Yandex

Yandex

CatBoost

Fast Open-Source Gradient Boosting Library For GPU

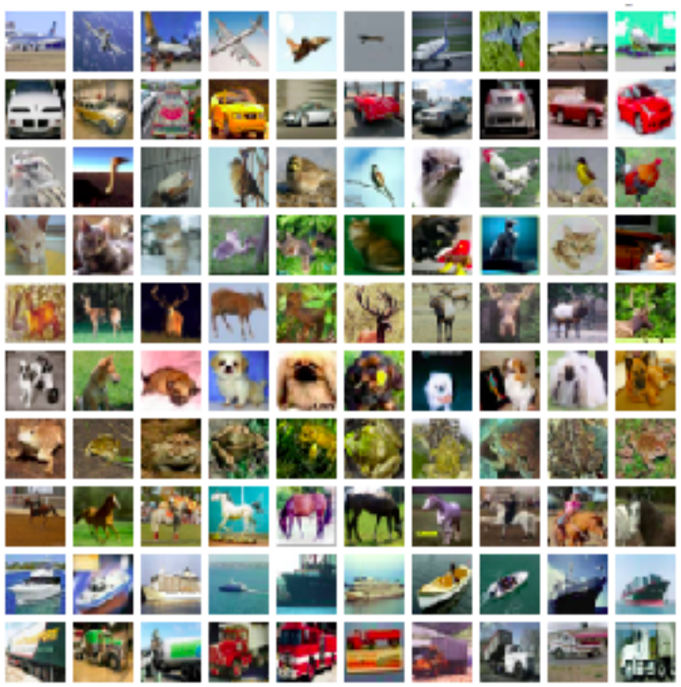
Vasily Ershov, Software Developer

Content

- What is our place in ML world?
- Why do we use GPU?
- How to use the library efficiently:
  - Functionality
  - GPU-specific tips

