Description of the experimental setup

1 Introduction

This document contains technical details about the experiments held in order to compare out-of-the-box performance of different gradient boosting methods on the tasks of binary classification. The goal of the description is to clarify the design of the experiments and guarantee the fairness of comparison and high reliability of the results.

We compare CatBoost against popular gradient boosting libraries: LightGBM, XGBoost, H2O. For each algorithm we calculate 2 metrics:

- **Default**. Each of the algorithms has its own specific set of parameters, for which the default values are proposed by their authors. We use these values and tune only number of trees.
- **Tuned**. We tune all the key parameters of an algorithm using hyperopt library in the mode algo=tpe.suggest (i.e., by the sequential Tree Parzen Estimator algorithm).

We use 5-fold cross-validation to tune all the parameters, including number of trees. We fit the model, which performs best on validation, and apply this tuned model to test set.

We guarantee reproducibility of all the experiments by providing docker image with all necessary data and scripts.

2 Datasets

The experiments were performed on the following datasets.

Dataset name	Link	Instances	Features	Description
adult	link	48842	15	Prediction task is to determine whether a person makes over 50K a year. Extraction was done by Barry Becker from the 1994 Census database. A set of reasonably clean records was extracted using the following conditions: (AAGE>16) and (AGI>100) and (AFNLWGT>1) and (HRSWK>0)
amazon	link	32769	10	Data from the Kaggle Amazon Employee challenge.
appet	link	50000	231	Small version of KDD 2009 Cup data.
click	link	399482	12	This data is derived from the 2012 KDD Cup. The data is subsampled to 1% of the original number of instances, downsampling the majority class (click=0) so that the target feature is reasonably balanced (5 to 1). The data is about advertisements shown alongside search results in a search engine, and whether or not people clicked on these ads. The task is to build the best possible model to predict whether a user will click on a given ad.
internet	link	10108	69	Binarized version of the original data set. The multi-class target feature is converted to a two-class nominal target feature by re-labeling the majority class as positive ('P') and all others as negative ('N'). Originally converted by Quan Sun.
kdd98	link	191260	479	Dataset KDD98 challenge. The goal is to estimate the return from a direct mailing in order to maximize donation profits. This dataset represents problem of binary classification - whether there was a response to mailing. The data is subsampled to 50% of the original number of instances.
kddchurn	link	50000	231	Small version of KDD 2009 Cup data.
kick	link	72983	36	Data from "Don't Get Kicked!" Kaggle challenge.
upsel	link	50000	231	Small version of KDD 2009 Cup data.

3 Preprocessing

As the goal of the study is to compare out-of-the-box performance of algorithms themselves, no complex preprocessing (elimination of imbalanced classes, feature selection etc.) takes place. The simplest methods of imputation are used for both numeric and categorical variables:

- For categorical variables missing values are replaced with special value, i.e. we treat missing values as a special category
- For numeric variables missing values are be replaced with zeros, and a binary dummy feature for each imputed feature is added.

4 Preparation of Splits

Each dataset was split into training set (80%) and test set (20%). We denote them as $(X_{full\_train}, y_{full\_train})$ and (X_{test}, y_{test}). For each dataset column numbers with categorical features are known.

We use 5-fold cross-validation to tune parameters of each model on training set. Accordingly, $(X_{full\_train}, y_{full\_train})$ is split into 5 equally sized parts $(X_1, y_1), \ldots, (X_5, y_5)$ (for classification tasks the sampling is stratified). These parts are used to construct 5 training and validation sets $(X_i^{train}, y_i^{train})$, (X_i^{val}, y_i^{val}) so that (X_i^{val}, y_i^{val}) matches (X_i, y_i), and $(X_i^{train}, y_i^{train})$ matches $\cup_{j \neq i}(X_j, y_j)$.

