Learn Together, Stop Apart: an Inclusive Approach to Ensemble Pruning

Anonymous authors
Paper under double-blind review

Abstract

Gradient Boosting is the most popular method of constructing ensembles that allows to get state-of-the-art results on many tasks. One of the critical parameters affecting the quality of the learned model is the number of members in the ensemble or the number of boosting iterations. Unfortunately, the problem of selecting the optimal number of models still remains open and understudied. This paper proposes a new look at the optimal stop selection problem in Gradient Boosting. In contrast to the classical approaches that select a universal ensemble size using a hold–out validation set, our algorithm takes into account the heterogeneity of data in the feature space and adaptively sets different number of models for different regions of data, but it still uses the same common ensemble trained for the whole task. Experiments on SOTA implementations of Gradient Boosting show that the proposed method does not affect the complexity of learning algorithms and significantly increases quality on most standard benchmarks up to 2%.

1 Introduction

There are still many areas where classical machine learning algorithms prevail over deep neural networks despite the dramatic growth of their usage in artificial intelligence research and industry. One of such classical algorithms is Gradient Boosting (GB) (Friedman (2001)). It allows to obtain high-quality models on table data with no multimedia (e.g., images, audios, videos), with categorical features, noisy features and labels, and missing data (Zhang & Haghani, 2015; Li et al., 2007; Babajide Mustapha & Saeed, 2016). Also, the undoubted advantage of the boosting method is the relatively low computational cost of training and inference (Deng et al., 2018). For these reasons, Gradient Boosting is widely used in ranking (Chapelle & Chang, 2011), recommender systems (Cheng et al., 2014), automatic machine learning (LeDell & Poirier, 2020), and many other tasks (Touzani et al., 2018; Trofimov et al., 2012; Ling et al., 2017).

In recent years, different new options and hyperparameters have been proposed for GB influencing its performance (Ke et al., 2017; Ibragimov & Gusev, 2019). However, the learning rate and the size of the ensemble are still the key parameters. Large models reveal complex dependencies in the data but require more time for training and inference (Friedman, 2002). Moreover, they are prone to overfitting on noisy datasets with simple dependencies, while smaller models perform better on such data. The standard approach to select an optimal number of models in the ensemble is the *early stopping* method based on a hold-out *validation set*. The idea is to set a large enough model size and choose the value of the size (overfitting point) where the validation score stops growing and begins going down.

The standard early stopping method has a significant and surprisingly understudied weakness. The approach assumes the existence of a universal ensemble size equally effective for any data point. However, the learning task can consist of different subtasks corresponding to different regions in the input space and functional dependencies. Some regions can have complex surfaces for training where many boosting rounds are required until convergence, and some regions can reveal simple but noisy dependencies, where overfitting begins much earlier. In such a situation, the standard early stopping is a compromise between simple and complex areas, and it can provide models with a composition of overfitted and underfitted regions.

To handle this issue, we propose a new approach to early stopping in GB based on an adaptive choice of the optimal size of the ensemble. As in the standard version of GB (Friedman, 2001), we train one sequence of learners in an ensemble. However, at inference, we apply different number of learned models to different regions in the dataset. Namely, we build an additional *partition* model that sequentially divides the input space into regions of presumably homogeneous complexity and representativity of data and, at the same time, optimizes the size of the trained ensemble to each region individually. It turns out that building such a model is not an easy task, and that is the reason why such an algorithm was not implemented earlier in the main GB libraries. In particular, straightforward approaches such as direct prediction of the best iteration have fundamental drawbacks and do not work. Besides, the non-trivial problem is controlling the new hyper-parameters: partition's complexity and generalization ability of the partition model. These are, in some sense, meta-hyper-parameters over hyper-parameters, since the partition model defines ensemble sizes. We develop a specific two-level cross-validation scheme to control the whole construction.

Despite the apparent complexity, the proposed methods incur meager computational costs and can be easily incorporated into any existing learning pipeline and applied to any learning task with arbitrary loss. We apply the proposed approach to state-of-the-art open-source GB algorithms, LightGBM and CatBoost, and demonstrate its ability to outperform consistently on standard publicly available benchmarks for GB. We show that the described problem of the universal stopping moment highly affects the quality of trained models. To the best of our knowledge, this is the first research devoted to effective adaptive early stopping in GB, and we hope this paper will encourage further research of the GB algorithm.

2 Related work

2.1 Ensemble pruning

Pruning often refers to various techniques for compressing models for more efficient storage and inference complexity. The classic work on this task (Margineantu & Dietterich, 1997) compared five different pruning methods applied to boosting algorithm. In most cases, pruned models were able to maintain and increase the original quality with a moderate reduction in size. Most of the modern pruning techniques are based on the fact that similar learners in the ensemble duplicate the information about the dataset so that they can be eliminated from a model sequence (Cavalcanti et al., 2016; Li et al., 2012). There also have been attempts to formulate ensemble pruning as an optimization problem and apply genetic algorithms (Zhou & Tang, 2003) or semi-definite programming (Zhang et al., 2006) to find a solution.

Several papers (Cruz et al., 2015; 2018) addressed the problem of adaptive online pruning in Multiple Classifier Systems settings, where classifiers are learned independently and selected via meta-learning approaches. Also, (Oliveira et al., 2017) and (Hernández-Lobato et al., 2008) propose an instance–wise pruning methods that allow for halting some models at inference time, while in (Soto et al., 2014) both static (training time) and dynamic (inference time) pruning in AdaBoost are investigated.

2.2 Gradient Boosting early stopping

Unlike some other ensemble methods (including bagging), GB suffers from overfitting when the ensemble size is large. Therefore, the control of the number of boosting steps is primarily a regularization technique (Fan et al., 2002). In particular, the original paper (Friedman, 2001) contains direct guidance to tune the number of models in the ensemble (Section 5):" For additive expansions (2) a natural regularization parameter is the number of components M".

Since the GB size is responsible for the expressiveness of the ensemble, one of the ideas proposed in the literature (Chang et al., 2010; Mayr et al., 2012) is to penalize the complexity of the models, e.g., via AIC-based methods by approximating the ensemble's degrees of freedom. Some works use generalization bounds of the algorithm employing VC-dimension (Freund & Schapire, 1997), Rademacher complexity (Cortes et al., 2019), or in the PAC setting (Yao et al., 2007; Wei et al., 2017). These methods do not require separate validation control but, in most cases, are not applicable in real-world tasks since the obtained bounds are distribution–agnostic.

The standard approach of early stopping used in all modern GB implementations utilizes the simple "waiting" idea. If the validation quality does not improve for some "reasonable" number of iterations, then the training must be stopped (see, e.g., (Click et al., 2016)). In this paper, we adopt a standard early stopping scheme described in (Margineantu & Dietterich, 1997): shrink the model to the first M learners that give the best validation score. Nevertheless, unlike all previous works on Gradient Boosting, instead of a universal constant, we strive to select this number adaptively for different regions of the input space, taking into account the training data distribution.

3 Background

In this section, we introduce necessary notations and briefly discuss basic concepts concerning Gradient Boosting and cross-validation for independent reading reasons.