CatBoost place in ML world

Different input data => different tools to use



Images => CNN

DNA

Text

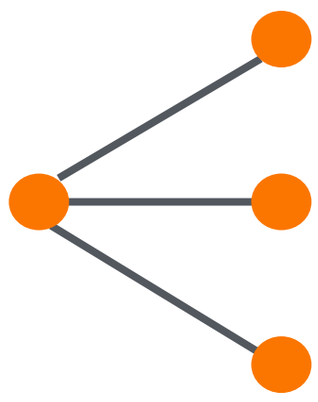
=> RNN

CatBoost place in ML world

Different input data => different tools to use

Ordered (numerical) features

Track length



2 minutes

5 minutes

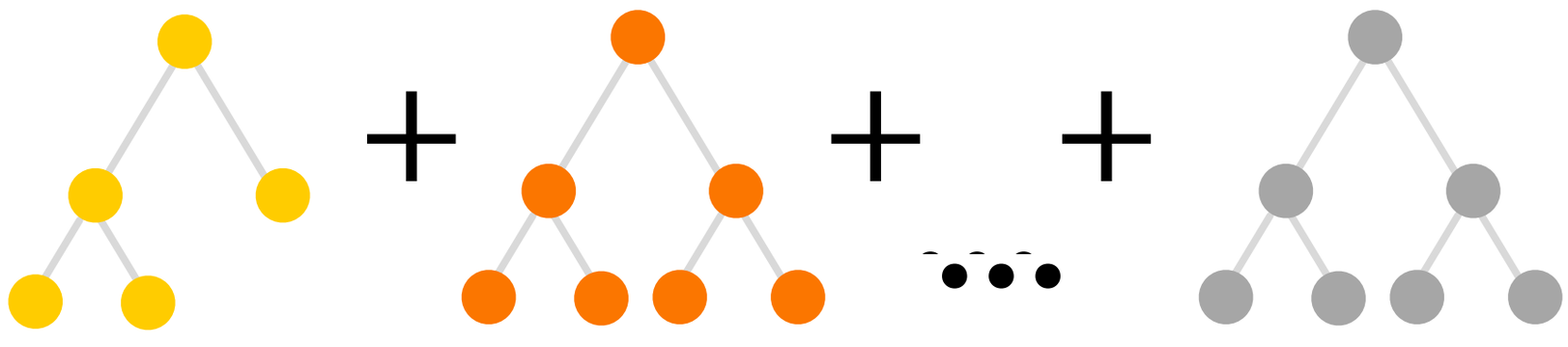
15 minutes

Gradient Boosting on Decision Trees

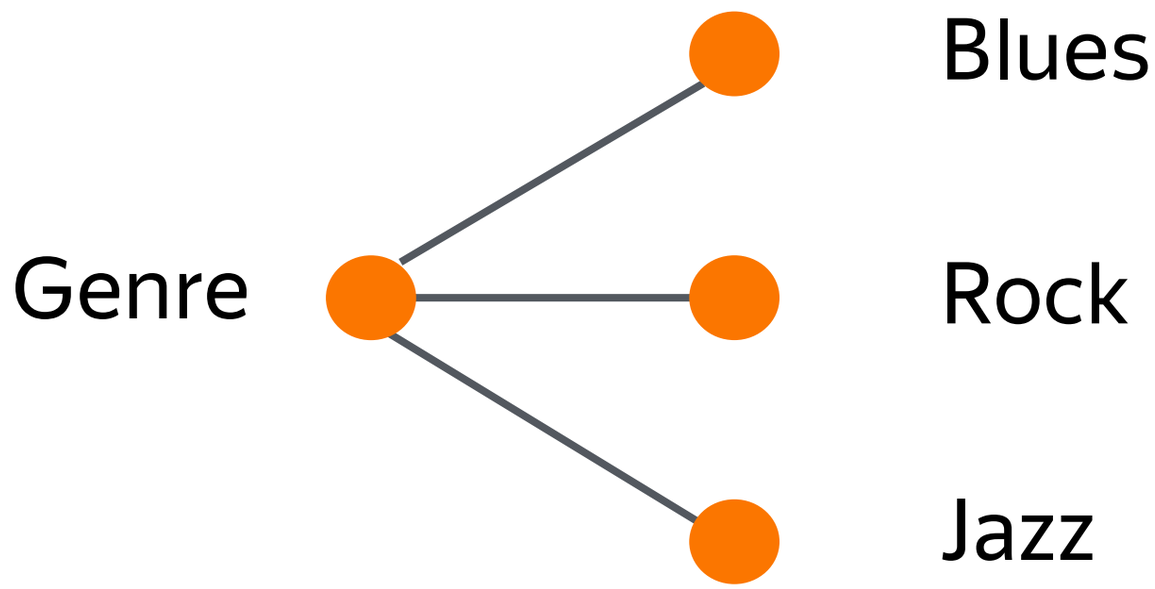
XGBoot

LigthGBM

CatBoost

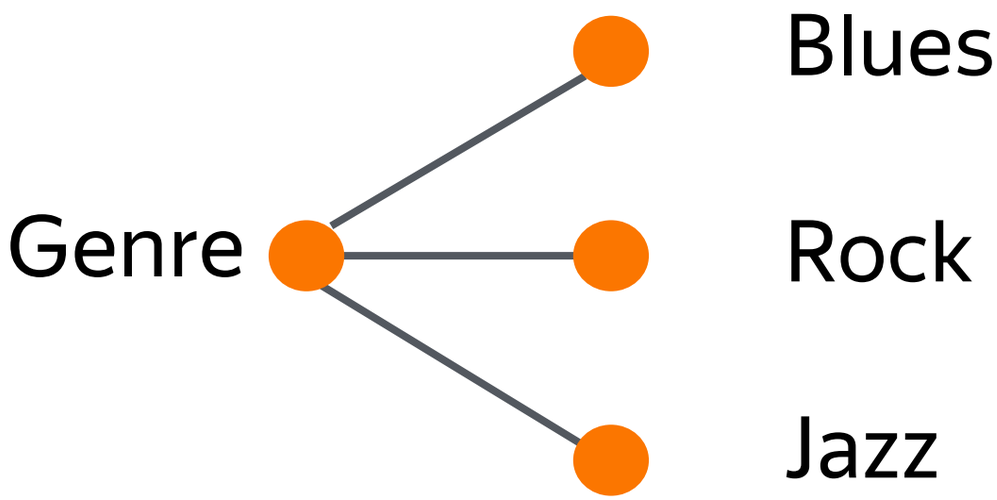


Categorical features



CatBoost place in ML world

Categorical features: before CatBoost



Use linear models

Manually convert to numeric and use boosting

- one-hot-encoding

  (useless for high-cardinality)

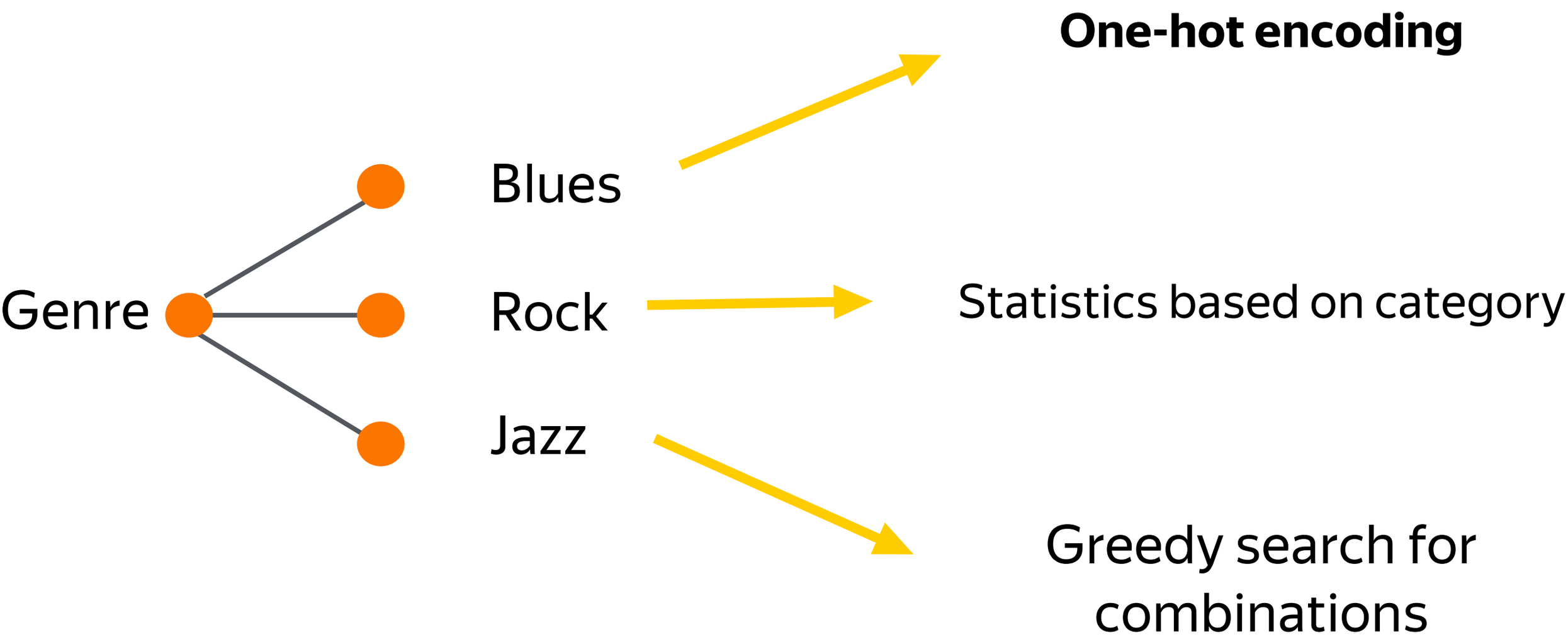
- Feature engineering

  (including linear models)

