CatBoost

2018.08.28

helpfully tiered resources

- full paper: https://arxiv.org/pdf/1706.09516.pdf
- short paper: http://learningsys.org/nips17/assets/papers/paper_11.pdf
- blog post: https://towardsdatascience.com/catboost-vs-light-gbm-vs-xgboost-5f93620723db

outline

- gradient boosting review
- comparison of gb methods
- cat boost vs. other gb methods
    - categorical features
    - how it does boosting
- conclusion

gradient boosting

- process of constructing an ensemble predictor, F, by iteratively combining weaker learners, h, in a greedy/ additive manner $F_{m+1}(x) = F_m(x) + h(x)$

- the goal of learning is to train an approximation of a function which minimizes the expected value of some loss function

  - in least squares regression, the goal is to teach a model to predict values, $\hat{y} = F(x)$, by minimizing the mean squared error, $1/n \sum_i (\hat{y}_i - y_i)^2$.

gradient boosting

- to find h, gb starts with the observation that a perfect h would imply $F_{m+1}(x) = F_m(x) + h(x) = y$

- gb fits h to the residual $h(x) = y - F_m(x)$

- generalization of this idea to other loss functions and to classification/ranking problems follows from the observation that residuals for the given model are negative gradients with respect to the loss function.

gradient boosting

- most implementations of gb use decision trees as base predictors

  - convenient for numerical features, but have to preprocess categorical features

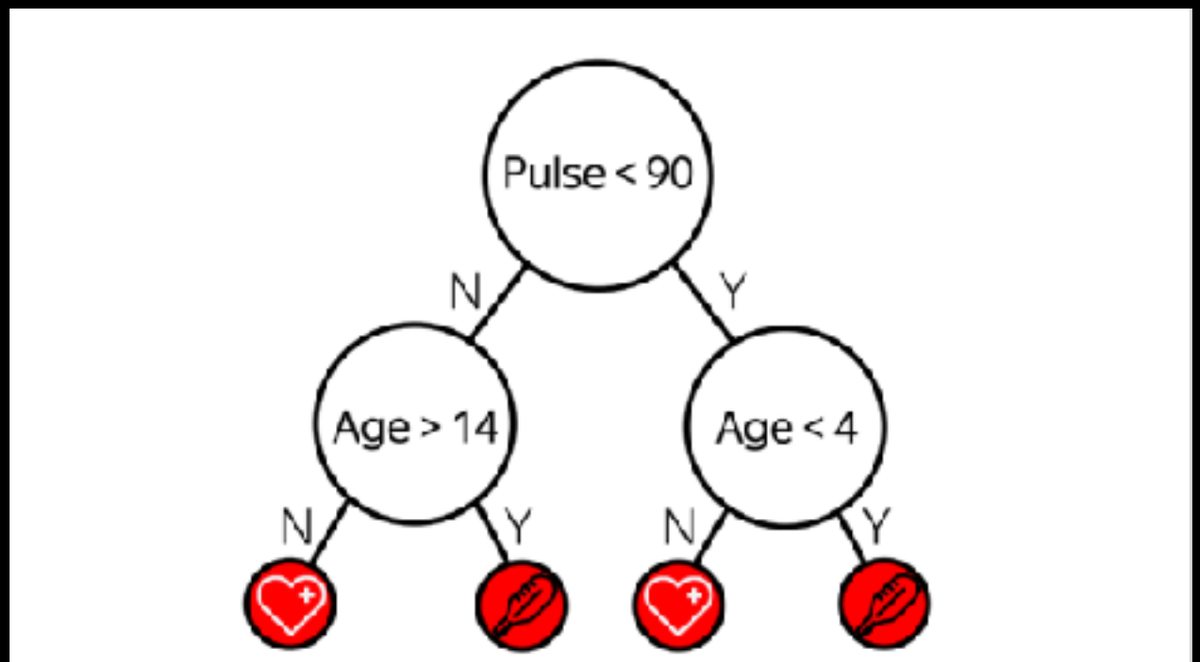
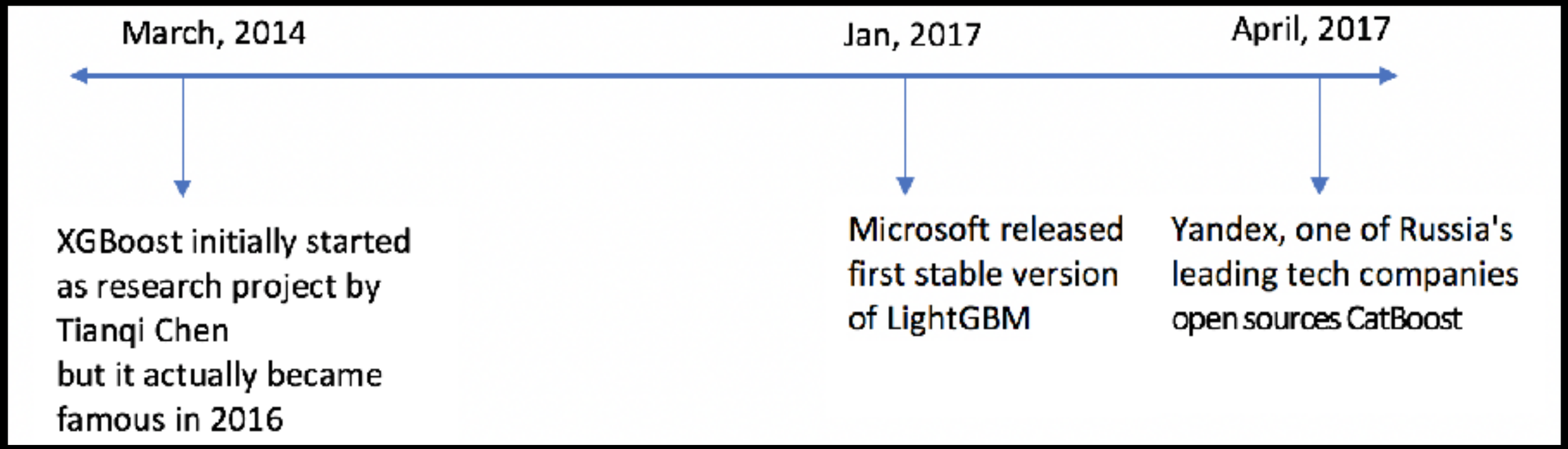