3.1 Gradient Boosting

Let $\mathcal{S} = \{\boldsymbol{x}_i, y_i\}_{i=1}^n$ be a sample from some fixed but unknown distribution $P(\boldsymbol{x}, y)$, where $\boldsymbol{x}_i = (x_i^1, ..., x_i^m) \in \mathbb{X}$ is an m-dimensional feature representation and $y_i \in \mathbb{Y}$ is a target value of the i-th observation. We consider the learning problem that consists in constructing a function $F : \mathbb{X} \to \mathbb{Y}$ minimizing the expected target prediction error, which is calculated using a loss function $L : \mathbb{Y} \times \mathbb{Y} \to \mathbb{R}_+$: $\mathcal{L}(P, F) := \mathbb{E}_{(\boldsymbol{x},y)\sim P}[L(F(\boldsymbol{x}), y)] \to \min\limits_F$. Since the distribution P is not given, the task reduces to empirical risk minimization problem:

$$\hat{\mathcal{L}}(\mathcal{S}, F) = \hat{\mathbb{E}}_{(\boldsymbol{x},y)\sim \mathcal{S}}[L(F(\boldsymbol{x}), y)] = \frac{1}{n}\sum_{i=1}^n L(F(\boldsymbol{x}_i), y_i) \to \min_F$$

The ability to achieve the smaller value of the empirical risk is bounded by the complexity of the set $\mathcal{F}$ from which the desired function $F \in \mathcal{F}$ is selected. *Gradient Boosting (GB)* increases the expressiveness of the learned model by building a composition (or an *ensemble*) F_B of size B as a weighted sum of base functions $\{f_1, f_2, ..., f_B\} \subset \mathcal{F}$:

$$F_B(\boldsymbol{x}) = \sum_{i=1}^B \alpha_i f_i(\boldsymbol{x}) \tag{1}$$

When the set of available ba The approximate solution of the latter equation in GB is usually constructed as follows. Algorithm calculates first and second order derivatives of $\hat{\mathcal{L}}$ at the point F_{t-1} w.r.t. predicted values $\hat{y}$: $g_i^t = \frac{\partial L(\hat{y}_i, y_i)}{\partial \hat{y}_i}\Big|_{\hat{y}_i = F_{t-1}(\boldsymbol{x}_i)}$, $h_i^t = \frac{\partial^2 L(\hat{y}_i, y_i)}{\partial \hat{y}_i^2}\Big|_{\hat{y}_i = F_{t-1}(\boldsymbol{x}_i)}$, and selects a least squares estimator to Newton's gradient step in the functional space:
$f_t = \arg\min\limits_{f\in\mathcal{F}} \sum\limits_{i=1}^N h_i^t(\vec{x_i}, y_i)\left(f(\vec{x_i}) - \left(-\frac{g_i^t(\vec{x_i}, y_i)}{h_i^t(\vec{x_i}, y_i)}\right)\right)^2$, see (Chen & Guestrin, 2016) for details.

3.2 Model selection and early stopping via cross-validation

Since the quality estimation based on a train set used in the learning process is biased (Prokhorenkova et al., 2017), it is conventional to use a separate independent set, called *validation set*, to control the generalization ability of the algorithm. The whole dataset $\mathcal{S}$ is split into two disjoint sets $\mathcal{S}_{train}$ and $\mathcal{S}_{valid}$, where the first one is used for learning and the latter one for quality estimation.

The final result of this procedure is often highly dependent on the particular train-validation split and, therefore, quality estimation can be noisy. A common way to tackle this issue is to use *cross-validation* (Stone, 1974) method: split the data $\mathcal{S}$ into k disjoint subsets or *folds* $(\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_k)$ of approximately equal size s.t. $\mathcal{S} = \bigsqcup\limits_{i=1}^k \mathcal{S}_i$ and perform k rounds of training–evaluation cycle using $\mathcal{S}_{-i} := \mathcal{S} \setminus \mathcal{S}_i$ as the training set and $\mathcal{S}_i$ as the validation data for each $i \in \{1, 2, ..., k\}$. In this way, we get k different B-sized models $\{F_B^j(\boldsymbol{x}) = \sum\limits_{i=1}^B \alpha f_i^j(\boldsymbol{x})\}_{j=1}^k$ learned by k training sets $\{\mathcal{S}_{-j}\}_{j=1}^k$.

Algorithm 1 Adaptive stopping procedure

```
**Input:** $\mathcal{S} = (\boldsymbol{X}, \boldsymbol{y})$
$folds \leftarrow (\mathcal{S}_1, \mathcal{S}_2, ..., \mathcal{S}_k) \leftarrow CvSplit(k, \mathcal{S})$
$cvPred \leftarrow CvPredict(folds)$
$partition \leftarrow (\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_C) \leftarrow GetPartition(\mathcal{S})$
$bestIter \leftarrow EvalBestIter(folds, cvPred, partition)$
$finalModel \leftarrow Train(\boldsymbol{X}, \boldsymbol{y}, partition, bestIter)$
**return** $finalModel$
```

At j-th cross–validation step, we apply all the prefixes F_i^j of the model F_B^j to validation set $\mathcal{S}_j$ and obtain quality estimators $\boldsymbol{l}_j = (l_j^{(1)}, l_j^{(2)}, ..., l_j^{(B)})$, where $l_j^{(b)} = \frac{1}{|\mathcal{S}_j|} \sum\limits_{(\boldsymbol{x},y)\in\mathcal{S}_j} L\left(F_b^j(\boldsymbol{x}), y\right)$. The aggregated estimator $\boldsymbol{l} = \frac{1}{k}\sum \boldsymbol{l}_j$ is further used to define $\hat{B} := \underset{1\le b\le B}{\arg\min}\, l^{(b)}$. The model shrinked to the first $\hat{B}$ iterations provides an estimator with the test quality close to $\min\limits_{1\le b\le B} \mathbb{E}_{(\boldsymbol{x},y)\sim P}[L(F_b(\boldsymbol{x}), y)]$.

4 Adaptive early stopping

The problem with the standard early stopping method described in Section 3.2 is that the desired goal should be to minimize $\mathbb{E}_{(\boldsymbol{x},y)\sim P} \min\limits_{1\le b\le B}[L(F_b(\boldsymbol{x}), y)]$ due to an obvious inequality:

$$\mathbb{E}_{(\boldsymbol{x},y)\sim P} \min_{1\le b\le B}[L(F_b(\boldsymbol{x}), y)] \le \min_{1\le b\le B} \mathbb{E}_{(\boldsymbol{x},y)\sim P}[L(F_b(\boldsymbol{x}), y)]. \tag{2}$$

This simple mathematical fact convinces us that the existing early stopping scheme is ineffective. Adaptive selection of individual ensemble sizes for specific areas can achieve better quality by eliminating the theoretical gap given by inequality 2. In the following sections, we describe possible approaches to adaptive iterations count selection and to control its effect.

4.1 Main idea

Suppose the input space $\mathcal{D}$ is divided into C disjoint regions $(\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_C)$ in such a way that all samples in $\mathcal{D}_i$ are close to each other in some sense (they follow the same latent distribution or geometry). Note that this partition is unrelated to the split induced by cross-validation since the latter split is done randomly, and there is no reason to expect the closeness of samples inside a single fold. We assume that $(\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_C)$ is clustering in the sense that data points of the same cluster $\mathcal{D}_i$ behave similarly during the procedure of training an ensemble in the sense that the optimal number of boosting iterations $\hat{B}_i$ estimated for $\mathcal{D}_i$ may differ a lot from the one estimated for $\mathcal{D}_j$. Therefore, by analogy with the inequality 2, we can conclude that ensemble size selection based on partition $\mathcal{D}$, where the size is chosen individually for each cluster $\mathcal{D}_i$, can have better quality compared to one "universal" common size:

$$\mathbb{E}_{\mathcal{D}_i\sim\mathcal{D}} \min_{1\le b\le B} \mathbb{E}[L(F_b(\boldsymbol{x}), y)|\mathcal{D}_i] \le \min_{1\le b\le B} \mathbb{E}_P[L(F_b(\boldsymbol{x}), y)]. \tag{3}$$

Setting $C = n$ may achieve the theoretical lower bound of the left-hand side of Equation 3. However, the size $\hat{B}_i$ of the ensemble will be optimized based on the empirical estimation of the loss, and the growth in C is accompanied by the growth of the variance of this estimation for each region $\mathcal{D}_i$. So the number of regions should be selected reasonably (we discuss it further in the text).