CatBoost place in ML world

Categorical features: with CatBoost

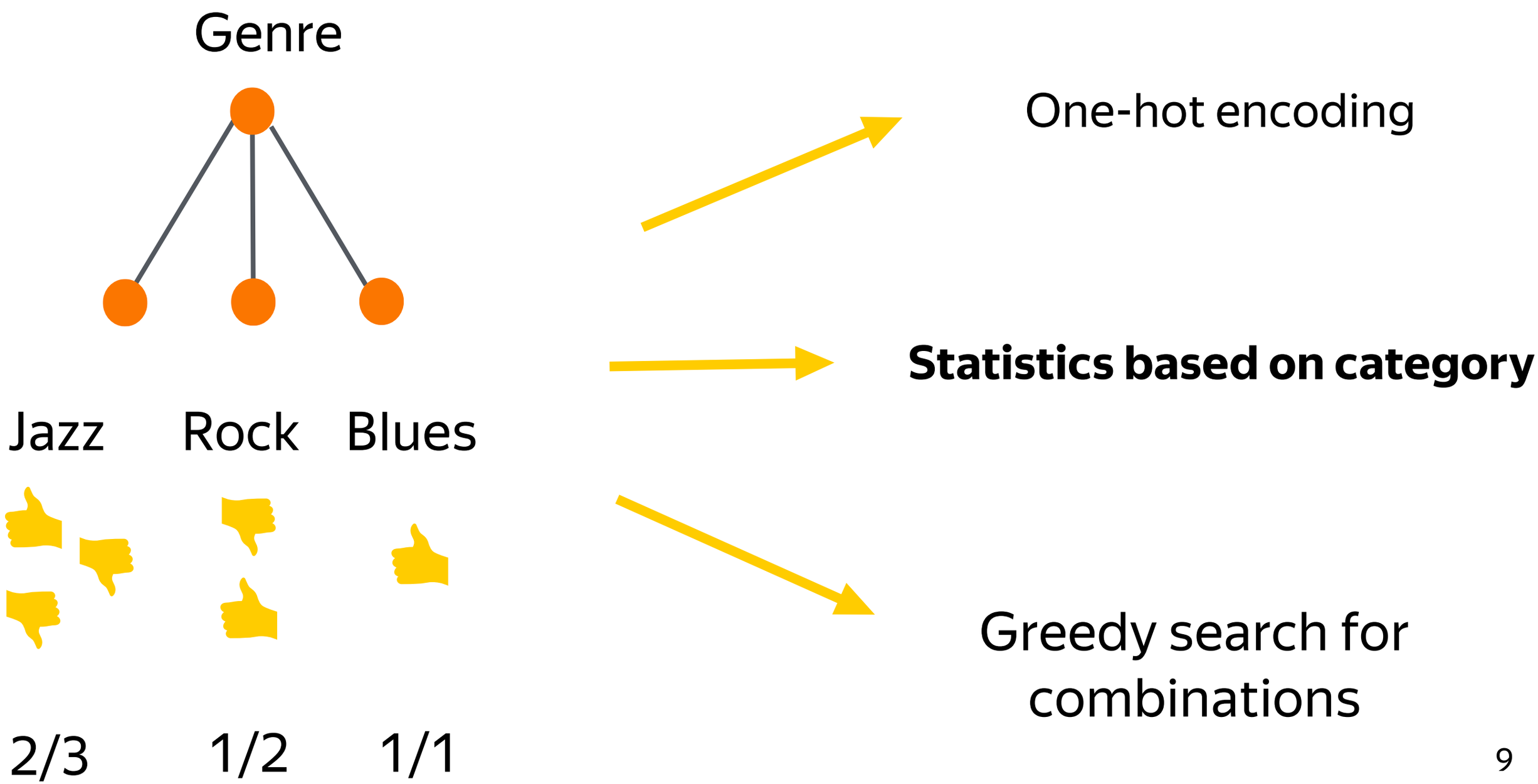
Boosting + out-of-box categorical features