Figure 1: Example of a simple decision tree classifying patients into healthy and sick.

why use gradient boosting methods?

- they rule
- data with heterogeneous features, noisy data, complex dependencies
- learning tasks such as…
  - web search, recommendation systems, weather forecasting,
  - and in our group: student early warning system, Amgen

a brief history lesson



which one should you use?

- kaggle flight delays dataset (sampled 500k rows, 11 cat/num features, binary outcome (>10 min delay or <10 min delay)
- **overall winner: catboost (max test accuracy, min overfitting, min prediction/tuning time)**
  - cat boost > xgboost > light GBM in terms of accuracy, but only when cat feats present
  - xgboost performs almost as well as catboost, but training time is slow
- early warning system performs better with xgboost because many missing values

	XGBoost	Light BGM		CatBoost	
Parameters Used	max_depth: 50 learning_rate: 0.16 min_child_weight: 1 n_estimators: 200	max_depth: 50 learning_rate: 0.1 num_leaves: 900 n_estimators: 300		depth: 10 learning_rate: 0.15 l2_leaf_reg= 9 iterations: 500 one_hot_max_size = 50	
Training AUC Score	0.999	Without passing indices of categorical features	Passing indices of categorical features	Without passing indices of categorical features	Passing indices of categorical features
		0.992	0.999	0.842	0.887
Test AUC Score	0.789	0.785	0.772	0.752	0.816
Training Time	970 secs	153 secs	326 secs	180 secs	390 secs
Prediction Time	184 secs	40 secs	156 secs	2 secs	14 secs
Parameter Tuning Time (for 81 fits, 200 iteration)	500 minutes	200 minutes		120 minutes	

the ordering principle

- two main differences between catboost and other gradient boosting methods
    - 1. how it deals with categorical features: ordered target based statistic (TBS)
    - 2. how it does gradient boosting: ordered boosting
- uses random permutations of the data in both to avoid a statistical issue called **target leakage**, which leads to a **prediction shift** of the learned model

target leakage and prediction shift

- **prediction shift**: shift of the distribution for a training example from the distribution for a test example

- results from **target leakage**: the prediction model relies on the targets of all training examples, so it contains more information about the target of training examples than the target of a test example with the same input feature vector

- finally, these problems lead to overfitting.

prediction shift and boosting

- chain of shifts as a consequence of target leakage
  - the conditional distribution of the trained model in each step of gb for a training example is shifted from that distribution on a test example
  - base predictor is biased with respect to the solution to the loss function
  - the trained model is overfit

1. categorical feats

- handles categorical features as inputs, rather than needing pre-processing (ex: one hot encoding)
    - especially useful when there are many categories
- will compute a target based statistic (TBS) for categorical columns with number unique category values > one_hot_max_size
    - replace observations in a cat. feature with an estimate for the expected target, y, conditioned by category

what should the TBS be?

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

- straightforward approach is for a category, xk, in a categorical feature, xi, take the average target value over the whole training dataset
- target leakage: if the categorical feature has all unique categories, the new numeric feature will be equal to the target variable. in a binary classification task, for example, where P(y=1|xi=category) = 0.5, the classifier would be able to make only one split at t = 0.5 and perfectly classify all training examples. however, it would make errors on the test set with probability 0.5.

a better TBS...

