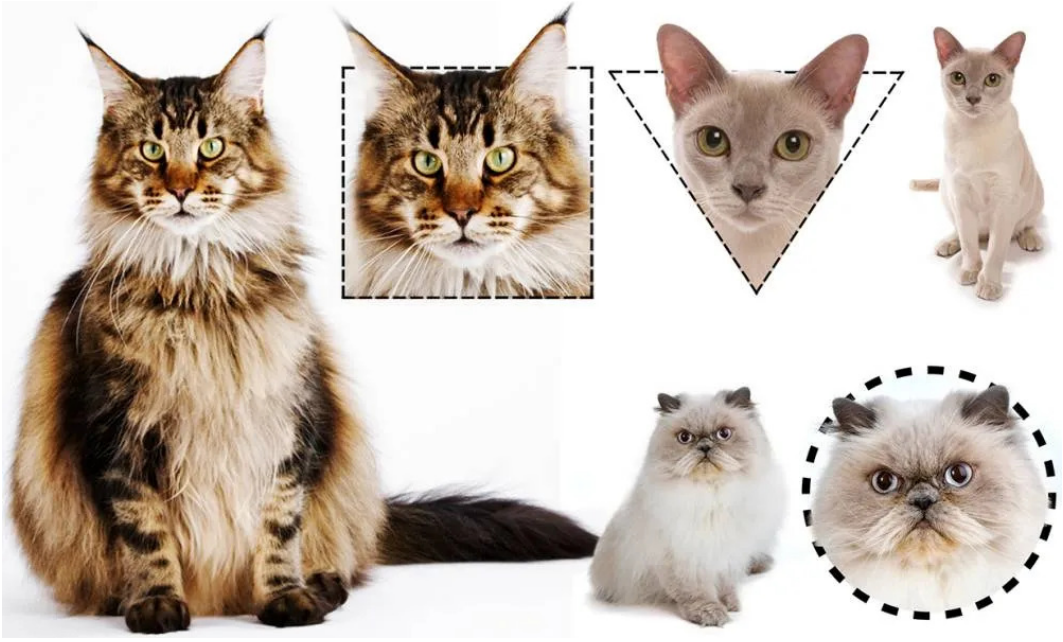
Алгоритм Ordered Boosting

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

CatBoost и при чём тут котики



Кодирование категориальных признаков

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

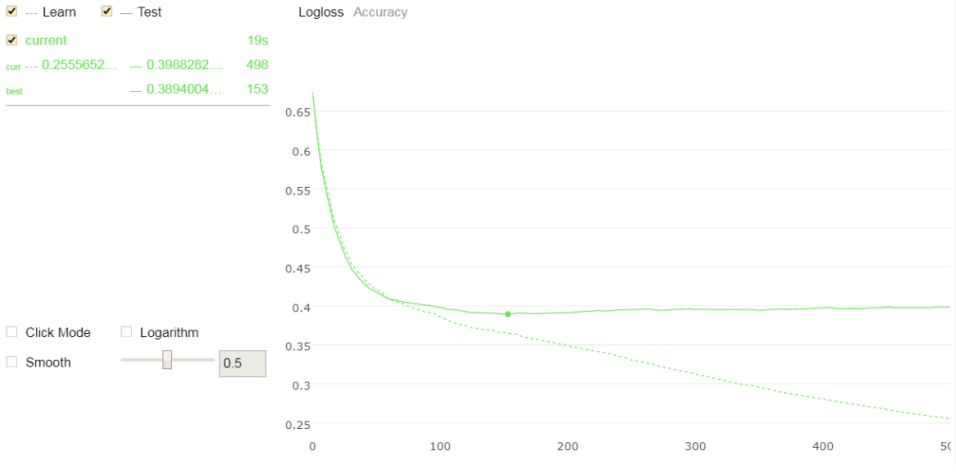
$$\tilde{f}(x) = \frac{\sum_{i=1}^{\ell}[f(x_i) = f(x)]y_i + \gamma p}{\sum_{i=1}^{\ell}[f(x_i) = f(x)] + \gamma}$$