CatBoost place in ML world

Categorical features: with CatBoost

Boosting + out-of-box categorical features

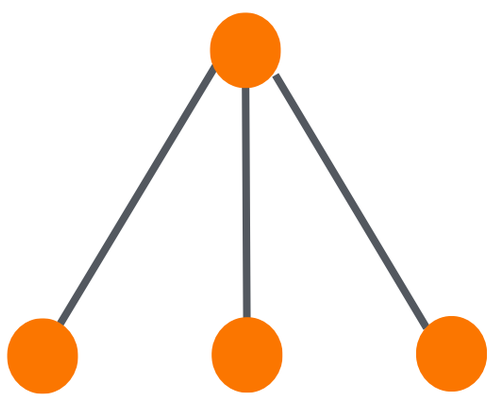


CatBoost place in ML world

Categorical features: with CatBoost

Boosting + out-of-box categorical features

Genre



Jazz Rock Blues

2/3 1/2 1/1

One-hot encoding

Statistics based on category

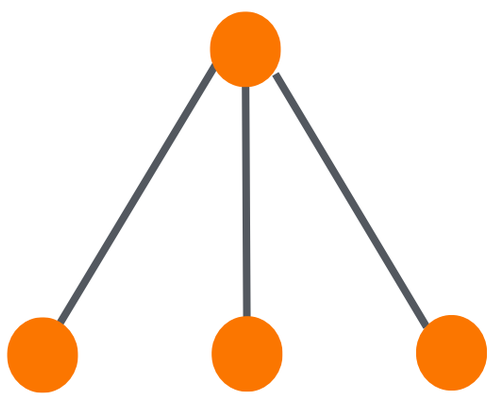
Greedy search for combinations

CatBoost place in ML world

Categorical features: with CatBoost

Boosting + out-of-box categorical features

Genre



Jazz Rock Blues

$$\frac{1+\alpha}{1+\alpha+\beta}$$

One-hot encoding

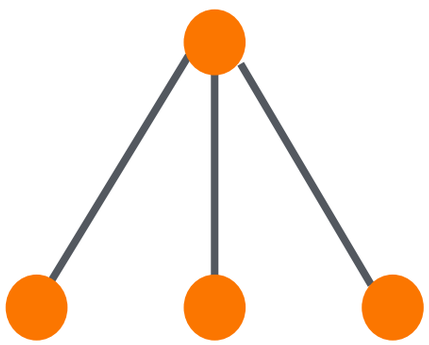
Statistics based on category

Greedy search for combinations

CatBoost place in ML world

Categorical features: with CatBoost

Boosting + out-of-box categorical features

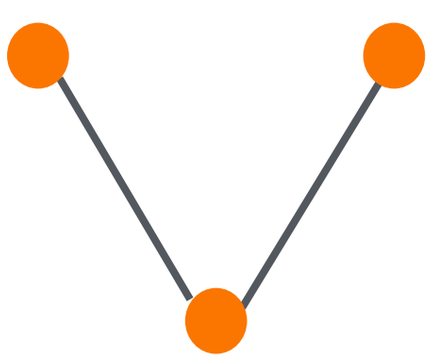


Jazz Rock Blues

One-hot encoding

Statistics based on category

Alice Bob



Greedy search for combinations

CatBoost Quality

	CatBoost	LightGBM		XGBoost		H2O	
Adult	0.269741	0.276018	+ 2.33 %	0.275423	+ 2.11%	0.275104	+ 1.99%
Amazon	0.137720	0.163600	+ 18.79 %	0.163271	+ 18.55%	0.162641	+ 18.09%
Appet	0.071511	0.071795	+ 0.40 %	0.071760	+ 0.35%	0.072457	+ 1.32%
Click	0.390902	0.396328	+ 1.39 %	0.396242	+ 1.37%	0.397595	+ 1.71%
Internet	0.208748	0.223154	+ 6.90 %	0.225323	+ 7.94%	0.222091	+ 6.39%
Kdd98	0.194668	0.195759	+ 0.56 %	0.195677	+ 0.52%	0.195395	+ 0.37%
Kddchurn	0.231289	0.232049	+ 0.33 %	0.233123	+ 0.79%	0.232752	+ 0.63%
Kick	0.284793	0.295660	+ 3.82 %	0.294647	+ 3.46%	0.294814	+ 3.52%

Look for experiments description on our GitHub

Why GPUs?

Boosting in industry

- More data => ~~more quality~~ more money
- More trees => ~~more quality~~ more money
- Faster learning => ~~more experiments~~ more money

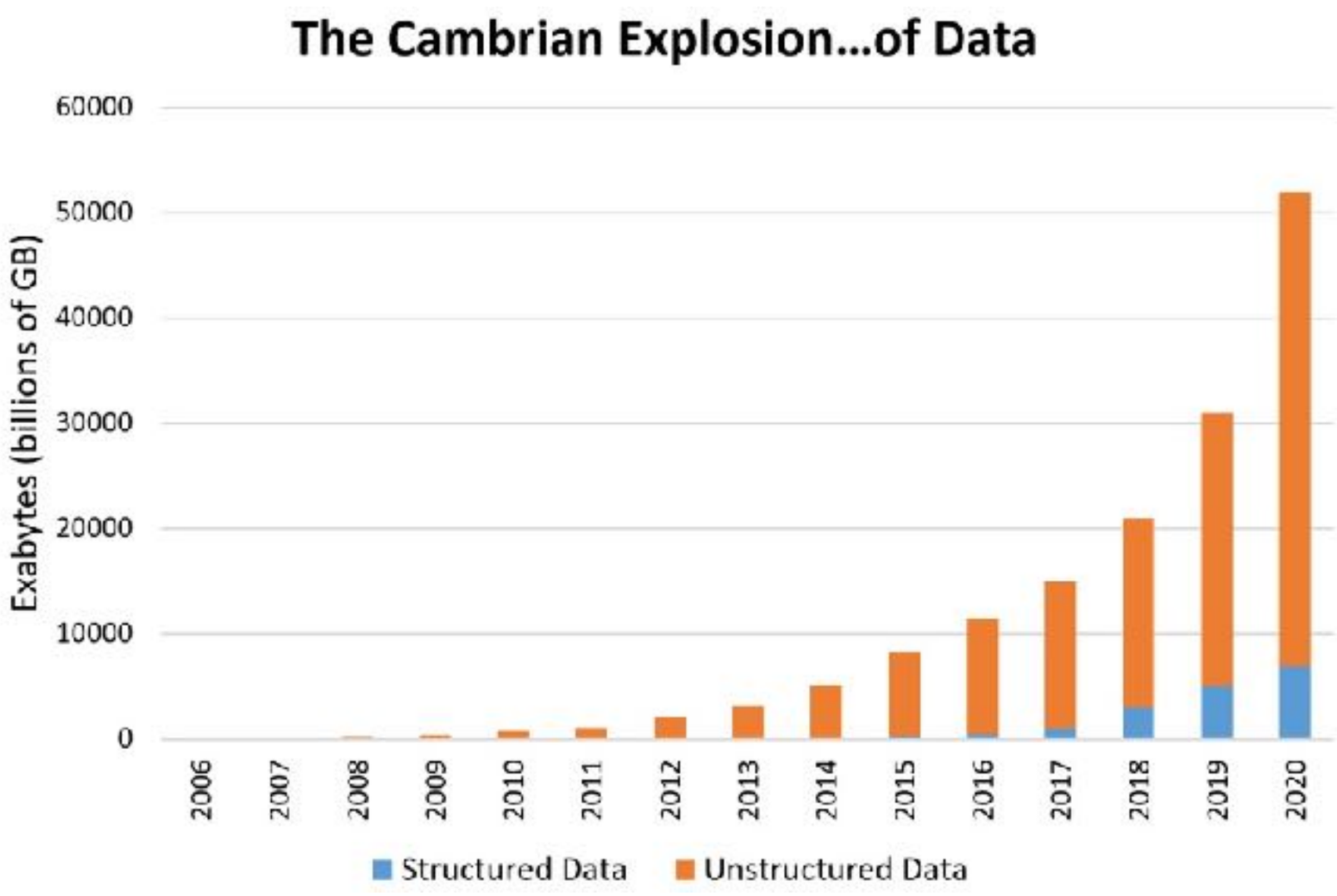
DataSet sizes

Modern research and production:

- Yandex: 100GB is small
- 8 GPU, 24 GB per each, for production models

Classical research and competitions:

- Higgs, 28 features, 11M samples, 7GB
- 500MB GPU Memory, 1 GPU



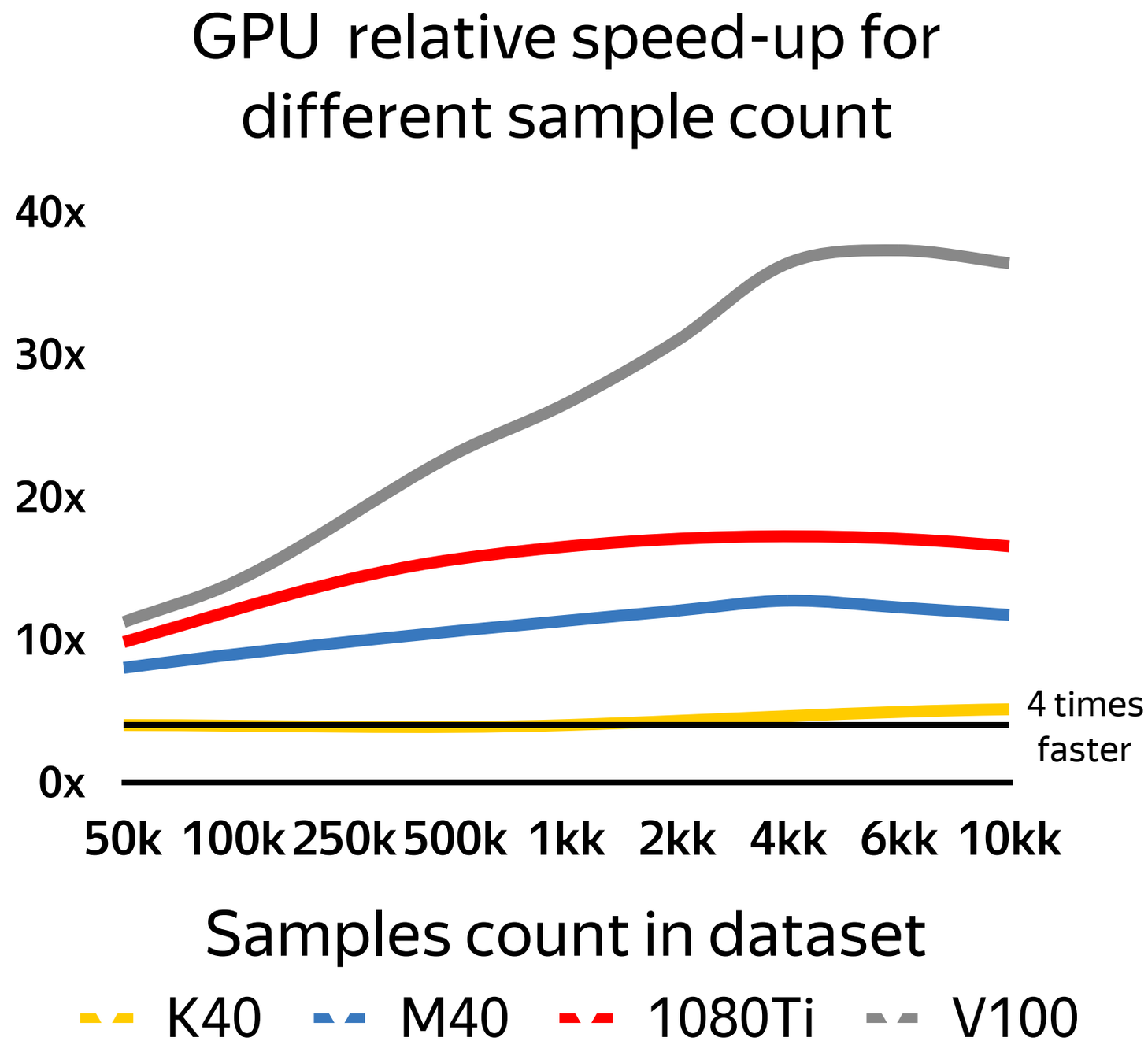
GPU vs CPU

Hardware

- Dual-Socket Intel Xeon E5-2660v4 as baseline
- Several modern GPU as competitors