- satisfies two conditions:
  - p1: eliminate target leakage / reduce overfitting
  - p2: the TBS should have low variances for each training example

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

- compute the TBS for each training example with a subset of the whole training dataset, Dk
  - holdout TBS: violates p2 because we have less data to calculate the statistic
  - leave-one-out TBS: reduces variance, but violates p1

ordered TBS

- inspired by online learning problems where training examples become available sequentially in time and the model is updated by considering a sliding window of previous examples

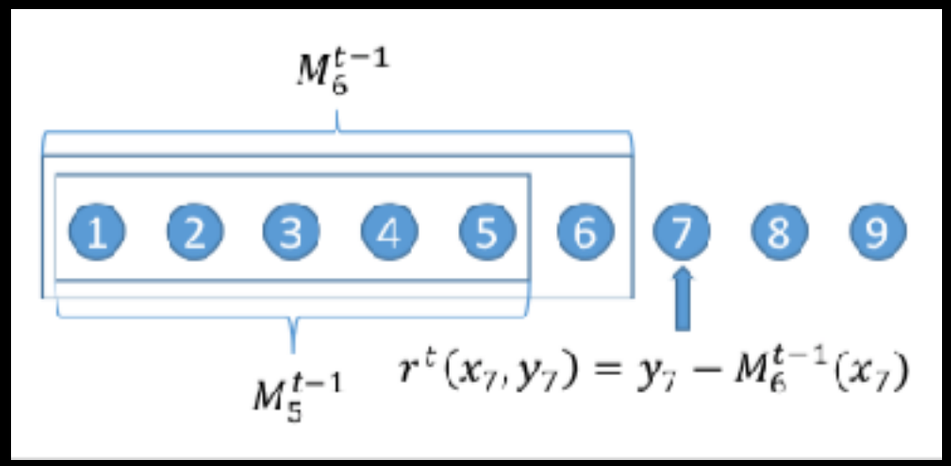
- randomly permute the training data and for each training example, compute the TBS using the previous training data

- use several different permutations for different iterations of gradient boosting rather than a fixed permutation, so preceding examples don’t have TBS with much higher variance than subsequent ones

2. ordered boosting

- example: regression task where at each step, t, the base predictor, $h^t(x)$, approximates the residual function, $r^{t-1}(x, y)$
- we want unshifted residuals for each training example
  - solution 1: at each step of boosting, sample a new dataset independently and obtain unshifted residuals by applying the current model to the new training data
    - not possible in practice bc we have limited training data
  - solution 2: maintain a set of models for each training example

ordered boosting



- random permutation of examples
- maintain n different models, such that the current model is learned using only the first i samples in the permutation
- for each step in training, the residual for the j-th sample is obtained using the model j-1
- this ordered boosting algorithm is not feasible because of the complexity and memory requirements to maintain n models

ob implementation

- construction of a decision tree is a two step process: 1. choosing the tree structure (splitting attributes) and 2. choosing the values in leaves
- phase 1 is done using two modes: Ordered and Plain
  - Ordered mode is an efficient modification of ordered boosting that **uses one tree structure shared by all of the models to be built**
  - Plain mode is an combination of standard GBDT with ordered TBS

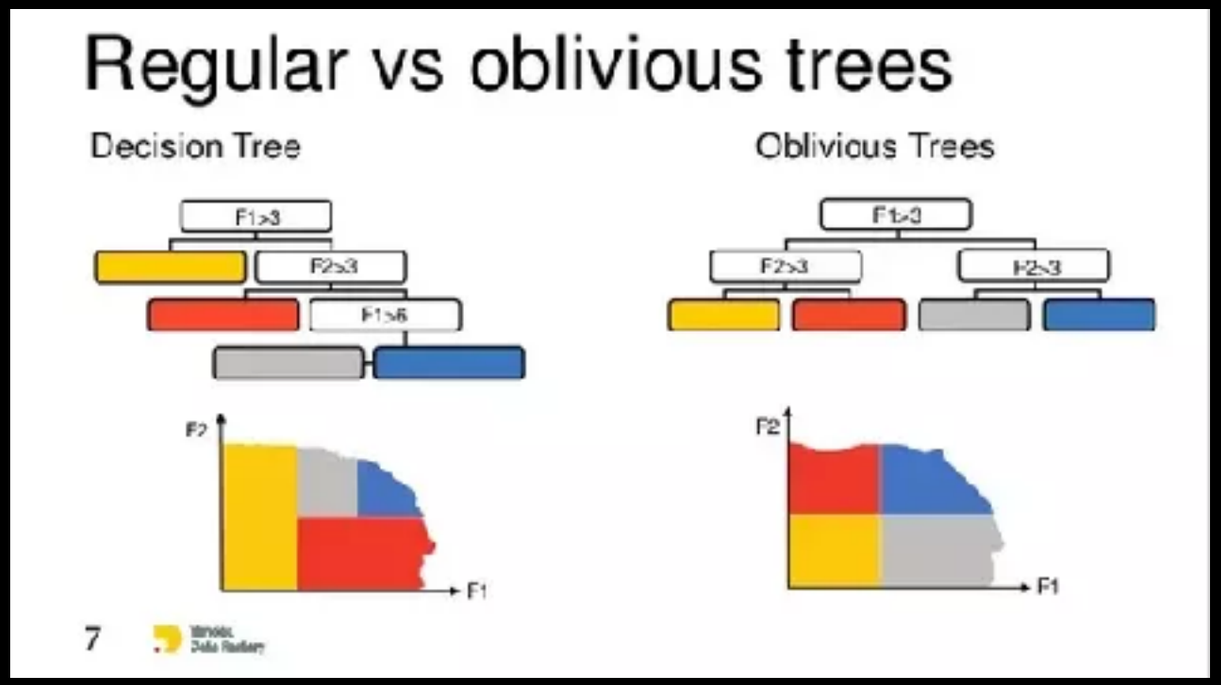
combining ordered boosting with categorical features