The upper-level training algorithm consists of 4 steps: 1) Cross-validated training of k models; 2) Distribution-based partition $(\mathcal{D}_1, \mathcal{D}_2, ..., \mathcal{D}_C)$ of the sample space; 3) Selecting optimal number of iterations $(\hat{B}_1, \hat{B}_2, ..., \hat{B}_C)$ for each region obtained on the step 2; 4) Train the final model on the whole training data. The formal description is presented in the Algorithm 1.

4.2 Optimal point regression

One of the most straightforward ideas towards adaptive stopping is to learn a regression model, which predicts an optimal number of term in the trained ensemble to be applied to each example. In terms of the general adapting stopping framework (Algorithm 1), $GetPartition$ should be a function assigning each data point to an individual cluster and $EvalBestIter$ should train a mapping from data points to the best number of iterations (obtained from $cvPredictions$ matrix). This regression model is further applied to test examples to estimate the early stopping moments.

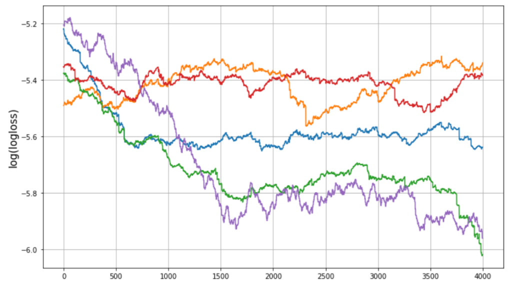

Figure 1: GB: learning curve samples (test set)

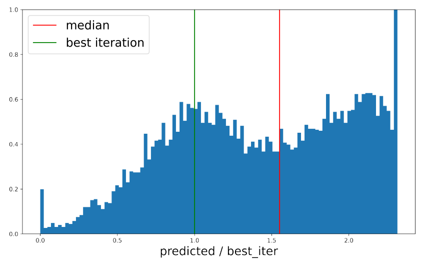

Figure 2: Distribution of relative shift of best iteration predictions

Unfortunately, this simple idea does not work in practice. Training a separate GB regression model with the same size as the primary model followed by applying it on the test sample and individual pruning degrades the prediction logloss compared to the standard non-adaptive scheme for selecting the number of trees by 1.5% on average.

This is because the best iteration is known to be a very noisy target as the Gradient Boosting training process is complex and in some cases partially random (Friedman, 2002). Visualization of learning curves (Figure 1) of test samples demonstrates that loss histories are very chaotic and look like realizations of some random process. Therefore, each curve can accidentally reach the minimum value at an arbitrarily late point. However, as practice shows, the general minimum point (following the trend line) is bounded with some finite value. That is why training a good regressor is a challenging task. Figure 2 demonstrates the results of best iteration regression via a separate Gradient Boosting model. We take the trained regression model predictions and divide them by the actual best stopping moment. From there, we can see that at least half of the samples are overestimated by about two times, so selected stops are far suboptimal.

4.3 Unsupervised partition

As was mentioned at the beginning of this Section, the partition should reflect the internal structure of the data to be sophisticated enough to select a fair number of models. Let us use a reasonable assumption that close observations in the feature space are also close in their properties. Then we can use one of clustering algorithms (e.g., KMeans (Lloyd, 1982), EM (Dempster et al., 1977), agglomerative method (Sibson, 1973)) to get data partition (function $GetPartition$ in Algorithm 1).

It is essential to preserve the initial geometry of the input space since most of the modern implementations of Gradient Boosting use Decision Tree (Breiman et al., 2017) as a base learner. Decision Tree constructs piecewise-constant approximations at each step, and it is more likely for close instances to get into the same leaves during training and inference, so they tend to fit equally. The unsupervised partition method allows controlling the number of partition regions and their sizes via setting the desired number of clusters, and minimal samples count in each cluster.

This method applied to real data exhibits several disadvantages. First, clustering does not work well with data with non-numeric categorical features. Numeric encoding of high-cardinality categorical features leads to sparse input space and dramatically affects clusterization's capacity. Second, unsupervised partition does not consider the labels of the data points, although they may contain

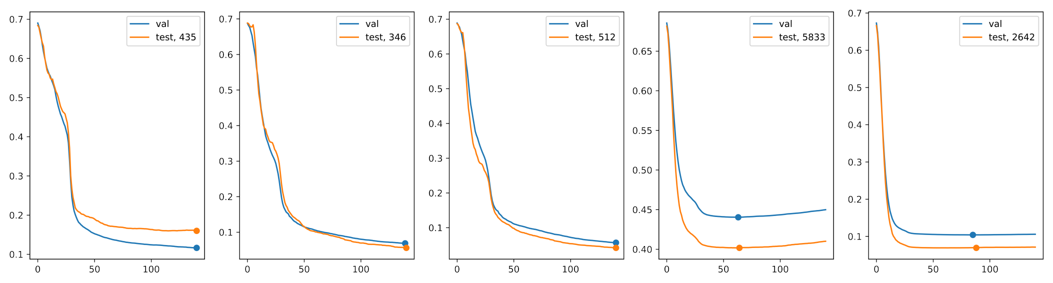

Figure 3: Clustering obtained via decision tree trained with Eq. 4 as split scoring function (DSP), validation and test results. The number after "test" is test samples count per cluster.)

valuable information about the required number of boosting steps. Last, some advanced clustering algorithms require high computational costs, becoming a bottleneck when training a model.

To validate the statements and assumptions above, we use large numeric datasets Higgs and HEPMASS and compare KMeans clustering with the algorithm proposed in this paper. The unsupervised approach turns out to be a good choice for adaptive pruning strategy but, at the same time, strictly worse (see experiments Section 6) than the supervised one described in the following subsection.

4.4 Direct Supervised Partition (DSP)

To avoid issues described in the previous paragraph, the partition unit (function $GetPartition$ in Algorithm 1) should be scalable, interpretable in terms of built subspaces, and tolerant to heterogeneous feature input. We find the Decision Tree model to be a suitable candidate since it satisfies all the listed properties: the training algorithm is parallelizable and not memory consuming (Sharp, 2008), cluster manifolds are similar to the ones built by base learners, and there are efficient categorical feature supporting methods (Prokhorenkova et al., 2017).

Since we aim to find data regions with similar training properties, we can utilize the validation learning curves to split the dataset. So the idea is to train a decision tree that divides the feature space into regions (leaves) using the learning curves as a target. However, the naive multi-regression tree based on the MSE loss may find similar learning curves, chosen clusters do not necessarily have diversified stopping moments. The true goal is to find a partition that will most benefit from assigning individual stops to each cluster. Formally speaking, at each node of the tree, we seek for a split L, R which maximizes the following score (minimizes loss):

$$\mathcal{S}(L, R) := -\left(\min_{1\leq b\leq B}\sum_{i\in L} l_{i,b} + \min_{1\leq b\leq B}\sum_{i\in R} l_{i,b}\right), \tag{4}$$

where $l_{i,b} := L(F_b(\boldsymbol{x}_i), y_i)$. Figure 3 demonstrates how this partitioning works. It is also worth noting that it is possible to optimize any quality metric rather than loss function. In the general case, it is sufficient to store predictions from prefixes of the ensemble and use them to calculate a particular metric function in each split. In particular, storing predictions instead of learning curves can help directly optimize the $ROC - AUC$ score, an example of a non-summable metric.