Dataset

- ≈800 features
- Sample count on x-axis



Comparison with competitors

Parameters

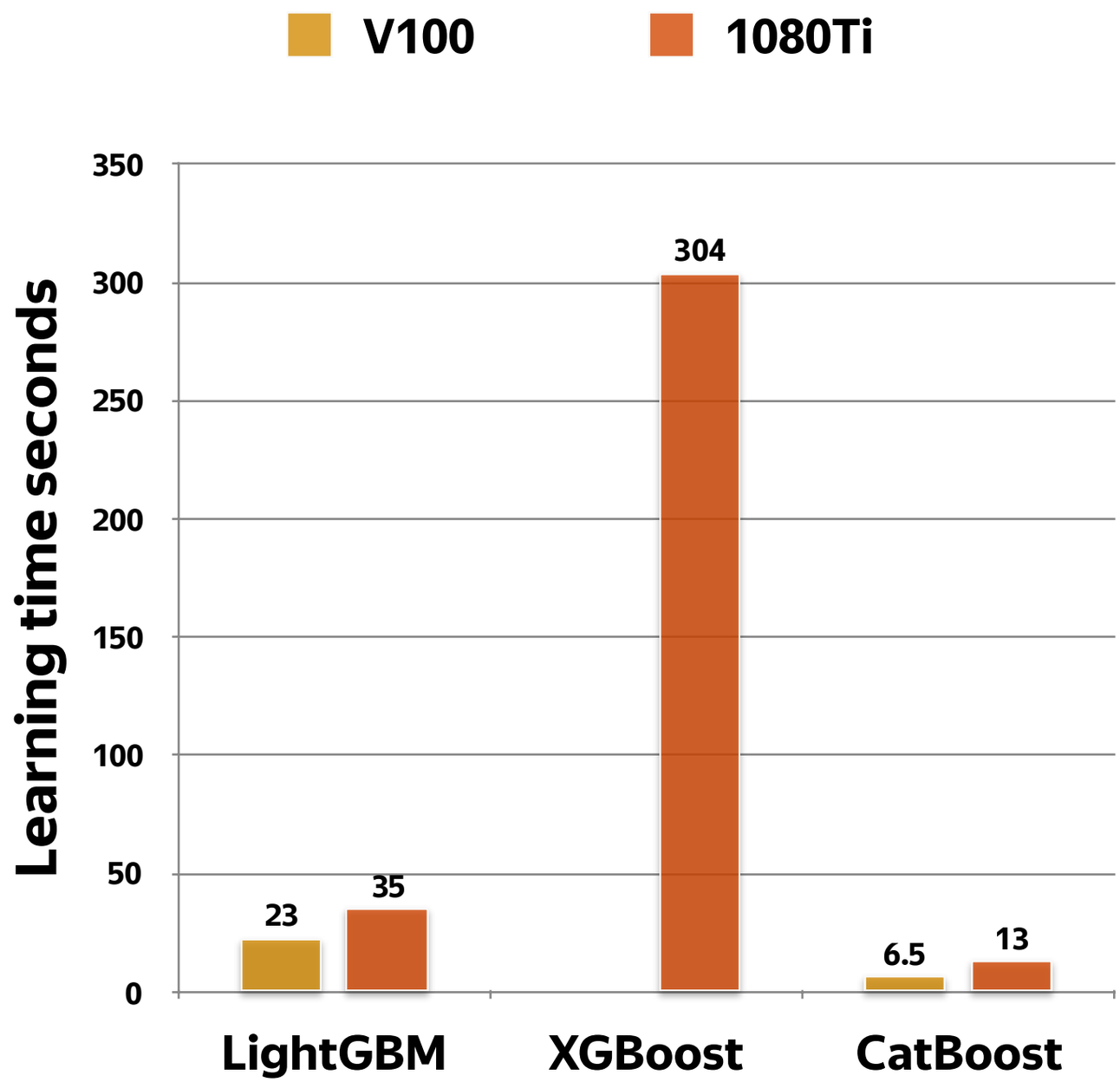
- 32 bins, 64 leaves, 200 iterations

Dataset

- ≈800 features
- 4M samples

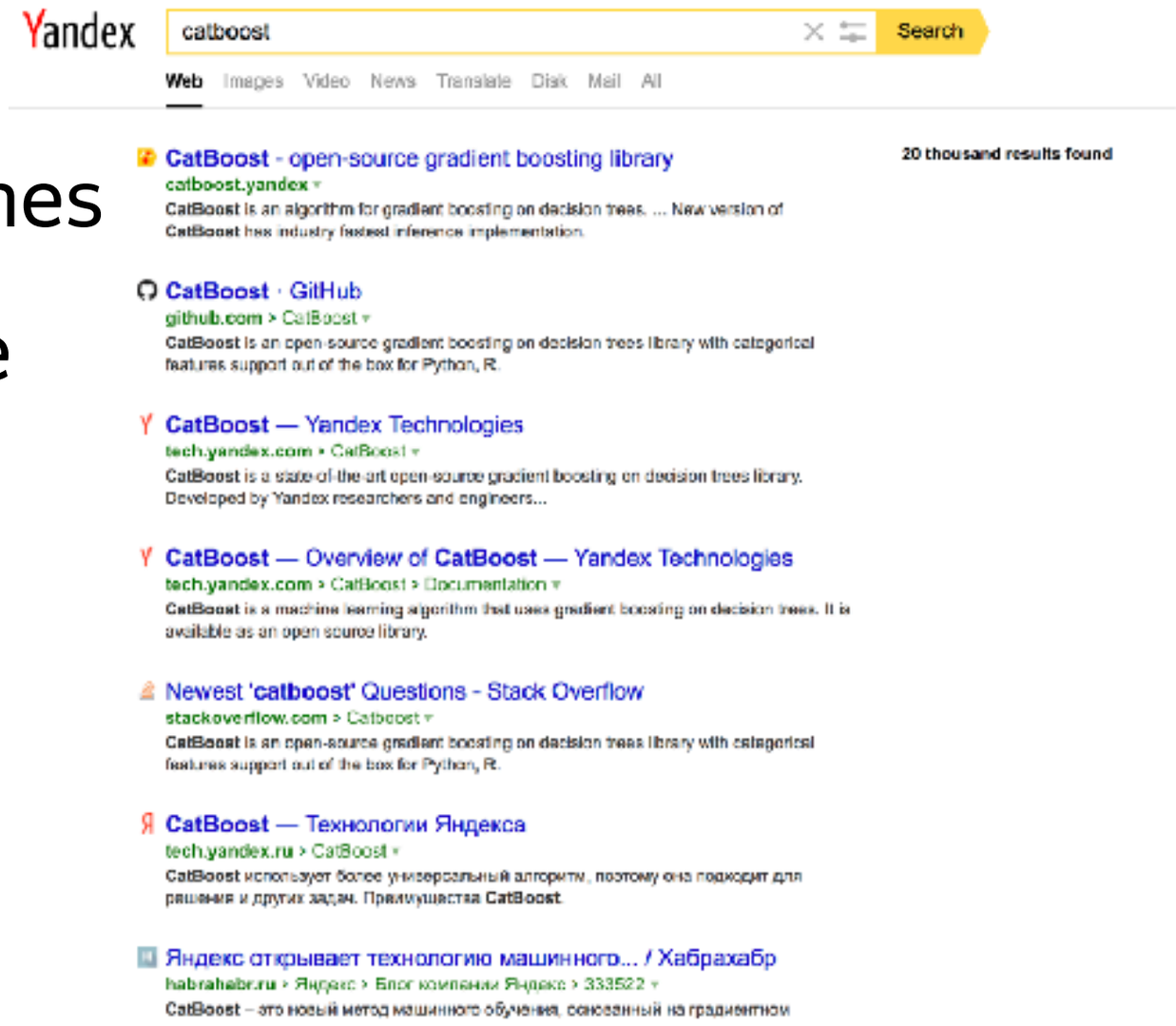
XGBoost + V100? 0.72?