For each of these pairs, we then preprocess the categorical features as follows. Suppose that there is a training set (X^{train}, y^{train}) and a validation set (X^{val}, y^{val}). Next, after a random permutation of the objects for j-th categorical feature and i-th object, we calculate the 2 numbers a_{ij} and b_{ij} (in the formulae [*boolean expression*] is the indicator, it equals 1 if the expression is true and 0, in the other case):

$$a_{ij} = \sum_{k=1}^{i-1} [X_{ij}^{train} = X_{kj}^{train}] y_{kj}^{train},$$

$$b_{ij} = \sum_{k=1}^{i-1} [X_{ij}^{train} = X_{kj}^{train}]$$

After that the categorical features in the training set are replaced by numeric ones using the following formula.

$$X_{ij}^{train} = \frac{a_{ij} + 1}{b_{ij} + 2}.$$

The next step is to replace the categorical features in the validation set. To do this, for j-th categorical feature and i-th object, the 2 numbers c_{ij} and d_{ij} are calculated the same way:

$$c_{ij} = \sum_{k} [X_{ij}^{val} = X_{kj}^{train}] y_{kj}^{train},$$

$$d_{ij} = \sum_{k} [X_{ij}^{val} = X_{kj}^{train}]$$

Now the cat features in the validation set are replaced by numeric ones using the following formula.

$$X_{ij}^{val} = \frac{c_{ij} + 1}{d_{ij} + 2}.$$

Finally, for the original samples $(X_{full\_train}, y_{full\_train})$ and (X_{test}, y_{test}), we replace the categorical features with numerical ones following the same method as for (X^{train}, y^{train}) and (X^{val}, y^{val}).

5 Parameter Tuning Design

As a result of data preparation steps, we have:

- 5 pairs of samples (training and validation) that contain only numeric values which are used to find the best parameters via 5-fold cross-validaton
- pair of samples (full training, test) that contains only numeric values to fit the model with best parameters on the full training set and get predictions for the test set

To estimate quality we use LogLoss.

In the process of parameter tuning, we perform exhaustive search for number of trees, having fixed all other parameters.

The maximum number of trees for all algorithms is 5000.

For a specific set of parameters quality metrics on the validation sets are calculated for each of the 5 folds when adding the next tree. The result is five 5000-dimensional vectors that are averaged into one vector, which is used for getting the argmax or argmin. The resulting number is the optimal number of trees for the given set of parameters. I.e. having all other parameters fixed, we find optimal number of trees by maximizing average quality on 5 folds.

In total, this process is repeated for 50 different parameter sets, and the parameters with the best quality are selected. After obtaining the best parameters, the model is trained on the preprocessed $(X_{full\_train}, y_{full\_train})$. With this model the final metric value for the pre-processed test set (X_{test}, y_{test}) is calculated.

6 Default Parameters

A list of default parameters for each algorithm:

XGBoost.

- 'eta': 0.3
- 'max_depth': 6
- 'subsample': 1.0
- 'colsample_bytree': 1.0
- 'colsample_bylevel': 1.0
- 'min_child_weight': 1
- 'alpha':0
- 'lambda': 1
- 'gamma': 1

LightGBM.

- 'learning_rate': 0.1
- 'num_leaves' : 127
- 'feature_fraction': 1.0
- 'bagging_fraction': 1.0
- 'min_sum_hessian_in_leaf': 10
- 'min_data_in_leaf': 100
- 'lambda_l1': 0
- 'lambda_l2': 0

H2OGradientBoosting.

- 'learn_rate': 0.1
- 'max_depth': 5
- 'sample_rate': 1.0
- 'col_sample_rate': 1.0
- 'col_sample_rate_change_per_level': 1
- 'col_sample_rate_per_tree': 1
- 'min_split_improvement': $1e-5$

- 'min_rows': 10
- 'histogram_type': auto

CatBoost.