The complexity of calculating the score above is linear on the ensemble's whole (unpruned) size, so the full tree growing complexity is $O(nmBd)$, which is similar to the GB training complexity. Also, the method requires storing learning histories of size B for each validation point. Since these costs are inappropriate in many real-world tasks (e.g., where large ensembles are needed), we propose to use values from some chosen iterations, e.g., 1, 2, 4, 7, 11, ... (the step increases by one), which reduces the complexity by a factor of $\sqrt{B}$. Here we assume that as the number of iterations increases, predictions change less, so points from earlier iterations carry more information about the curves. Besides, the proposed scheme dramatically reduces time $\left(O(nm\sqrt{B}d)\right)$ and memory costs $\left(O(n\sqrt{B})\right)$, making them moderate compared to the GB training algorithm.

4.5 Indirect Supervised Partition (ISP)

Let us take the idea from the previous section and set the number of iterations that are used to build a clustering tree to 1. We will get an extreme case of the method described earlier, which will depend only on the dataset's characteristics. In other words, the partition procedure boils down to training a single decision tree on the initial training samples and targets. Then, we denote each leaf as a separate data cluster forming the partition. Since the tree learning process utilizes both geometry of feature space and target distribution in leaves to split the data, this method is encouraged to find the regions similar by feature representation and label. In other words, it uses all available *static* information about the data.

This partition tree may be trained separately from the primary boosting model and be the first booster in the ensemble. The latter means that this step does not affect the training time. However, since Gradient Boosting usually consists of hundreds and thousands of trees, the effect on time costs of using a separate partition model is negligible.

The practice shows that these clusters are good enough to find regions with diversified optimal stops and increase the final quality of the model (Section 6 for the details).

5 Validation protocol

Proposed methods introduce additional hyperparameters, which can be tuned: cluster count and minimal size of clusters. Since both DSP and ISP algorithms utilize a decision tree, those hyperparameters are controlled by limiting the maximum number of leaves and the minimal size of each leaf.

Let us denote $\mathcal{D}_{i,j} = \mathcal{D}_i \cap \mathcal{S}_j$ the set of observations from the j-th fold belonging to the cluster $\mathcal{D}_i$ and $n_{i,j} = |\mathcal{D}_{i,j}|$. Naive approach of evaluation consists of applying cross–validation model trained on the sample $\mathcal{S}_{-j}$ to the validation set $\mathcal{S}_j$ for any j, obtaining quality estimators $\boldsymbol{l}_{i,j}$:

$$l_{i,j}^{(b)} = \frac{1}{n_{i,j}} \sum_{(\boldsymbol{x},y)\in\mathcal{D}_{i,j}} L\left(F_b^j(\boldsymbol{x}), y\right).$$

The resulting estimator $\boldsymbol{L}_i$ for each cluster i is a weighted sum of corresponding cluster estimators over all folds: $L_i^{(b)} = \frac{\sum_{j=1}^k n_{i,j}\cdot l_{i,j}^{(b)}}{\sum_{j=1}^k n_{i,j}}$, then $\hat{B}_i := \arg\min \boldsymbol{L}_i$ and the cross–validation score of cluster i equals to $\min \boldsymbol{L}_i$. The total complexity of the described procedure is $O(C(B+k)+nB)$, which is meager compared to the ensemble training complexity, which is at least $O(nmdB)$ (Friedman, 2001) (for m binary features and trees of depth d).

The quality assessment obtained in the way described above is biased and always gives an optimistic estimate. It is especially impossible to use this quality estimator to determine an optimal number of clusters, as it always monotonically increases with finer clustering. To avoid this issue in the DSP case, we conduct an additional validation step to verify that the partition will be profitable by training the early stopping model and applying it to a separate hold-out sample. In particular, if cross-validation is used during GB tuning, we have a learning curve for each train instance, so the whole sample can again be used. Also, DSP can use the minimal gain criterion - if the resulting score after split is not significantly better than the current score, tree growth should stop.

For the ISP case, when the first booster of the full model is used, and there is no possibility to retrain the clustering tree (or retraining is considered costly), we suggest using the following cross-validation evaluation procedure (formal description may be found in Supplementary Materials), which does not allow target leakage and strong bias. For each fold $\mathcal{S}_q$, we compute an optimal stopping moment for cluster i by averaging evaluation metrics for all observations from cluster i that do not belong to fold $\mathcal{S}_q$. More formally, we compute $\boldsymbol{L}_{i,-q}$ as $L_{i,-q}^{(b)} = \frac{\sum_{j\neq q} n_{i,j}\cdot l_{i,j}^{(b)}}{\sum_{j\neq q} n_{i,j}}$, by applying $EvalBestIter$ to all folds except the q-th one (ignoring $\mathcal{S}_q$ from $folds$). After this step, we have $(\hat{B}_1^q, ..., \hat{B}_C^q)$ estimated on $\mathcal{S}_{-q}$. Then we use $\mathcal{S}_q$ as a set validating the quality of predicted $(\hat{B}_1^q, ..., \hat{B}_C^q)$. After averaging the obtained results over folds $\mathcal{S}_q$, we get a more accurate estimation of the quality of clustering, which is used to select the number and size of clusters and to estimate the possible profit of applying adaptive stopping procedure, all this with a minor additional time consumption relative to the training time of the ensemble model. There is still some bias because fold S_q is used both to train models applied to $\mathcal{S}_{-q}$ and to estimate the performance of stopping points. However, the desired property of not using the same set for both tuning and evaluating $\hat{B}$ is satisfied and allows us to get useful estimations.

6 EXPERIMENTS

Table 1: 0-1 loss / logloss, Average value, mean relative error change w.r.t. baseline. Zero-percent improvements mean, that one cluster is the best option selected by validation protocol.