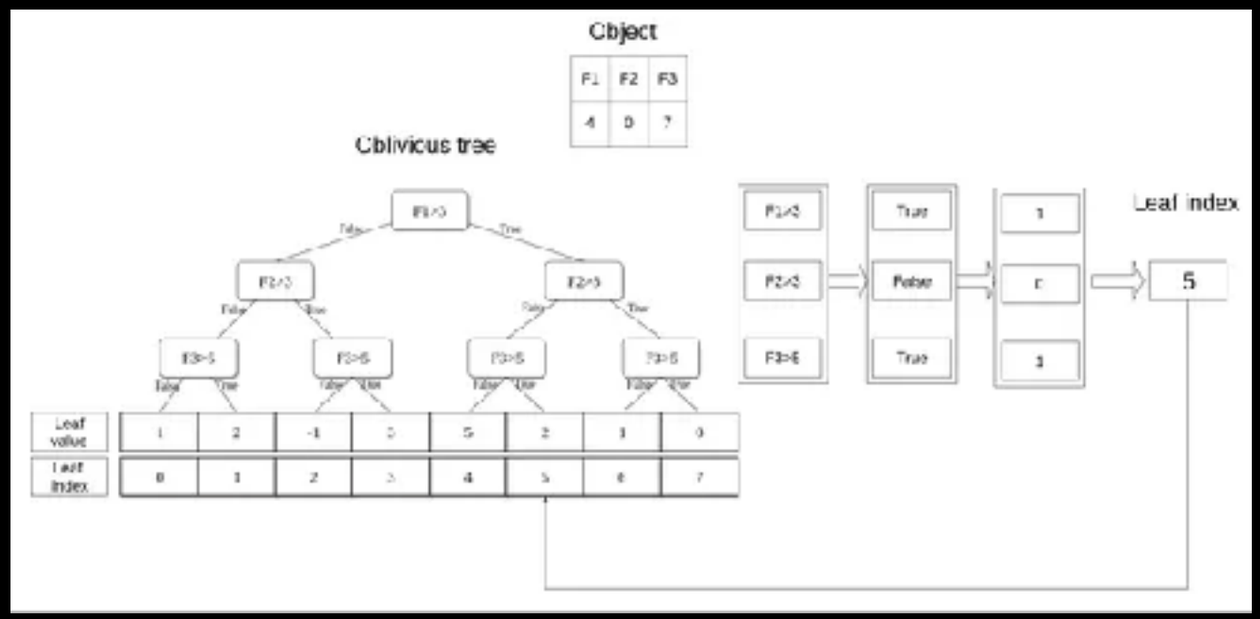
- s+1 independent random permutations of training data
- s0 is used for choosing the leaf values
- each permutation, 1,…,s, is used for both computing ordered TBS and for Ordered boosting, simultaneously
- use several permutations to reduce the effect of high variances on examples with short history

prediction speed up

- oblivious decision trees: same splitting criterion is used across an entire level of the tree, so tree can be represented as a binary vector of length equal to tree depth

- balanced, less prone to overfitting, speeds up prediction at testing time





https://www.quora.com/Lets-say-that-I-want-to-build-a-decision-tree-ensemble-that-can-make-predictions-very-quickly-Is-there-a-library-that-can-help-me-build-such-an-efficient-ensemble

conclusion

- for ML problems,
  - use catboost when there are categorical features and minimizing training/tuning/prediction time is important
  - use xgboost when there aren’t many categorical features or the number of categories is small or when there are many missing values