DOI: 10.18721/JCSTCS.14301
УДК 004

DEEP GRADIENT BOOSTING FOR REGRESSION PROBLEMS

A.V. Konstantinov

Peter the Great St. Petersburg Polytechnic University,
St. Petersburg, Russian Federation

Deep Forest is a new machine-learning algorithm that combines the advantages of Deep Neural Networks and Decision Trees. It uses representation learning and allows building accurate compositions with a small amount of training data. A significant disadvantage of this approach is the inability to apply it directly to regression problems. First, feature generation method should be determined. Secondly, when replacing classification models with regression models, the set of distinct values of the Deep Forest model becomes limited by the set of values of its last layer. To eliminate the shortcomings, a new model, the Deep gradient boosting is proposed. The main idea is to iteratively improve the prediction using a new feature space. Features are generated based on the predictions of previously constructed cascade layers, by transforming predictions to a probability distribution. To reduce the time of model construction and overfitting, a mechanism for points screening is proposed. Experiments show the effectiveness of the proposed algorithm, in comparison with many existing methods for solving regression problems.

Keywords: regression, Deep Forest, gradient boosting, ensembles, decision trees.

Citation: Konstantinov A.V. Deep gradient boosting for regression problems. Computing, Telecommunications and Control, 2021, Vol. 14, No. 3, Pp. 7–19. DOI: 10.18721/JCST-CS.14301

ГЛУБОКИЙ ГРАДИЕНТНЫЙ БУСТИНГ ДЛЯ РЕШЕНИЯ ЗАДАЧ РЕГРЕССИИ

А.В. Константинов

Санкт-Петербургский политехнический университет Петра Великого,
Санкт-Петербург, Российская Федерация

Глубокий лес является новым подходом машинного обучения, сочетающим преимущества глубоких нейронных сетей и деревьев решений. Он использует обучение представлениям и позволяет строить точные композиции при условии небольшого количества обучающих данных. Существенный недостаток данного подхода – невозможность напрямую применить его к задачам восстановления регрессии. Во-первых, требуется определить способ генерирования набора признаков. Во-вторых, при замене классификационных моделей на регрессионные, множество значений модели Глубокого леса становится ограниченным множеством значений последнего слоя. Для устранения недостатков предложена новая модель Глубокого градиентного бустинга. Основная идея состоит в итеративном улучшении предсказания, с использованием нового пространства признаков. Генерирование признаков производится на основании предсказаний ранее построенных моделей, путём трансформации их к распределению вероятностей. Для снижения времени построения модели и переобучения предложен механизм раннего отсева точек. Эксперименты показывают эффективность предложенного алгоритма по сравнению со многими существующими методами решения задачи регрессии.

Ключевые слова: регрессия, Глубокий лес, градиентный бустинг, ансамбли, деревья решений.

Ссылка при цитировании: Konstantinov A.V. Deep gradient boosting for regression problems // Computing, Telecommunications and Control. 2021. Vol. 14. No. 3. Pp. 7–19. DOI: 10.18721/JCSTCS.14301

Introduction

Over the past few years, one of the most important developments in the field of ensemble methods, that is, methods for constructing model compositions [1–4], has been a combination of ensemble-based models, including Random Forest [5] and stacking [6], proposed by Zhou and Feng and called Deep Forest, or gcForest [7]. The Deep Forest structure consists of layers similar to multilayer neural networks, but each gcForest layer contains several Random Forests, instead of perceptrons. As it was noted [7], Deep Forest is much easier to train, and it also has the ability to obtain high-quality results with extremely small data sizes available for training unlike Deep Neural Network, which requires a lot of effort to select hyperparameters and large data sizes for training. A lot of numerical experiments provided in [7] show that Deep Forest is superior to many widely used methods, or demonstrates results similar to existing methods when using default parameters.

One of the main advantages of Deep Forest is the ability to build deep models using representation learning without differentiable modules, that is without using reverse error propagation. Representation training allows you to identify complex dependencies in the data, and Deep Forest is built in such a way that a small amount of training sample is sufficient for training. For this reason, many modifications of Deep Forest have been developed, as well as adaptations to specific application tasks. In particular, Wang et al. [8] suggested using Deep Forest for predicting health status, or diagnostic health monitoring. Yang et al. [9] used Deep Forest to solve the problem of finding ships using thermal maps. Zheng et al. [10] considered the application of Deep Forest to the pedestrian detection problem.

The Deep Forest structure is cascaded [7]. Each layer of the cascade is an ensemble of decision trees. This structure embodies the idea of learning representations: the feature space changes from layer to layer, creating an increasingly informative representation of the input data. Each level of the cascade, or layer, receives a set of features generated by the previous layer, and the features obtained at the output of the layer are transmitted to the next level. The cascade architecture is shown in Fig. 1 (RF in the Figure is a random forest, and ET is Extra Trees, extremely randomized trees [11]). The diagram shows that each level of the cascade consists of two different pairs of random forests that generate probability distributions of classes (dimension 3), concatenated together with the original input. It is important to note that the structure of the layers can be modified to improve Deep Forest when solving problems of a specific type. The predictions of the last layer contain a set of probability distributions of classes, from which you can get the final prediction by averaging. Deep Forest's ability to learn representations can be supplemented by the second part of Deep Forest, called multi-grained scanning, which allows processing data with a spatial structure. Multi-level scanning uses the sliding window principle to process the original features. The result of such a scan is new feature vectors corresponding to windows of various sizes. Moreover, the scanning itself is performed using a set of decision trees trained with the original set of classes. In this paper, the first part of Deep Forest is considered, since the tasks under consideration do not require the use of multi-level scanning, which significantly affects the results in such areas as image or sound processing. For the input data instance, each forest in the layer determines an estimate of the probability distribution of classes by counting the frequencies of different classes for data instances that fell into the same sheet as this instance during training. After that, the overall estimate of the probability distribution of classes is calculated as the average of the estimates of each decision tree included in the random forest. The probability distribution of classes is represented by a vector, which is then concatenated with the original feature vector in order to use it as an input feature vector at the next stage. The use of the probability distribution vector for constructing