	LightGBM				CatBoost			
Dataset (#samples/#features)	**Unpruned**	**Baseline**	**DSP**	**ISP**	**Unpruned**	**Baseline**	**DSP**	**ISP**
Adult (49K/15)	.1335 / .2944	.1269 / .2753	$.1253/.2726$ $-1.27\%/-0.99\%$	$.1265/.2742$ $-0.30\%/-0.39\%$	.1276 / .2755	.1266 / .2725	$.1253/.2696$ $-1.05\%/-1.07\%$	$.1260/.2717$ $-0.48\%/-0.28\%$
Amazon (33K/10)	.0596 / .1663	.0529 / .1631	$.0525/.1628$ $-0.75\%/-0.21\%$	$.0527/.1629$ $-0.46\%/-0.12\%$	.0445 / .1404	.0448 / .1395	$.0440/.1395$ $-1.68\%/0\%$	$.0444/.1391$ $-0.87\%/-0.29\%$
Click (400K/12)	.1592 / .4067	.1581 / .3964	$.1578/.3964$ $-0.18\%/0\%$	$.1581/.3962$ $0\%/-0.04\%$	.1577 / .3919	.1564 / .3916	$.1563/.3914$ $-0.06\%/-0.04\%$	$.1564/.3914$ $+0.01\%/-0.04\%$
Default (30K/23)	.2011 / .4694	.1888 / .4518	$.1878/.4504$ $-0.51\%/-0.32\%$	$.1862/.4497$ $-1.4\%/-0.47\%$	.1833 / .4361	.1828 / .4328	$.1821/.4324$ $-0.36\%/-0.1\%$	$.1798/.4323$ $-1.63\%/-0.12\%$
HEPMASS (840K/25)	.1337 / .2843	.1270 / .2791	$.1238/.2754$ $-2.55\%/-1.33\%$	$.1267/.2780$ $-0.25\%/-0.39\%$	.1309 / .2803	.1258 / .2768	$.1245/.2753$ $-1.05\%/-0.55\%$	$.1256/.2764$ $-0.17\%/-0.16\%$
Higgs (11KK/28)	.2476 / .5021	.2381 / .4922	$.2369/.4897$ $-0.5\%/-0.51\%$	$.2375/.4906$ $-0.27\%/-0.33\%$	.2385 / .4864	.2364 / .4810	$.2351/.4794$ $-0.57\%/-0.34\%$	$.2361/.4803$ $-0.14\%/-0.14\%$
KDD Churn (50K/231)	.0725 / .2342	.0725 / .2323	$.0725/.2323$ $0\%/0\%$	$.0725/.2323$ $0\%/0\%$	.072 / .2382	.0718 / .2326	$.0715/.2326$ $-0.42\%/0\%$	$.0718/.2326$ $0\%/0\%$
KDD Internet (10K/69)	.0974 / .2334	.0989 / .2202	$.0980/.2167$ $-0.92\%/-1.58\%$	$.0969/.2190$ $-1.98\%/-0.53\%$	.0994 / .2199	.0984 / .2167	$.0969/.2127$ $-1.55\%/-1.85\%$	$.0963/.2152$ $-2.17\%/-0.68\%$
KDD Upselling (50K/231)	.0495 / .1694	.0495 / .1669	$.0492/.1670$ $-0.67\%/+0.08\%$	$.0495/.1669$ $-0.04\%/0\%$	.0487 / .1677	.0489 / .1670	$.0484/.1665$ $-1.01\%/-0.28\%$	$.0488/.1668$ $-0.10\%/-0.13\%$
Kick (73K/36)	.0987 / .3073	.0991 / .2956	$.0987/.2939$ $-0.44\%/-0.56\%$	$.0987/.2949$ $-0.37\%/-0.25\%$	.0953 / .2864	.0951 / .2855	$.0949/.2849$ $-0.22\%/-0.22\%$	$.0948/.2850$ $-0.36\%/-0.16\%$
Marketing (45K/16)	.0941 / .2268	.0931 / .2044	$.0919/.2035$ $-1.25\%/-0.43\%$	$.0931/.2040$ $0\%/-0.20\%$	.0913 / .2032	.0911 / .1938	$.0911/.1925$ $0\%/-0.68\%$	$.0895/.1922$ $-1.78\%/-0.80\%$
Average	-	-	-0.82% / -0.53%	-0.46% / -0.25%	-	-	-0.72% / -0.47%	-0.70% / -0.25%

In this section, we perform numeric experiments, analyze the effectiveness of the proposed framework, and validate statements made in Section 4. We take two popular open–source Gradient Boosting libraries, LightGBM (LightGBM, 2017) and CatBoost (CatBoost, 2017). In accordance with Prokhorenkova et al. (2017), we tune each model using 5-fold cross–validation scheme via 50 iterations of hyperopt (Bergstra et al., 2013) (the set of parameters and the grid may be found in Supplementary Materials). Every hyperopt iteration is followed by a greedy search of the best number of trees (single stopping moment), so we optimize the quality of the pruned ensemble during hyperparameter tuning. Models are tuned with a maximum number of boosting iterations $B = 5,000$. We use the tuned hyperparameters for further experiments, except for the total number of trees, which we set to $B' = 2B = 10,000$ to be more confident that all the models are converged.

To compare the quality of the standard early stopping approach and those proposed in this paper, we hold out 20% of samples from each dataset for the test. The 5–fold stratified cross-validation is utilized to determine the optimal stopping moment. We use the standard pruning algorithm as a baseline and compare it with the methods proposed in Section 4. For the DSP algorithm, we store "sparse" learning histories (as in Section 4.4) for every data point and again perform 5-fold cross-validation (5 clustering tree retrainings, not the whole ensemble) to tune the depth of the the tree and minimal leaf size. In the ISP case, we follow the validation protocol from Section 5.

Datasets used in this investigation and their properties are listed in Table 1, and their links can be found in references. We consider all datasets from (Prokhorenkova et al., 2017) and Ibragimov & Gusev (2019) to compare our method with SOTA GB implementations.

Does the validation protocol proposed in Section 5 have good generalization ability? For this investigation we applied naive validation control, described in Section 5, and advanced evaluation procedure to every dataset. As we can see from Figure 4 and Figure 5 naive validation protocol monotonically decreases with the number of clusters, as was expected, and it gives no insight into the optimal cluster count and possible improvement compared to the baseline. In contrast, the quality estimation produced by the advanced approach is highly correlated with test quality. The quality patterns for test and validation are repeated, and there is an opportunity to make an informed choice of a cluster count and other parameters affecting clustering.

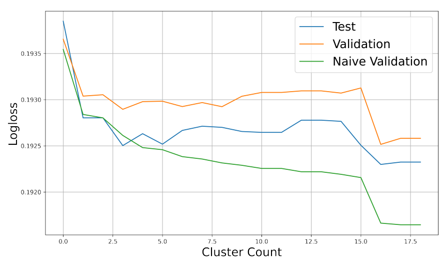

Figure 4: Marketing, validation

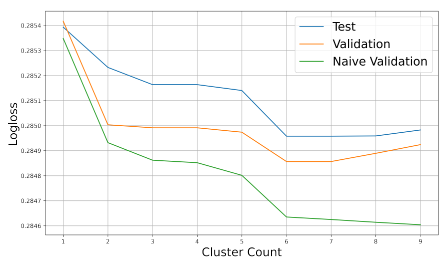

Figure 5: Kick, validation

Does the proposed algorithm help to increase the quality of boosting models? In this paragraph, we carry out an extensive search of the best partition in terms of two loss metrics (lower is better): Logloss and 0-1 loss. For each cluster we get from DSP or ISP algorithm, we find the optimal iteration count and apply the corresponding number of trees (boosters) to each test sample (as in Algorithm 1). The described process is repeated ten times for each dataset with different CV splits (random seeds). We calculate an average loss over all experiment runs and the mean relative improvement (loss decrease). The comparison is presented in Table 1. The results show the superiority of the proposed techniques over the classic early stopping in most settings. The improvements are significant according to paired (ten pairs "baseline vs ours" for each dataset) Wilcoxon signed-rank test, with $p - value \ll 0.001$, except for datasets Click and KDD Churn. Also, it is worth noting that the relative magnitude of improvements is comparable to the one obtained by hyperparameter tuning (50 cross-validated iterations of hyperopt). From this, we can conclude that modern Gradient Boosting implementations do not use the full power of the models, limiting themselves to the shared stopping moment for all examples. At the same time, the personalized selection of this parameter allows significant improvements in the algorithm's performance.

Does it make sense to use unsupervised clustering instead of the one proposed in the paper? In Section 4.3 we discussed the disadvantages of the lack of information about targets when constructing a space partition, as well as the use of algorithms that do not use the geometry of learning surfaces. To test these hypotheses, we conduct the same pool of experiments as described in the previous bullet to Higgs and HEPMASS datasets with the only change of the clustering algorithm to KMeans. The choice of datasets is due to the lack of categorical features. The results show that unsupervised clustering can be effectively applied to the adaptive stopping problem (mean improvements $0.1\%/0.08\%$ and $0.09\%/0.1\%$ respectively), performs significantly better than the standard approach, but significantly worse than the one proposed in Section 4.5 (both due to paired Wilcoxon signed-rank test).

7 Conclusion and Future Work

In this paper, we discovered a problem of ensemble pruning previously uncovered in the literature. We discussed possible problems that the simultaneous stopping rule brings to the modern boosting models and proposed a cluster-based framework of early stopping that can be directly applied to any implementation of Gradient Boosting (and possibly other ensemble methods) without harming its quality and training/inference time. We proposed an evaluation protocol for our method, so it is simple and at the same time computationally cheap to determine whether the adaptive stopping works well for any particular data. Our experiments with the well-known implementation of boosting demonstrate the validity of the assumptions and conclusions made in the paper and great potential for applications and further research since this work still uncovers many problems.