- XGBoost **0.72** crashed with "Illegal Memory Access";
- They have issue from 29 November, just closed without fix



GPU Gradient boosting usage in Yandex

- Ranking formulas:
  - CPU: 75 hours on 100 machines
  - GPU: 7-9 hours on 1 machine with 8 Tesla P40
- First open-source distributed GBDT on GPU



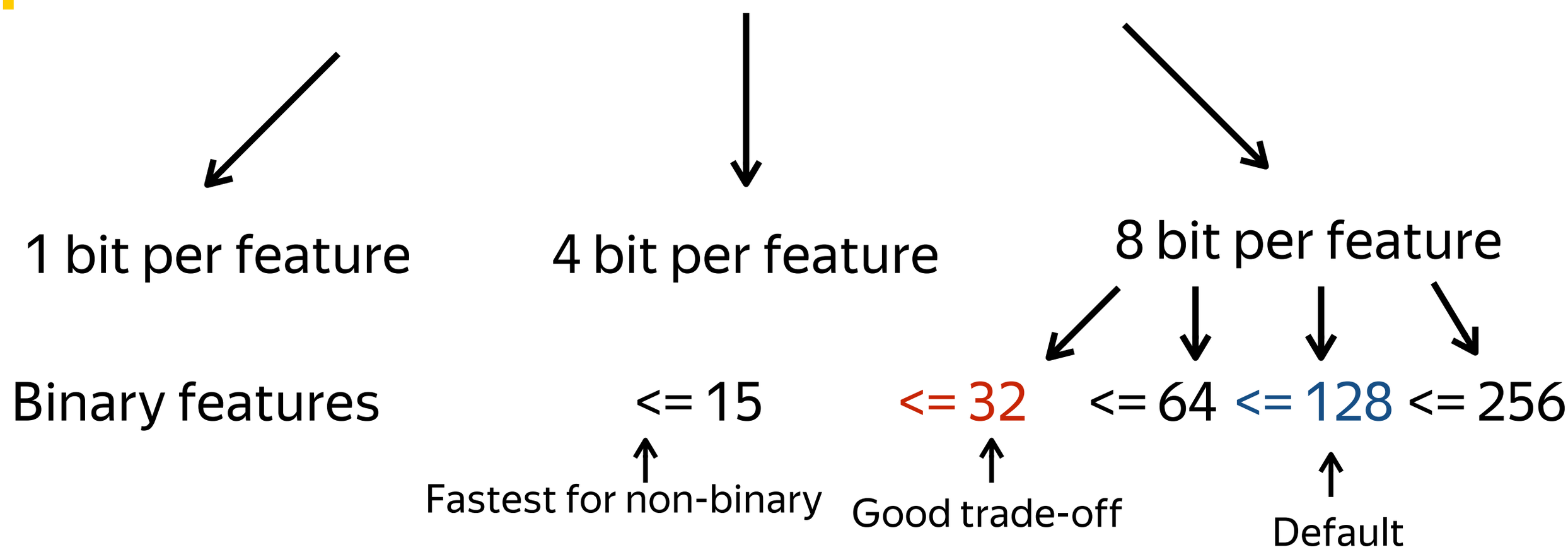
How to use?

First steps

- Install
  - Pip
  - Conda-forge
  - Build from source
- Look at documentation and tutorials
- System requirements:
  - CUDA-compatible devices, 3.0+ (Kepler and later devices)
  - CUDA 9.1 (soon just NVIDIA driver)
  - Python 2.7 or Python 3.4+
  - Windows, Linux, OS X

Ordered features: feature quantisation

- Float -> byte (8-bit float) (border_count param) =>
  - Reduce overfitting
  - Faster learning (histograms for tree fitting)
- Reduce memory usage
- Specialisation for different levels