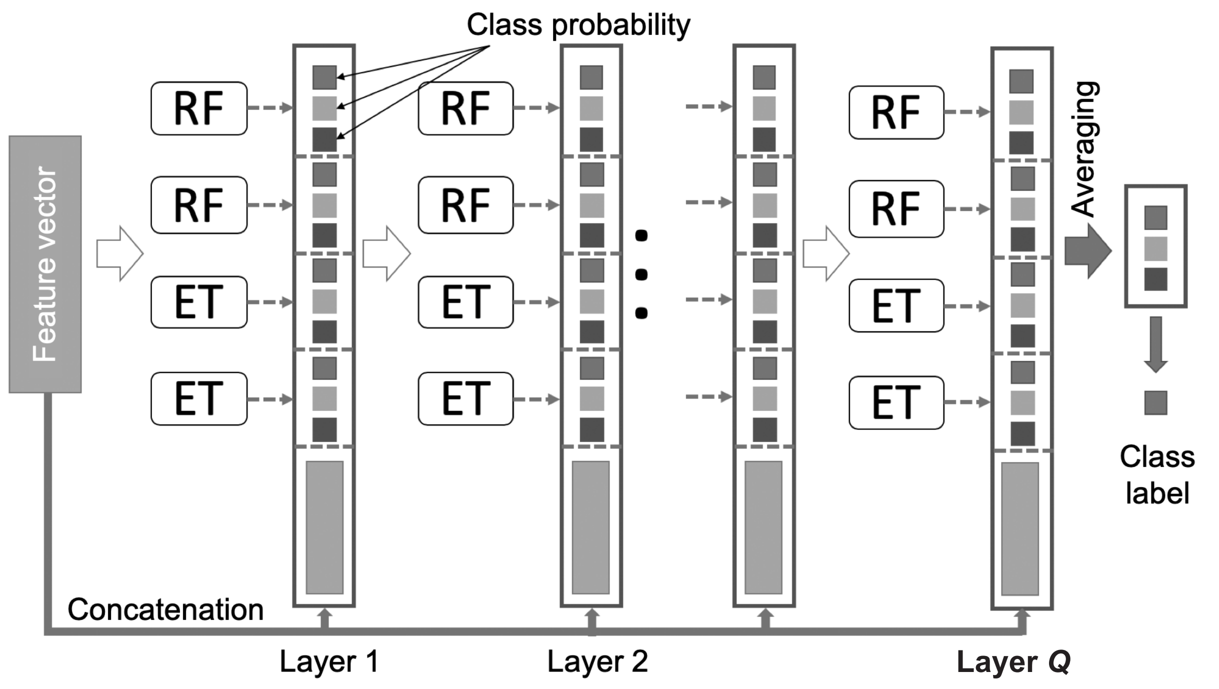

Fig. 1. Deep Forest cascade scheme

the next layer is similar to the idea of stacking [6]: in it, the basic models of the first layer are built using the original data set. After that, the stacking algorithm builds a new data set for training the second layer (meta-model), using the output vectors of the classes of the first layer as input features, and the same values are used as labels (values of the target variable) as for training the first layer. Also, the stacking algorithm often includes a cross-validation procedure to increase the generalizing ability when constructing the second layer. Unlike the standard stacking algorithm, Deep Forest uses both the original feature vector and the class vectors obtained from the previous layer on each layer. Zhou and Feng [7] also suggest using random forests of various types to provide a greater variety of predictions and create more informative features for the next layer.