References

Ismail Babajide Mustapha and Faisal Saeed. Bioactive molecule prediction using extreme gradient boosting. *Molecules*, 21(8):983, 2016.

James Bergstra, Dan Yamins, David D Cox, et al. Hyperopt: A python library for optimizing the hyperparameters of machine learning algorithms. In *Proceedings of the 12th Python in science conference*, volume 13, pp. 20. Citeseer, 2013.

Leo Breiman, Jerome H Friedman, Richard A Olshen, and Charles J Stone. *Classification and regression trees*. Routledge, 2017.

CatBoost. Catboost library. `https://github.com/catboost/catboost`, 2017.

George DC Cavalcanti, Luiz S Oliveira, Thiago JM Moura, and Guilherme V Carvalho. Combining diversity measures for ensemble pruning. *Pattern Recognition Letters*, 74:38–45, 2016.

Yuan-Chin Ivan Chang, Yufen Huang, and Yu-Pai Huang. Early stopping in l2boosting. *Computational Statistics & Data Analysis*, 54(10):2203–2213, 2010.

Olivier Chapelle and Yi Chang. Yahoo! learning to rank challenge overview. In *Proceedings of the learning to rank challenge*, pp. 1–24. PMLR, 2011.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pp. 785–794, 2016.

Chen Cheng, Fen Xia, Tong Zhang, Irwin King, and Michael R Lyu. Gradient boosting factorization machines. In *Proceedings of the 8th ACM Conference on Recommender systems*, pp. 265–272, 2014.

Cliff Click, Michal Malohlava, Arno Candel, Hank Roark, and Viraj Parmar. Gradient boosting machine with h2o. *H2O. ai*, 11:12, 2016.

Corinna Cortes, Mehryar Mohri, and Dmitry Storcheus. Regularized gradient boosting. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL `https://proceedings.neurips.cc/paper/2019/file/465636eb4a7ff4b267f3b765d07a02da-Paper.pdf`.

R. M. Cruz, R. Sabourin, G. D. Cavalcanti, and T. I. Ren. Meta-des: A dynamic ensemble selection framework using meta-learning. *Pattern Recognition*, 48:1925–1935, 2015.

Rafael MO Cruz, Robert Sabourin, and George DC Cavalcanti. Dynamic classifier selection: Recent advances and perspectives. *Information Fusion*, 41:195–216, 2018.

Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society: Series B (Methodological)*, 39(1): 1–22, 1977.

Lei Deng, Juan Pan, Xiaojie Xu, Wenyi Yang, Chuyao Liu, and Hui Liu. Pdrlgb: precise dna-binding residue prediction using a light gradient boosting machine. *BMC bioinformatics*, 19(19):135–145, 2018.

Wei Fan, Fang Chu, Haixun Wang, and Philip S Yu. Pruning and dynamic scheduling of cost-sensitive ensembles. In *AAAI/IAAI*, pp. 146–151, 2002.

Yoav Freund and Robert E Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997. ISSN 0022-0000. doi: https://doi.org/10.1006/jcss.1997.1504. URL `https://www.sciencedirect.com/science/article/pii/S002200009791504X`.

Jerome H Friedman. Greedy function approximation: a gradient boosting machine. *Annals of statistics*, pp. 1189–1232, 2001.

Jerome H Friedman. Stochastic gradient boosting. *Computational statistics & data analysis*, 38(4): 367–378, 2002.

Daniel Hernández-Lobato, Gonzalo Martinez-Munoz, and Alberto Suárez. Statistical instance-based pruning in ensembles of independent classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(2):364–369, 2008.

Bulat Ibragimov and Gleb Gusev. Minimal variance sampling in stochastic gradient boosting. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pp. 15087–15097, 2019.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. *Advances in neural information processing systems*, 30:3146–3154, 2017.

Erin LeDell and Sebastien Poirier. H2o automl: Scalable automatic machine learning. In *Proceedings of the AutoML Workshop at ICML*, volume 2020, 2020.

Nan Li, Yang Yu, and Zhi-Hua Zhou. Diversity regularized ensemble pruning. In *Joint European conference on machine learning and knowledge discovery in databases*, pp. 330–345. Springer, 2012.

Ping Li, Qiang Wu, and Christopher Burges. Mcrank: Learning to rank using multiple classification and gradient boosting. *Advances in neural information processing systems*, 20:897–904, 2007.

LightGBM. Catboost library. `https://github.com/microsoft/LightGBM`, 2017.

Xiaoliang Ling, Weiwei Deng, Chen Gu, Hucheng Zhou, Cui Li, and Feng Sun. Model ensemble for click prediction in bing search ads. In *Proceedings of the 26th International Conference on World Wide Web Companion*, pp. 689–698, 2017.

S. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, 28(2): 129–137, 1982. doi: 10.1109/TIT.1982.1056489.

Dragos D Margineantu and Thomas G Dietterich. Pruning adaptive boosting. In *ICML*, volume 97, pp. 211–218. Citeseer, 1997.

Andreas Mayr, Benjamin Hofner, and Matthias Schmid. The importance of knowing when to stop. *Methods of Information in Medicine*, 51(02):178–186, 2012.

Dayvid VR Oliveira, George DC Cavalcanti, and Robert Sabourin. Online pruning of base classifiers for dynamic ensemble selection. *Pattern Recognition*, 72:44–58, 2017.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. *arXiv preprint arXiv:1706.09516*, 2017.

Toby Sharp. Implementing decision trees and forests on a gpu. In *European conference on computer vision*, pp. 595–608. Springer, 2008.

Robin Sibson. Slink: an optimally efficient algorithm for the single-link cluster method. *The computer journal*, 16(1):30–34, 1973.

Víctor Soto, Sergio García-Moratilla, Gonzalo Martínez-Muñoz, Daniel Hernández-Lobato, and Alberto Suárez. A double pruning scheme for boosting ensembles. *IEEE transactions on cybernetics*, 44(12):2682–2695, 2014.

Mervyn Stone. Cross-validatory choice and assessment of statistical predictions. *Journal of the royal statistical society: Series B (Methodological)*, 36(2):111–133, 1974.

Samir Touzani, Jessica Granderson, and Samuel Fernandes. Gradient boosting machine for modeling the energy consumption of commercial buildings. *Energy and Buildings*, 158:1533–1543, 2018.

Ilya Trofimov, Anna Kornetova, and Valery Topinskiy. Using boosted trees for click-through rate prediction for sponsored search. In *In Proceedings of the Sixth International Workshop on Data Mining for Online Advertising and Internet Economy*, pp. 1–6, 2012.

Yuting Wei, Fanny Yang, and Martin J Wainwright. Early stopping for kernel boosting algorithms: A general analysis with localized complexities. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper/2017/file/a081cab429ff7a3b96e0a07319f1049e-Paper.pdf`.

Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. On early stopping in gradient descent learning. *Constructive Approximation*, 26:289–315, 08 2007. doi: 10.1007/s00365-006-0663-2.

Yanru Zhang and Ali Haghani. A gradient boosting method to improve travel time prediction. *Transportation Research Part C: Emerging Technologies*, 58:308–324, 2015.

Yi Zhang, Samuel Burer, W Nick Street, Kristin P Bennett, and Emilio Parrado-Hernández. Ensemble pruning via semi-definite programming. *Journal of machine learning research*, 7(7), 2006.

Zhi-Hua Zhou and Wei Tang. Selective ensemble of decision trees. In *International workshop on rough sets, fuzzy sets, data mining, and granular-soft computing*, pp. 476–483. Springer, 2003.