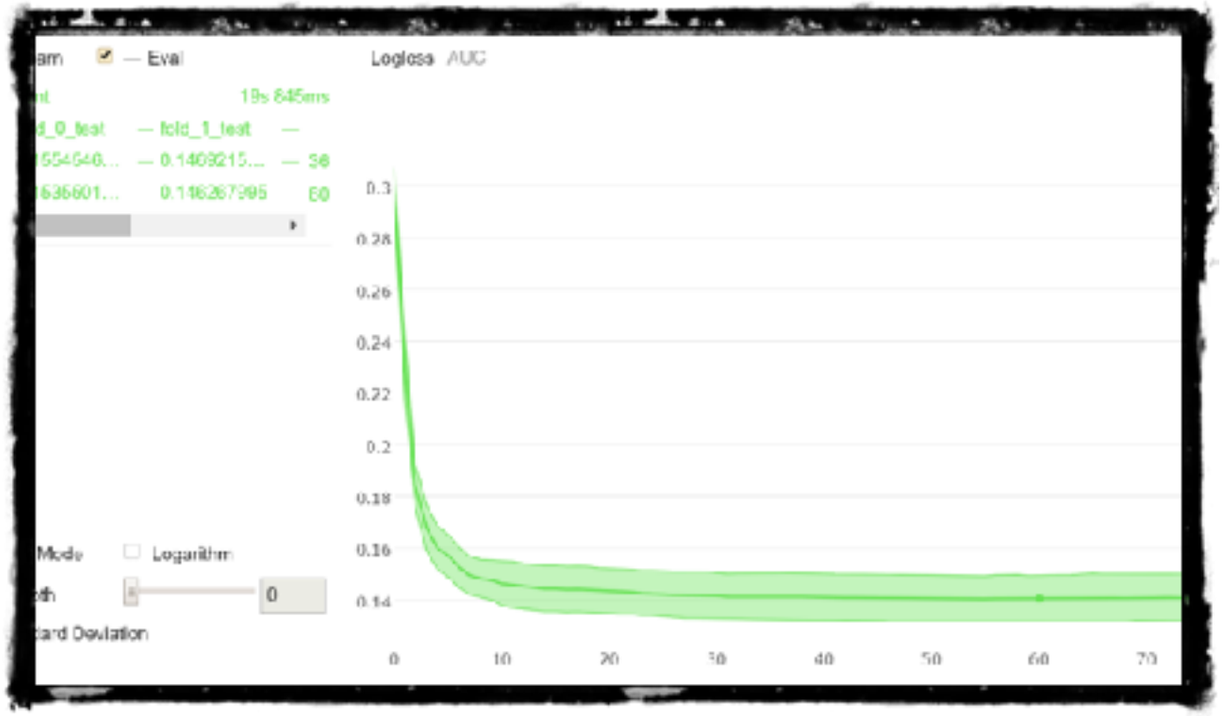


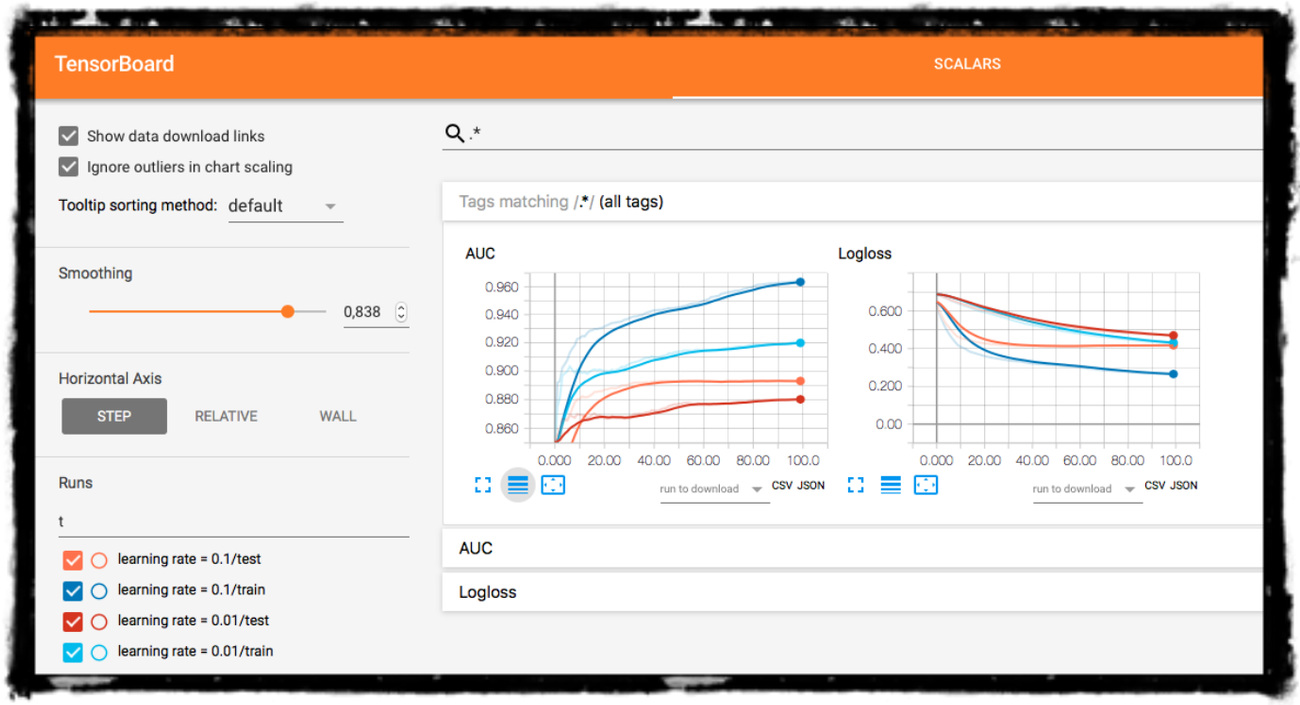
Categorical features

- One-hot-encoding:
  - one_hot_max_size (default 2, maximum 255)
- Control categorical feature combinations search:
  - max_ctr_complexity (default 4, fastest 1)
- Stream categorical features from CPU ram:
  - gpu_cat_features_storage

Other useful features

- Metric evaluation during training (CatBoost viewer, TensorBoard)
  - Some metrics are slow (AUC, ranking metrics) and are computed on CPU, skip_train~true hint for metric and metric_period options could significantly speed-up training





Other useful features

- Metric evaluation during training (CatBoost viewer, TensorBoard)
- Overfitting detector
- Missing values support
- Features and document importance (shap values, etc)
- Fastest inference:
  - Apply and staged predict
  - metric evaluation on datasets

CatBoost

- Regression, Classification, Ranking
- Efficient CPU and multi-GPU version
- State-of-the-art quality on openly available datasets with categorical features
- World fastest inference: thanks to our special type of trees and Intel SSE intrinsics
- Analytical tools
- Stand-alone binary, R, **Python 2.7 and 3.4+**

More math if you are interested

- http://learningsys.org/nips17/assets/papers/paper_11.pdf
- https://arxiv.org/abs/1706.09516

Questions?

For more information:

https://catboost.yandex

github.com/catboost

Vasily Ershov

Software developer

noxoomo@yandex-team.ru