Deep Forest has a significant shortcoming in comparison with neural networks: it is incapable of using arbitrary differentiable loss functions for any problems other than the standard classification one. In particular, traditional Deep Forest fails to solve such problems as transfer learning and distance metric learning. However, paper [12] eliminates this shortcoming by introducing weights allocated to decision trees, which allows the authors to solve a considerably wider range of problems with the help of the modified Deep Forest. Introduction of the weights for data instances similar to AdaBoost algorithms [13] together with the confidence screening mechanism allows both increasing the accuracy of classification and decreasing the time of building Deep Forest in a number of cases [15]. The AdaBoost algorithm [13] is based on the idea of boosting–iterative model construction as a combination of basic models, so that each subsequent basic model compensates for the shortcomings of the model in the previous iteration. According to [13], the training examples with the largest error in the previous iteration should correspond to large weights, but this approach is not the only possible one for applying the idea of boosting. Thus, one of the most popular approaches over the past decade is gradient boosting and its modifications [16]. Its key feature is the application of the principles of gradient descent in the functional space [17]. Each subsequent basic model is constructed in such a way as to approximate the value of the derivative of the loss function multiplied by minus one. Adding the predictions of the new base model to the already constructed combination corresponds to one step of gradient descent. Moreover, if the training example has an error close to the minimum, the partial derivative will be close to zero, which is similar to the low weight of the training example in AdaBoost. Gradient boosting of decision trees is one of the most commonly used approaches for solving regression recovery problems with tabular data. The disadvantages of gradient boosting include the impossibility of parallel construction of ensemble elements, which leads to low performance with a large number of iterations, as well as a set of values determined by the training set in the leaves of each individual decision tree, which significantly limits the set of values of the entire ensemble. However, in [18], an algorithm for randomized gradient boosting (hereinafter referred to as RMGB) was proposed, based on partially random decision trees. This approach allows you to get a much larger set of values of the trained model, as well as improve performance by reducing the time of building each individual decision tree.

Despite Deep Forest's great success in solving the classification problem, the cascade structure of models based on decision trees has not previously been used to solve the regression recovery problem, where the target variable has a continuous distribution. There are approaches that combine the principles of building neural networks and decision trees [19, 20]. Thus, in [19], an approach is described that allows the construction of regression models based on decision trees, in combination with convolutional neural networks, applied to the problem of determining the age from a photo. Its key disadvantage is the use of training the entire model using error back propagation, which greatly distinguishes this approach from Deep Forest, which has the advantages described above, in comparison with neural networks. Also, the approach proposed in [20], although it combines decision trees with neural networks, does not rely on a cascade structure, the construction of which can be performed iteratively without an error back propagation algorithm.

This paper discusses the problems of adapting Deep Forest to the regression problem, including the need to determine the method of combining predictions to obtain a new set of features, as well as the complexity of constructing functions, the set of values of which is quite large. Next, we propose a new approach based on the ideas of gradient boosting, as well as a method for generating a set of informative features. It allows you to build models using representation learning, without relying on an algorithm for back propagation of the error, obtaining arbitrarily accurate approximations of the original dependencies. To reduce the number of calculations and retraining, a mechanism for early elimination of points for which the predictions are already sufficiently accurate is also proposed.

Deep Forest

The main idea of a Deep Forest for the classification problem is to iteratively construct a set of layers, where each layer relies both on the original set of features and on an intermediate representation obtained from the previous layer. This approach allows you to increase the accuracy of the model simultaneously with the generalizing ability, while maintaining a low level of retraining. Let us imagine a deep forest as a composition of L layers S_i:

$$gcForest_L\left(x\right)=\left(S_L\circ S_{L-1}\circ\ldots\circ S_1\right)\left(x\right).$$

Each layer can consist of arbitrary models that can be used to estimate the probabilities of classes. Namely, instead of predicting the class number, such models should be able to estimate the probability distribution of classes, such as, for example, compositions of decision trees: decision trees, random forests and extremely randomized trees. Indeed, the leaves of decision trees contain vectors consisting of the class frequencies of all elements of the training sample that fell into the corresponding leaf, and the compositions of decision trees allow us to estimate the probability distribution by averaging the class frequency vectors over all decision trees. The predicted frequency vectors of classes of different M_j^i compositions included in one layer of width W are concatenated into one vector, along with the original set of features:

$$S_i(\zeta)=\left(x,\left|M_1^i(\zeta)\right|,\dots,\left|M_W^i(\zeta)\right|\right)^{\mathrm{T}}.$$

Each new layer either allows you to improve the representation of features from the previous layer, or has comparable performance. In the first case, the next layer is built in such a way that the target variable, that is, the class, is predicted from the set of features from the previous layer. In the second case, the composition is stopped. At the same time, a validation set is used to determine the quality of the representation of features.

Applicability of Deep Forest to the regression problem

This approach works well for classification, since the estimates of the probability distribution of classes contain significantly more information than the numbers of the predicted classes. And even the probability distribution, with the help of which it is impossible to make a correct prediction of the class, can be an informative set of features for constructing the next layer. Accordingly, this approach cannot be applied directly to the regression problem, for the following reasons. First, the complexity of generating a more suitable feature space for solving the problem, since it is impossible to directly calculate an analog of the class frequency vector for the regression problem, as in the case of the classification problem. Secondly, in addition to the representation of features suitable for solving the problem, in the case of regression, the ability to predict a large number of different values is also required, that is, the resolution, which for the entire Deep Forest will be limited by the resolution of the last layer. Thus, a Deep Forest in which classification trees are replaced with regression trees does not allow you to get better results than a random forest. Also, reducing the regression problem to classification will not allow achieving a higher quality of the model using Deep Forest, since such a reduction similarly reduces the resolution.