A Appendix

A.1 Algorithms

Algorithm 2 contains the formal description of an adaptive (cluster-based) optimal stops selection procedure described in Section 4.1 ($EvalBestIter$ from Algorithm 1).

In the Algorithm 3, we describe debiased clustering evaluation procedure we use for ISP hyperparameters tuning (depth of the clustering tree and minimal leaf size).

Algorithm 2 Best Iteration Selection

```
    EvalBestIter($folds, cvPred, partition$) **for** $\mathcal{D}_i \leftarrow partition$ **do**
    $\boldsymbol{L}_i \leftarrow \vec{0}$ {vector of $B$ zeros}
    $n_i \leftarrow 0$
    **for** $\mathcal{S}_j \leftarrow folds$ **do**
      $\mathcal{D}_{i,j} \leftarrow \mathcal{D}_i \cap \mathcal{S}_j$
      $n_{i,j} \leftarrow |\mathcal{D}_{i,j}|$
      $\Delta_{i,j} \leftarrow Eval(cvPred[\mathcal{D}_{i,j}]) \cdot n_{i,j}$
      $\boldsymbol{L}_i \leftarrow \boldsymbol{L}_i + \Delta_{i,j}$ {elementwise vector sum}
      $n_i \leftarrow n_i + n_{i,j}$
    **end for**
    $\boldsymbol{L}_i \leftarrow \boldsymbol{L}_i / n_i$
    $M_i \leftarrow \arg\min \boldsymbol{L}_i$
  **end for**
  **return** $\{M_i\}$
```

Algorithm 3 Evaluation Procedure

```
    Evaluate($folds, cvPred, p = partition$) **for** $\mathcal{S}_q \leftarrow folds$ **do**
    $\{M_i^q\} \leftarrow EvalBestIter(folds \setminus \mathcal{S}_q, cvPred, p)$
    $predictions_q \leftarrow cvPred[\mathcal{S}_q]$
    **for** $\mathcal{D}_i \leftarrow p$ **do**
      Shrink($predictions_q[\mathcal{S}_q \cap \mathcal{D}_i], M_i^q$)
    **end for**
    $\boldsymbol{L}_q = Eval(predictions_q)$
  **end for**
  **return** $Mean(\{\boldsymbol{L}_q\})$
```

A.2 EXPERIMENTAL SETUP

We set the single train/test split in the ratio of 4:1 and use train data for training and hyperparameter search. Test data is used at the evaluation step only. For the initial hyperparameter tuning (baseline), we perform 50 iterations of Tree Parzen Estimator from the Hyperopt library using LogLoss as a target metric (lower-better). We consider the following list of hyperparameters to be tuned.

LightGBM:

- "num_leaves" - max number of terminal nodes. Loguniform grid from 1 to 10^5;
- "learning_rate" - the weight of each tree in the ensemble. Loguniform grid from 10^{-7} to 1;
- "min_data_in_leaf" - minimal number of data points in a leaf (node is not considered for splitting). Loguniform grid from 1 to 10^6;
- "min_sum_hessian_in_leaf" - minimal value of sum of hessians in a leaf (node is not considered for splitting). Loguniform grid from 0 to 10^5;
- "lambda_l1" - regularizing term multiplier for predictions in trees' leaves. Loguniform grid from 0 to 100;
- "lambda_l2" - regularizing term multiplier for predictions in trees' leaves. Loguniform grid from 0 to 100;
- "bagging_fraction" - SGB sampling ratio. Uniform grid from 0.5 to 1;
- "feature_fraction" - feature sampling ratio (random subspace). Uniform grid from 0.5 to 1;
- "n_estimators" - number of trees in the ensemble. Fixed constant 5000. The best value is selected at the end by greedy search.

CatBoost:

- "depth" - max depth of each decision tree. Integer grid from 1 to 6;
- "learning_rate" - the weight of each tree in the ensemble. Loguniform grid from 10^{-5} to 1;
- "random_strength" - regularizing randomized term in the split scoring function. Integer grid from 1 to 20;
- "one_hot_max_size" - use one-hot encoding for categorical features with number of unique values less than given parameter. Integer grid from 0 to 25;
- "l2_leaf_reg" - regularizing term multiplier for predictions in trees' leaves. Loguniform grid from 1 to 10;
- "subsample" - SGB sampling ratio. Uniform grid from 0.5 to 1;
- "rsm" - feature sampling ratio (random subspace). Uniform grid from 0.5 to 1;
- "iterations" - number of trees in the ensemble. Fixed constant 5000. The best value is selected at the end by greedy search.

A.3 CLUSTERING RESULTS

Here we report the results of the clustering algorithm DSP for the train set (used to predict optimal stops in each cluster) and the test set.

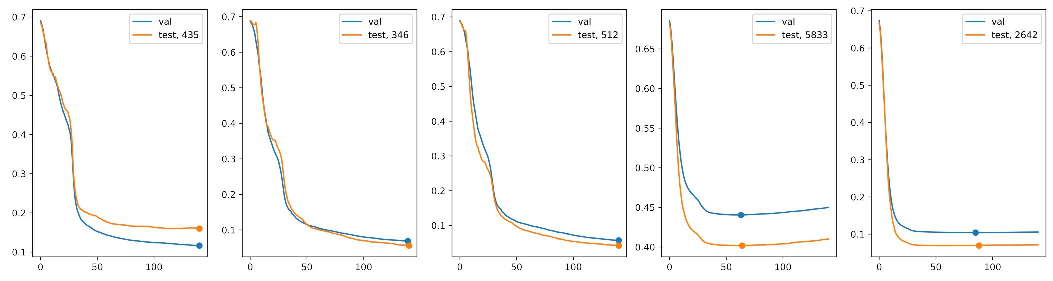

Figure 6: Adult dataset. Clustering obtained via DSP, validation and test results. The number after "test" is test samples count per cluster.)

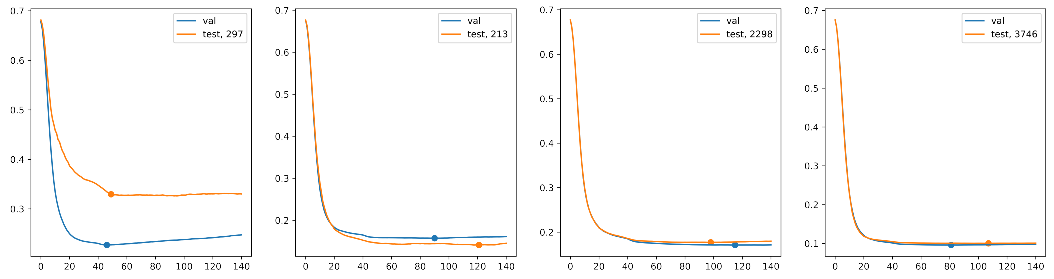

Figure 7: Amazon dataset. Clustering obtained via DSP, validation and test results. The number after "test" is test samples count per cluster.)

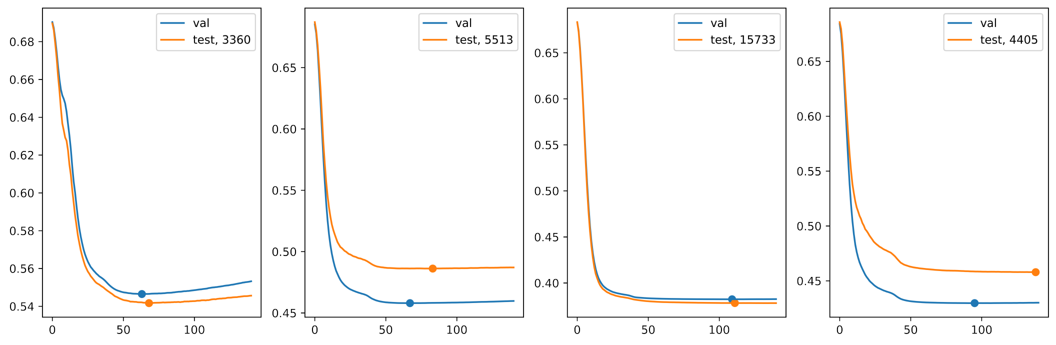

Figure 8: Click dataset. Clustering obtained via DSP, validation and test results. The number after "test" is test samples count per cluster.)