- 'learning_rate': 0.03
- 'depth' : 6
- 'fold_len_multiplier': 2
- 'rsm': 1.0
- 'border_count': 128
- 'ctr_border_count': 16
- 'l2_leaf_reg': 3
- 'leaf_estimation_method': 'Newton'
- 'gradient_iterations': 10
- 'ctr_description': ['Borders', 'CounterMax']
- 'random_strength': 1
- 'one_hot_max_size': 0
- 'bagging_temperature': 1

7 Parameter Tuning

Parameters for XGBoost, LightGBM and CatBoost were tuned using python hyperopt library. Parameters for H2OGradientBoosting were tuned using h2o.grid (randomized grid search) in R interface. Below is a list of tuned parameters and the distributions they were selected from for each algorithm:

XGBoost.

- 'eta': Log-uniform distribution $[e^{-7}, 1]$
- 'max_depth': Discrete uniform distribution $[2, 10]$
- 'subsample': Uniform $[0.5, 1]$
- 'colsample_bytree': Uniform $[0.5, 1]$
- 'colsample_bylevel': Uniform $[0.5, 1]$
- 'min_child_weight': Log-uniform distribution $[e^{-16}, e^{5}]$
- 'alpha': Mixed: 0.5 * Degenerate at 0 + 0.5 * Log-uniform distribution $[e^{-16}, e^{2}]$
- 'lambda': Mixed: 0.5 * Degenerate at 0 + 0.5 * Log-uniform distribution $[e^{-16}, e^{2}]$
- 'gamma': Mixed: 0.5 * Degenerate at 0 + 0.5 * Log-uniform distribution $[e^{-16}, e^{2}]$

LightGBM.

- 'learning_rate': Log-uniform distribution $[e^{-7}, 1]$
- 'num_leaves' : Discrete log-uniform distribution $[1, e^{7}]$
- 'feature_fraction': Uniform $[0.5, 1]$
- 'bagging_fraction': Uniform $[0.5, 1]$
- 'min_sum_hessian_in_leaf': Log-uniform distribution $[e^{-16}, e^{5}]$

- 'min_data_in_leaf': Discrete log-uniform distribution $[1, e^6]$
- 'lambda_l1': Mixed: 0.5 * Degenerate at 0 + 0.5 * Log-uniform distribution $[e^{-16}, e^2]$
- 'lambda_l2': Mixed: 0.5 * Degenerate at 0 + 0.5 * Log-uniform distribution $[e^{-16}, e^2]$

H2OGradientBoosting.

- 'learn_rate': Log-uniform distribution $[e^{-7}, 1]$
- 'max_depth': Discrete uniform distribution $[2, 10]$
- 'sample_rate': Uniform $[0.5, 1]$
- 'col_sample_rate': Uniform $[0.5, 1]$
- 'col_sample_rate_change_per_level': Uniform $[0, 2]$
- 'col_sample_rate_per_tree': Uniform $[0, 1]$
- 'min_split_improvement': Log-uniform distribution $[e^{-16}, 1]$
- 'min_rows': Log-uniform distribution $[1, e^5]$
- 'histogram_type': Discrete uniform distribution over a set {uniform_adaptive, random, quantiles_global, round_robin}

CatBoost.

- 'learning_rate': Log-uniform distribution $[e^{-7}, 1]$
- 'random_strength': Discrete uniform distribution over a set {1, 20}
- 'one_hot_max_size': Discrete uniform distribution over a set {0, 25}
- 'l2_leaf_reg': Log-uniform distribution $[1, 10]$
- 'bagging_temperature': Uniform $[0, 1]$
- 'gradient_iterations' : Discrete uniform distribution over a set {1, 10}

8 Versions of the libraries

- xgboost (0.6)
- scikit-learn (0.18.1)
- scipy (0.19.0)
- pandas (0.19.2)
- numpy (1.12.1)
- lightgbm (0.1)
- hyperopt (0.0.2)
- h2o (3.10.4.6)
- R (3.3.3)