Gradient boosting based on Deep Forest

To solve the described problems, two significant changes are made to the Deep Forest model. First, each layer should consist of strong regression models, that is, models that allow you to approximate arbitrary functions with a given accuracy. For this purpose, random forests have been replaced with gradient boosting models of decision trees (hereinafter referred to as MGB). Secondly, the process of constructing a Deep Forest, as well as the process of deriving predictions, should be changed so that each next layer clarifies the general prediction of the model, and does not build it anew. The general model is built according to the principle of gradient boosting, that is, so that each next layer predicts additional terms, which add up to the estimate of the target variable. Let us imagine a new model of Deep gradient boosting from L layers, as:

$$DBF_L(x)=DBF_{L-1}(x)+\gamma M_L\left(x,F_0(x),\dots,F_{L-1}(x)\right),$$

where M_i – model of the i layer; γ – learning rate; F_i – algorithm for generating features in layer i. The scheme of the Deep gradient boosting model is shown in Fig. 2.

Let the loss functional l be given and a Deep gradient boosting model with an $(L-1)$ layer is already constructed. To minimize the loss functional, the next layer is constructed in such a way as to approximate the residual terms for each point of the training set:

$$r_i^{(L)}=\left.\left|-\frac{\partial l(z,y_i)}{\partial z}\right|\right._{z=DBF_{L-1}(x_i)}.$$

As an input, similar to a deep forest, the next layer uses a set of features from the previous layers obtained using a feature generation algorithm:

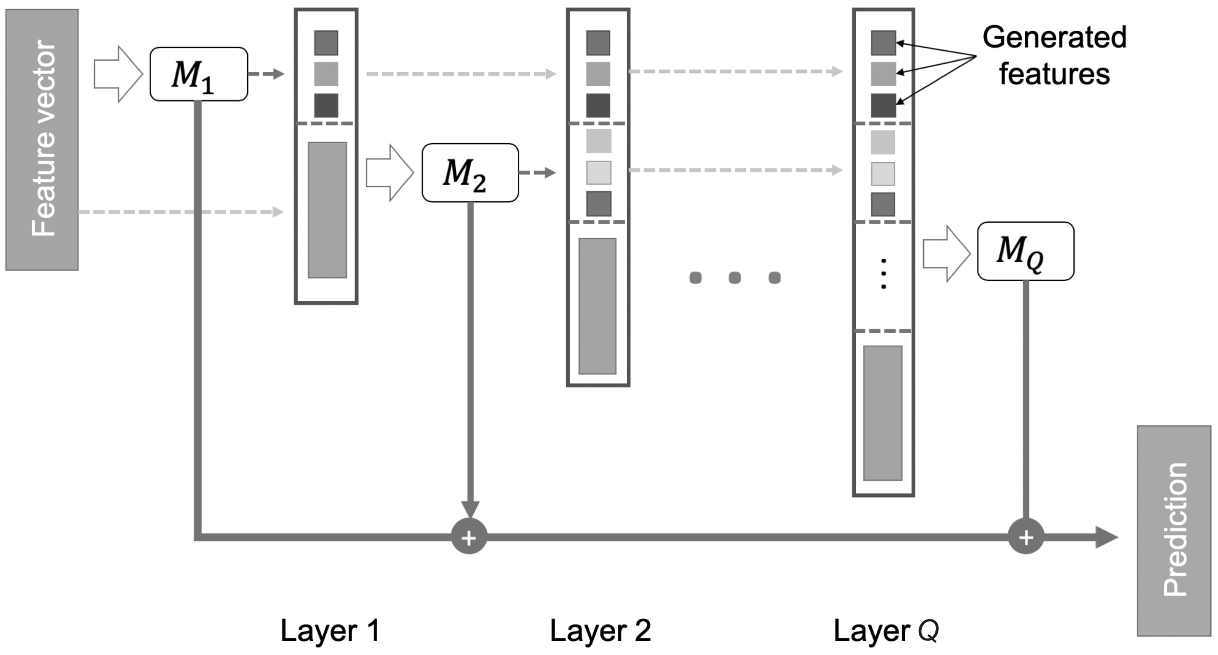

Fig. 2. Deep gradient boosting model scheme

$$I_M = \left(x_i, F_0\left(x_i\right), \ldots, F_{M-1}\left(x_i\right)\right).$$

MGB M_L is trained to predict the residual terms from the generated features.

Thus, the general algorithm for constructing a Deep boosting model consists of the following steps:

1. Building a zero layer, like an MGB composition, namely training several different MGB.
2. Calculation of the residual terms of r_i – partial derivatives of the error function.
3. Generating a set of features based on predictions.
4. Formation of a new feature space: concatenation of the generated feature vectors with the feature vectors of the previous layers.
5. Construction of a new layer approximating the residual terms using the resulting feature space.
6. If the stop condition is not met, go to step 2.

This algorithm is general and leaves it possible to determine the algorithm for generating features and the stop condition. So, if you omit the feature generation procedure (that is, set a feature generation algorithm that matches zero-length vectors to predictions), and choose a decision tree construction algorithm as the algorithm for constructing a new layer, you can get a traditional gradient boosting algorithm for decision trees. The stop condition can be set as a condition for the number of layers M to exceed a certain constant. You can also set a threshold for the value of the empirical risk functional, after which the algorithm is stopped, or use a plateau detection criterion: during a certain number of iterations, the fluctuation of the empirical risk functional (the difference between the maximum and minimum values) should not exceed the constants. Both of the latter criteria can also be applied with a validation sample – a separate data set used for quality assessment that does not overlap with the training set.