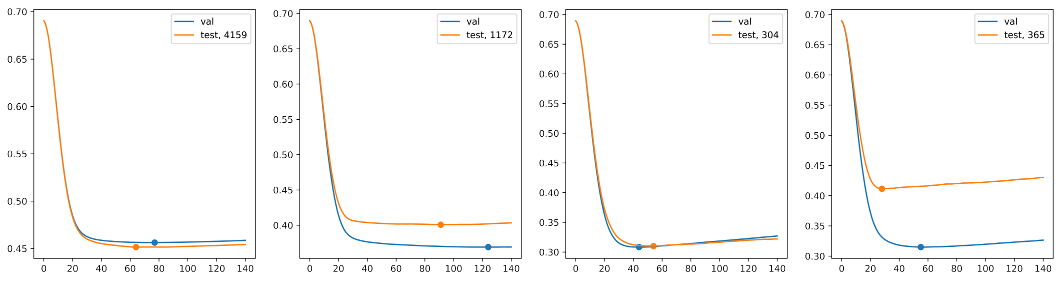

Figure 9: Default dataset. Clustering obtained via DSP, validation and test results. The number after "test" is test samples count per cluster.)

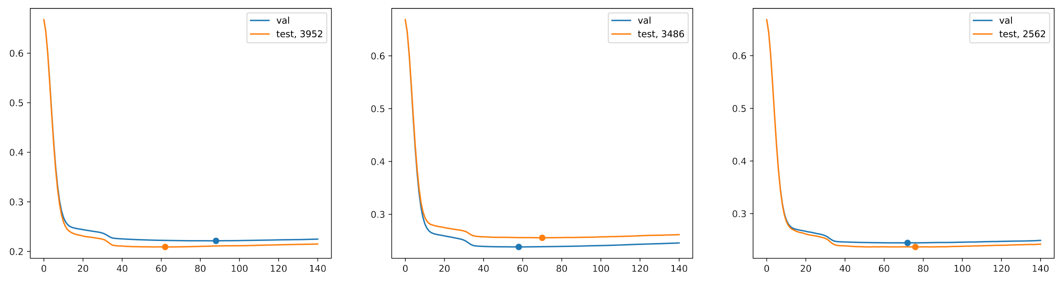

Figure 10: KDD Churn dataset. Clustering obtained via DSP, validation and test results. The number after "test" is test samples count per cluster.)

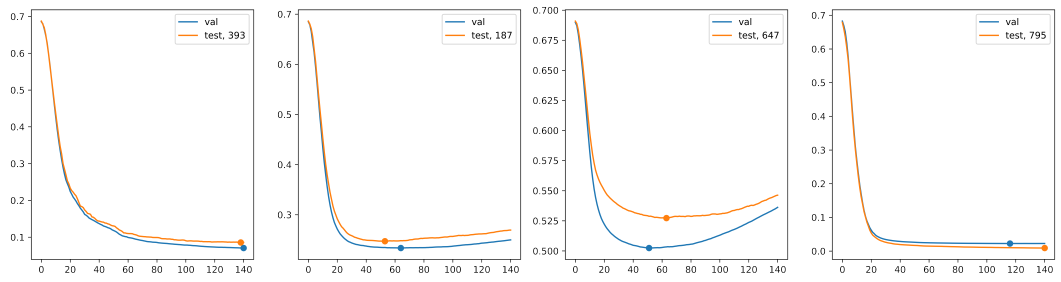

Figure 11: KDD Internet dataset. Clustering obtained via DSP, validation and test results. The number after "test" is test samples count per cluster.)

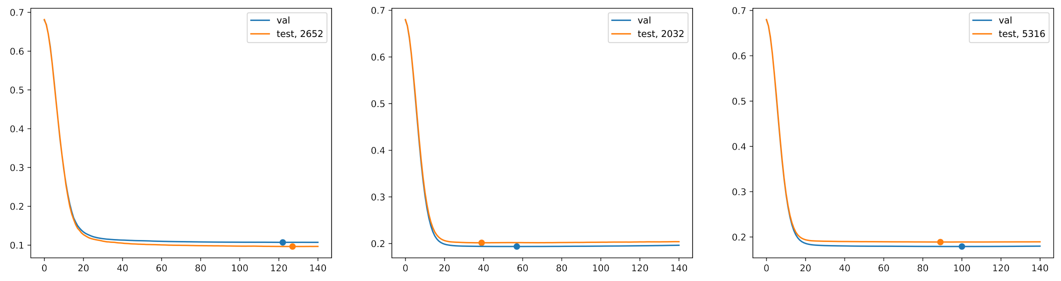

Figure 12: KDD Upselling dataset. Clustering obtained via DSP, validation and test results. The number after "test" is test samples count per cluster.)

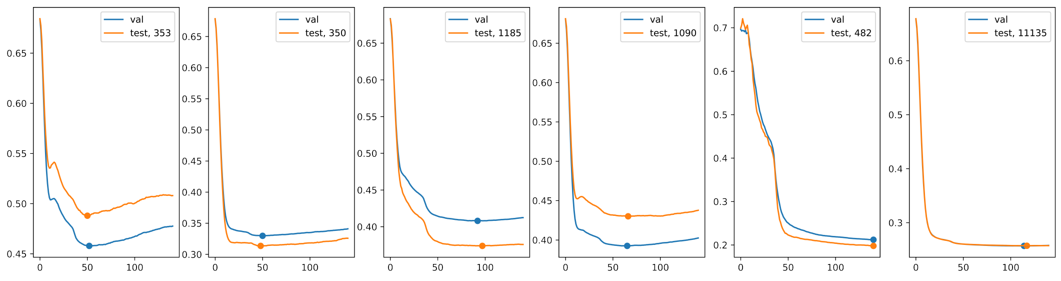

Figure 13: Kick dataset. Clustering obtained via DSP, validation and test results. The number after "test" is test samples count per cluster.)

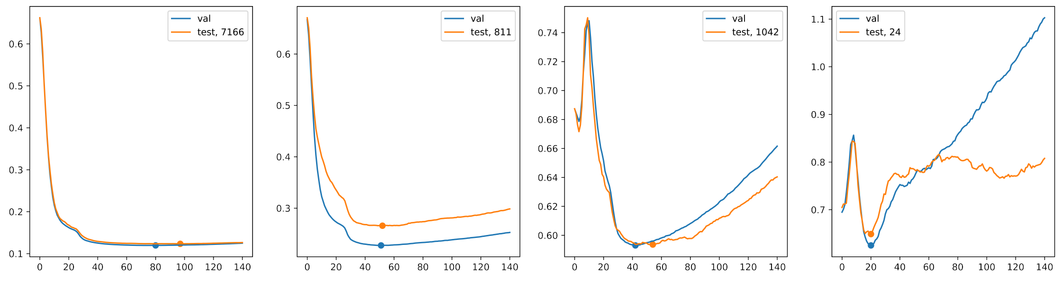

Figure 14: Marketing dataset. Clustering obtained via DSP, validation and test results. The number after "test" is test samples count per cluster.)