- **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
)
```

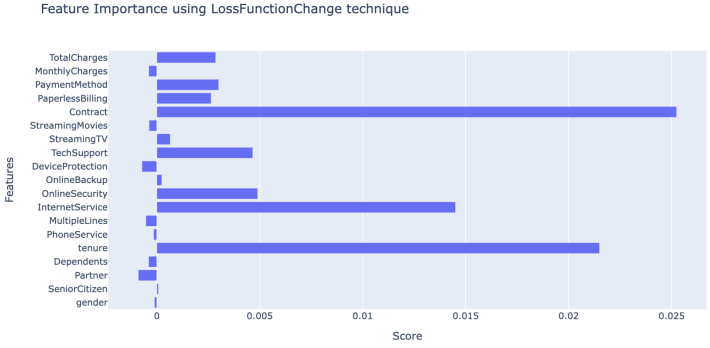


Текстовые признаки

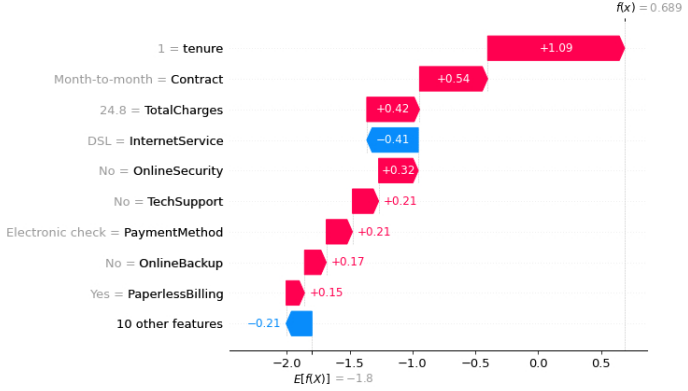
1. Токенизация
2. Векторизация (Мешок слов)
   - 2.1 Частотная
   - 2.2 NaiveBayes
   - 2.3 BM25

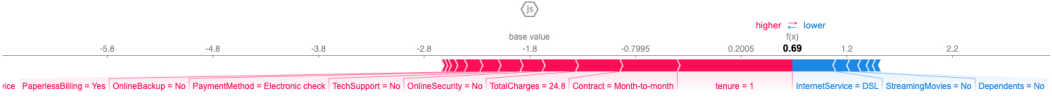
Отбор признаков

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

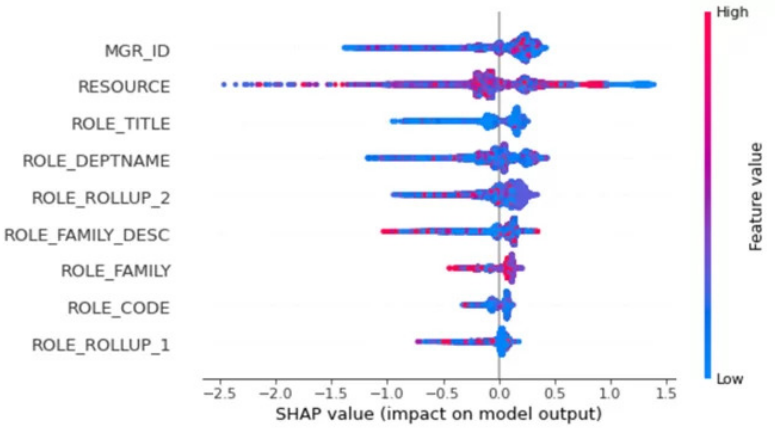


SHAP


f(x) = 0.689
1 = tenure
+1.09
Month-to-month = Contract
+0.54
24.8 = TotalCharges
+0.42
DSL = InternetService
−0.41
No = OnlineSecurity
+0.32
No = TechSupport
+0.21
Electronic check = PaymentMethod
+0.21
No = OnlineBackup
+0.17
Yes = PaperlessBilling
+0.15
10 other features
−0.21
−2.0
−1.5
−1.0
−0.5
0.0
0.5
E[f(X)] = −1.8


higher ⇄ lower
base value
f(x)
-5.8
-4.8
-3.8
-2.8
-1.8
-0.7995
0.2005
0.69
1.2
2.2
rice
PaperlessBilling = Yes
OnlineBackup = No
PaymentMethod = Electronic check
TechSupport = No
OnlineSecurity = No
TotalCharges = 24.8
Contract = Month-to-month
tenure = 1
InternetService = DSL
StreamingMovies = No
Dependents = No

SHAP


MGR_ID
RESOURCE
ROLE_TITLE
ROLE_DEPTNAME
ROLE_ROLLUP_2
ROLE_FAMILY_DESC
ROLE_FAMILY
ROLE_CODE
ROLE_ROLLUP_1
−2.5
−2.0
−1.5
−1.0
−0.5
0.0
0.5
1.0
1.5
SHAP value (impact on model output)
High
Feature value
Low

Сравнение точности

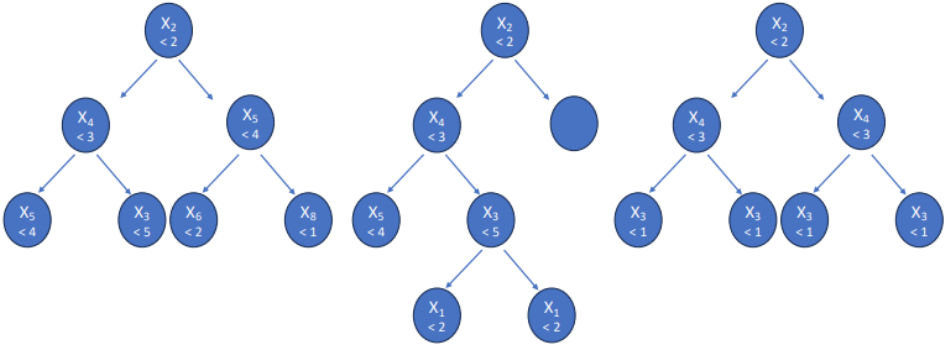
	CatBoost		LightGBM		XGBoost		H2O	
	Tuned	Default	Tuned	Default	Tuned	Default	Tuned	Default
Adult	**0.26974**	0.27298 +1.21%	0.27602 +2.33%	0.28716 +6.46%	0.27542 +2.11%	0.28009 +3.84%	0.27510 +1.99%	0.27607 +2.35%