The predictions of the MGB layer are used as the generated features, while the cumulative prediction of the entire composition is not taken into account. The features generated by layer i are defined as:

$$F_i\left(x\right) = \left[M_i^1\left(x, F_0\left(x\right), \ldots, F_{i-1}\left(x\right)\right), \ldots, M_i^K\left(x, F_0\left(x\right), \ldots, F_{i-1}\left(x\right)\right)\right]^{\mathrm{T}}.$$

By analogy with a deep forest, each layer uses not one MGB, but a composition of independent models M_i^j, in order to obtain a more informative feature space. The layer prediction is calculated by averaging the predictions of the composition models. In this case, the method of generating features is based on the predictions of a set of composition models. The layer prediction is calculated as:

$$P_i(t) = \frac{1}{K}\sum_{j=1}^{K} M_i^j(t).$$

Potentially, the disadvantage of such formation of new features is the inability to build a decisive rule that splits a set of data according to the criterion of consistency of model predictions in a layer, as well as a high correlation of features. To eliminate these shortcomings, it is possible to transform the obtained features into an analog of the class frequency vector. For the convenience of the presentation, without detracting from the generality, we will assume that the value of each feature lies in the interval [0, 1], and both boundaries of the interval are reached. If this is not the case, the values of all features should be pre-normalized, independently of each other at the model construction stage, and at the prediction stage, the same constants should be used for normalization as during training. The interval is divided into C disjoint half-intervals of equal length. Each prediction of the layer model is associated with a vector of C components:

$$u_i^j(x) = \left\langle 1 - \left| \left(F_i(x)\right)_j - \frac{t}{C} \right| \right\rangle_{t=1}^{C}.$$

The vectors corresponding to each layer model are combined by averaging:

$$u_i(x) = \frac{1}{K}\sum_{j=1}^{K} u_i^j(x).$$

A normalized exponential function (SoftMax) with temperature θ is applied to the resulting vector, calculated as:

$$\left(\sigma(z,\theta)\right)_i = \frac{\exp\left({}^{z_j}\!/_{\theta} \right)}{\sum_j \exp\left({}^{z_j}\!/_{\theta} \right)}.$$

Thus, the obtained analog of the probability vector of classes:

$$\hat{u}_i(x) = \sigma\left(u_i(x)\right).$$

The sum of all the elements of the vector $\hat{u}_i(x)$ will be equal to 1, and the value of each element will lie in the interval [0, 1]. If the predictions of the layer models are equal, the element of the vector $\hat{u}_i(x)$ corresponding to the predictions has the maximum value. By varying the temperature θ, it is possible to reduce or increase the influence of the predictions of individual models of the layer: the higher the temperature, the closer the elements of the vector are to $\frac{1}{K}$. The lower the temperature, the closer the value of one of the elements of the vector is to 1. The obtained representation of the features allows us to build decisive rules based on both the predicted values and the consistency of the layer models.

Let's pay attention to the physical meaning of the functions that are approximated by each layer: partial derivatives of the loss function. If the loss function is convex, then if its partial derivative at point x is zero, then the minimum of the loss function is reached at x. This means that no further iterations are required for the set of all such points Q. Each next layer of the model will need to predict values close to zero for Q points. If the predictions are different from zero, the difference will need to be compensated by the next layer after it. Thus, firstly, extra work will be performed for a certain set of points, and, secondly, the optimal prediction obtained at a certain iteration can be further worsened by adding small terms with an error. To eliminate these shortcomings, we will filter out data instances based on the absolute value of predictions. Let's consider two ways to filter out data instances: by a fixed threshold value d, and by the quantile of the empirical distribution of the absolute values of the predictions of the layer. For the first method, on each layer, we will determine the set of points for dropping out layer i, depending on the input features I_j, as:

$$Q_i = \left\{ I_j : \left| P_i\left(I_j\right) \right| < d \right\}.$$

The difficulty of using this approach is that it is impossible to determine the number of points to be eliminated in advance, and also with a fixed value of d, the result strongly depends on the scale of the data. However, the threshold value can be selected automatically and individually for the layer. To do this, we introduce the parameter p – the proportion of points that need to be filtered out on each layer and find the quantile of the empirical distribution of the absolute values of the predictions:

$$d_i = \arg\max_d \left\{ \# \left\{ \left(I, r\right) \in D^{(i)} : \left| P_i\left(I\right) \right| < d \right\} \le p \cdot \# D^{(i)} \right\}.$$

The set of points for dropping out, that is those that will not be used for further construction of the model, is located similarly. This approach will allow you to leave a known amount of data for training at each step in advance. Too large, close to one, p values can lead to a situation in which the size of the training set will be insufficient to build an accurate model, and / or an increase in the value of the risk functional, due to too early sifted points.

In order to avoid situations in which points are eliminated too early, that is, inflated d_i values, it is possible to adjust the threshold values based on the known values of the residuals. To do this, we will introduce an additional threshold value d_i', but we will calculate the quantile of the empirical distribution of the absolute values of the residuals, instead of predictions:

$$d_i' = \arg\max_d \left\{ \# \left\{ \left(I, r\right) \in D^{(i)} : \left| r \right| < d \right\} \le \eta \cdot \# D^{(i)} \right\}.$$

The adjusted threshold value will be calculated as:

$$\hat{d}_i = \min\left\{ d_i, d_i' \right\}.$$

Thus, it remains possible to control the upper bound of the number of points that need to be eliminated on each layer, but if the model's prediction at a certain point is small due to an error, such a point will not affect the value of $\hat{d}_i$.

The choice of parameters of the basic models when constructing the composition has an extremely strong influence on the accuracy of the aggregate model. For example, if the approximate function depends on four features, and the maximum depth of the decision trees used in MGB is two, then it will be impossible to build an exact approximation using such an MGB. If the depth of the trees is too large, there may be a negative effect of retraining. This means that it is necessary to select the most suitable parameters of the model. Since each layer solves its own separate machine-learning task (both the approximate function and the set of features are different for each layer), the most suitable parameters should be selected for each layer separately. To do this, we use a search of parameter values with cross-validation: a grid of parameters is set, including the maximum depth of the tree, the number of MGB iterations and the learning rate, and a combination of parameters is selected that gives the smallest error of cross-validation (cross-validation). All the basic models in the layer are trained using the locally obtained optimal parameters. Note that a similar improvement can also be applied to the classical gradient boosting algorithm for decision trees, but it is more justified in the case of the proposed deep gradient boosting, since it changes not only the objective function, but also the set of features at each iteration.

Experiments

For comparison with existing MGB and random forests, implementations from the "Scikit-Learn" package were used. To assess the quality of predictions in each experiment, the data were divided into two groups randomly: the training set (75 % of the original sample) and the test set (25 % of the original sample). Next, training was performed on the training set and quality assessment was performed on the test set. For each method, the experiment was repeated 100 times. Since MGB and random forest require the selection of parameters, such as the depth of trees, a procedure for automatically selecting the most suitable parameters with cross-checking was used. To do this, the training set was divided into 5 parts and such parameters were selected for which the average accuracy during cross-checking was maximum. After that, the model was built anew with the selected parameters using the entire training set. For a random forest, the maximum depth of the decision tree was chosen – an integer value from 2 to 7, or an unlimited depth of the tree, as well as the number of decision trees: 100 or 1000. For MGB, the parameters were selected in the same way, with the addition of the learning rate: 0.1, or 0.01. For deep boosting, the optimal values were not selected using cross-checking in advance.

To compare the methods on regression problems, the following data sets were used (Table 1):

• from the "StatLib" repositories and the "R"package: "ML housing data set", "California housing data set", "Diabetes", "Longley's Economic Regression Data";

• from the Kaggle online data analysis competition platform: "House Prices: Advanced Regression Techniques", based on the "Ames Housing data set", with a large number of categorical features;

• simulated (artificially generated) data sets: "Friedman 1", "Friedman 2", "Friedman 3", described in detail in [5]. And also "Regression" and "Sparse uncorrelated" from the "Scikit-Learn" package.

In Table 1, m denotes the number of features, and n is the number of observations.

On regression problems, the algorithms were compared: random forest, MGB, and the proposed Deep gradient boosting (with MGB on random decision trees). The standard error was used as the risk functional for all methods. Table 2 shows the values of standard errors (less – better) averaged over 100 experiments for data sets from repositories, and Table 3 – for artificial data sets, for:

• a random forest with the above-described method for selecting optimal hyperparameters (in the Table – the RF);

• extremely randomized trees with the above-described method of selecting optimal hyperparameters (in the Table – ERT);

• the classical model of gradient boosting with the above-described method of selecting optimal hyperparameters (in the Table – MGB);

• CatBoost gradient boosting models (in the Table – CatBoost);

• models of extreme gradient boosting XGBoost with the above-described method of selecting optimal hyperparameters (in the Table – XGBoost);

• the proposed Deep gradient boosting based on RMGB (in the Table – Deep RMGB);

• the proposed Deep gradient boosting based on extremely randomized trees (in the Table – Deep ERT).

As can be seen, the proposed algorithms allow us to build models with comparable or smaller root-mean-square errors than classical methods, as well as advanced widely used improvements to gradient boosting algorithms, such as CatBoost and XGBoost. The proposed deep gradient boosting RMGB in

Table 1

Description of regression data sets

Title	Abbreviated designation	*m*	*n*
California housing data set	California	8	20640
House Prices: Advanced Regression Techniques	HouseART	79	1460
ML housing data set	Boston	13	506
Diabetes	Diabetes	10	442
Longley's Economic Regression Data	Longley	7	16
Friedman 1	Friedman 1	10	100
Friedman 2	Friedman 2	4	100
Friedman 3	Friedman 3	4	100
Scikit-Learn Regression	Regression	100	100
Scikit-Learn Sparse uncorrelated	Sparse	10	100

Table 2

Standard error on regression problems

Method	California	HouseART	Boston	Diabetes	Longley
RF	2.559×10^{-1}	9.796×10^{8}	1.215×10^{1}	3.367×10^{3}	1.055×10^{0}
ERT	2.493×10^{-1}	9.645×10^{8}	1.136×10^{1}	**3.235×10^{3}**	9.666×10^{-1}
МГБ	2.083×10^{-1}	9.533×10^{8}	**1.052×10^{1}**	3.292×10^{3}	8.660×10^{-1}
CatBoost	2.197×10^{-1}	**9.148×10^{8}**	1.066×10^{1}	3.547×10^{3}	1.273×10^{0}
XGBoost	2.057×10^{-1}	9.835×10^{8}	1.124×10^{1}	3.276×10^{3}	1.020×10^{0}
Deep RMGB	**1.986×10^{-1}**	9.157×10^{8}	1.119×10^{1}	3.343×10^{3}	**8.145×10^{-1}**
Deep ERT	2.472×10^{-1}	9.220×10^{8}	1.082×10^{1}	3.315×10^{3}	9.045×10^{-1}