Amazon	**0.13772**	0.13811 +0.29%	0.16360 +18.80%	0.16716 +21.38%	0.16327 +18.56%	0.16536 +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 tune	*iterations*: number of trees *depth*: depth of tree *min_data_in_leaf*: control depth of tree	*num_leaves*: value should be less than 2^max_depth *min_data_in_leaf*: control depth of tree *max_depth*: depth of tree	*n_estimators*: number of trees *max_depth*: depth of tree *min_child_weight*: control depth of tree
Parameters for better accuracy		*max_bin*: maximum number of bins feature values will be bucketed in *num_leaves* Use bigger training data	
Parameters for faster speed	*subsample*: fraction of number of instances used in a tree *rsm*: random subspace method; fraction of number of features used in a split selection *iterations* *sampling_frequency*: frequency to sample weights and objects when building trees	*feature_fraction*: fraction of number of features used in a tree *bagging_fraction*: fraction of number of instances used in a tree *bagging_freq*: frequency for bagging *max_bin* *save_binary*: indicator to save dataset to binary file Use parallel learning	*colsample_bytree*: fraction of number of features used in a tree *subsample*: fraction of number of instances used in a tree *n_estimators*
Parameters to control overfitting	*early_stopping_rounds*: stop training after specified number of iterations since iteration with optimal metric value *od_type*: type of overfitting detector *learning_rate*: learning rate for reducing gradient step *depth* *l2_leaf_reg*: regularization parameter	*max_bin* *num_leaves* *max_depth* *bagging_fraction* *bagging_freq* *feature_fraction* *lambda_l1 / lambda_l2 / min_gain_to_split* Use bigger training data Regularization	*learning_rate* *gamma*: regularization parameter, higher gamma for more regularization *max_depth* *min_child_weight* *subsample*

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?