Table 3

Standard error on artificial regression problems

Method	Friedman 1	Friedman 2	Friedman 3	Regression	Sparse
RF	1.062×10^{1}	5.839×10^{3}	2.075×10^{-2}	1.254×10^{4}	2.794×10^{0}
ERT	8.869×10^{0}	1.475×10^{3}	1.575×10^{-2}	1.162×10^{4}	2.203×10^{0}
MGB	7.233×10^{0}	5.240×10^{3}	1.877×10^{-2}	9.929×10^{3}	1.952×10^{0}
CatBoost	8.485×10^{0}	1.008×10^{4}	2.871×10^{-2}	1.238×10^{4}	2.641×10^{0}
XGBoost	7.066×10^{0}	5.340×10^{3}	1.927×10^{-2}	**9.797×10^{3}**	2.050×10^{0}
Deep RMGB	**4.039×10^{0}**	**1.234×10^{3}**	**8.370×10^{-3}**	1.009×10^{4}	**1.238×10^{0}**
Deep ERT	7.916×10^{0}	1.490×10^{3}	1.326×10^{-2}	1.141×10^{4}	1.779×10^{0}

more than half of the cases (on the considered data sets) allows us to get better results than other algorithms. The lower accuracy of Deep gradient boosting ERT, compared to Deep gradient boosting RMGB, is consistent with the assumptions about the need to use MGB in deep boosting layers, instead of a random forest or ERT.

Conclusion

The algorithm of Deep gradient boosting is proposed, which allows to build deep models without using the algorithm of back propagation of the error for solving regression problems. It combines elements of representation learning characteristic of neural networks, due to which more accurate models are built in comparison with traditional gradient boosting and random forest, while maintaining the advantages of decision tree compositions, such as the ability to build models on small training sets and the ability to process categorical features. The described approach to generating features allows you to create more informative representations of input data improving the quality of the model. Experiments conducted on known regression problems and artificial data show the advantages of the proposed algorithm in comparison with existing algorithms.

This work is supported by the Russian Science Foundation under grant 21-11-00116, https://rscf.ru/en/project/21-11-00116/

REFERENCES

1. **Ferreira A.J., Figueiredo M.A.T.** Boosting algorithms: A review of methods, theory, and applications. *Ensemble Machine Learning*. Springer, Boston, MA, 2012, Pp. 35–85.

2. **Rokach L.** Ensemble-based classifiers. *Artificial Intelligence Review*, 2010, Vol. 33, No. 1-2, Pp. 1–39.

3. **Woźniak M., Graña M., Corchado E.** A survey of multiple classifier systems as hybrid systems. *Information Fusion*, 2014, Vol. 16, Pp. 3–17.

4. **Zhou Z.-H.** *Ensemble methods: Foundations and algorithms*. CRC Press, Boca Raton, 2012.

5. **Breiman L.** Bagging predictors. *Machine Learning*, 1996, Vol. 24, No. 2, Pp. 123–140.

6. **Wolpert D.H.** Stacked generalization. *Neural Networks*, 1992, Vol. 5, No. 2, Pp. 241–259.

7. **Zhou Z.-H., Feng J.** Deep Forest: Towards an alternative to Deep neural networks. *Proceedings of the 26th International Joint Conference on Artificial Intelligence* (*IJCAI'17*), AAAI Press, Melbourne, Australia, 2017, Pp. 3553–3559.

8. **Wang C., Lu N., Cheng Y., Jiang, B.** Deep Forest based multivariate classification for diagnostic health monitoring. *2018 Chinese Control and Decision Conference* (*CCDC*), IEEE, 2018, Pp. 6233–6238.

9. **Yang F., Xu Q., Li B., Ji Y.** Ship detection from thermal remote sensing imagery through region-based Deep Forest. *IEEE Geoscience and Remote Sensing Letters*, 2018, Vol. 15, No. 3, Pp. 449–453.

10. **Zheng W., Cao S., Jin X., Mo S., Gao H., Qu Y., Zhu Y.** Deep Forest with local experts based on elm for pedestrian detection. *Pacific Rim Conference on Multimedia*, Springer, Cham, 2018, Pp. 803–814.

11. **Geurts P., Ernst D., Wehenkel L.** Extremely randomized trees. *Machine Learning*, 2006, Vol. 63, No. 1, Pp. 3–42.

12. **Utkin L.V., Ryabinin M.A.** A siamese Deep Forest. *Knowledge-Based Systems*, 2018, Vol. 139, Pp. 13–22.

13. **Freung Y., Shapire R.** A decision-theoretic generalization of on-line learning and an application to boosting. *J. Comput. Syst. Sci.*, 1997, Vol. 55, Pp. 119–139.

14. **Pang M., Ting K.M., Zhao P., Zhou Z.H.** Improving Deep Forest by confidence screening. *2018 IEEE International Conference on Data Mining* (*ICDM*), IEEE, 2018, Pp. 1194–1199.

15. **Utkin L., Konstantinov A., Meldo A., Ryabinin M., Chukanov V.** A Deep Forest improvement by using weighted schemes. *2019 24th Conference of Open Innovations Association* (*FRUCT*), IEEE, 2019, Pp. 451–456.

16. **Natekin A., Knoll A.** Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, 2013, Vol. 7, P. 21.

17. **Friedman J.H.** Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 2001, Pp. 1189–1232.

18. **Konstantinov A., Utkin L., Muliukha V.** Gradient boosting machine with partially randomized decision trees. *2021 28th Conference of Open Innovations Association* (*FRUCT*), IEEE, 2021, Pp. 167–173.

19. **Shen W., Guo Y., Wang Y., Zhao K., Wang B., Yuille A.L.** Deep regression forests for age estimation. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, Pp. 2304–2313.

20. **Konstantinov A.V., Utkin L.V.** A generalized stacking for implementing ensembles of gradient boosting machines. *Cyber-Physical Systems: Digital Technologies and Applications*, 2020, P. 3.

Received 05.07.2021.

СПИСОК ЛИТЕРАТУРЫ

1. **Ferreira A.J., Figueiredo M.A.T.** Boosting algorithms: A review of methods, theory, and applications // Ensemble Machine Learning. Springer, Boston, MA, 2012. Pp. 35–85.

2. **Rokach L.** Ensemble-based classifiers // Artificial Intelligence Review. 2010. Vol. 33. No. 1-2. Pp. 1–39.

3. **Woźniak M., Graña M., Corchado E.** A survey of multiple classifier systems as hybrid systems // Information Fusion. 2014. Vol. 16. Pp. 3–17.

4. **Zhou Z.-H.** Ensemble methods: Foundations and algorithms. CRC Press, Boca Raton, 2012.

5. **Breiman L.** Bagging predictors // Machine Learning. 1996. Vol. 24. No. 2. Pp. 123–140.

6. **Wolpert D.H.** Stacked generalization // Neural Networks. 1992. Vol. 5. No. 2. Pp. 241–259.

7. **Zhou Z.-H., Feng J.** Deep Forest: Towards an alternative to Deep neural networks // Proc. of the 26th Internat. Joint Conf. on Artificial Intelligence. AAAI Press, Melbourne, Australia, 2017. Pp. 3553–3559.

8. **Wang C., Lu N., Cheng Y., Jiang, B.** Deep Forest based multivariate classification for diagnostic health monitoring // 2018 Chinese Control and Decision Conf. IEEE, 2018. Pp. 6233–6238.

9. **Yang F., Xu Q., Li B., Ji Y.** Ship detection from thermal remote sensing imagery through region-based Deep Forest // IEEE Geoscience and Remote Sensing Letters. 2018. Vol. 15. No. 3. Pp. 449–453.

10. **Zheng W., Cao S., Jin X., Mo S., Gao H., Qu Y., Zhu Y.** Deep Forest with local experts based on elm for pedestrian detection // Pacific Rim Conference on Multimedia. Springer, Cham, 2018. Pp. 803–814.

11. **Geurts P., Ernst D., Wehenkel L.** Extremely randomized trees // Machine Learning. 2006. Vol. 63. No. 1. Pp. 3–42.

12. **Utkin L.V., Ryabinin M.A.** A siamese Deep Forest // Knowledge-Based Systems. 2018. Vol. 139. Pp. 13–22.

13. **Freung Y., Shapire R.** A decision-theoretic generalization of on-line learning and an application to boosting // J. Comput. Syst. Sci. 1997. Vol. 55. Pp. 119–139.

14. **Pang M., Ting K.M., Zhao P., Zhou Z.H.** Improving Deep Forest by confidence screening // 2018 IEEE Internat. Conf. on Data Mining. IEEE, 2018. Pp. 1194–1199.

15. **Utkin L., Konstantinov A., Meldo A., Ryabinin M., Chukanov V.** A Deep Forest improvement by using weighted schemes // 2019 24th Conf. of Open Innovations Association (FRUCT). IEEE, 2019. Pp. 451–456.

16. **Natekin A., Knoll A.** Gradient boosting machines, a tutorial // Frontiers in Neurorobotics. 2013. Vol. 7. P. 21.

17. **Friedman J.H.** Greedy function approximation: A gradient boosting machine // Annals of Statistics. 2001. Pp. 1189–1232.

18. **Konstantinov A., Utkin L., Muliukha V.** Gradient boosting machine with partially randomized decision trees // 2021 28th Conf. of Open Innovations Association (FRUCT). IEEE, 2021. Pp. 167–173.

19. **Shen W., Guo Y., Wang Y., Zhao K., Wang B., Yuille A.L.** Deep regression forests for age estimation // Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition. 2018. Pp. 2304–2313.

20. **Konstantinov A.V., Utkin L.V.** A generalized stacking for implementing ensembles of gradient boosting machines // Cyber-Physical Systems: Digital Technologies and Applications. 2020. P. 3.

Статья поступила в редакцию 05.07.2021.

THE AUTHOR / СВЕДЕНИЯ ОБ АВТОРЕ

Konstantinov Andrei V.
Константинов Андрей Владимирович
E-mail: andrue.konst@gmail.com