Ben-Gurion University of the Negev

Faculty of Engineering Sciences

The Department of Software and Information Systems Engineering

Machine Learning

PaloBoost, Accelerated Gradient Boosting (AGB)

& Regularized Greedy Forest (RGF)

Table of Contents

Algorithm 1

1. Algorithm Name

PaloBoost

2. Reference

Yubin Park , Joyce Ho (2019). "Tackling Overfitting in Boosting for Noisy Healthcare Data." IEEE Transactions on Knowledge and Data Engineering.

3. Motivation for the algorithm (or which problems it tries to solve?)

The Stochastic Gradient TreeBoost (SGTB) is often used in many winning public data science challenges and does an excellent job of dealing with the traits inherent in healthcare data, such as:

- Missing data
- Various data types

While SGTB can generally provide reasonable performance with the default hyperparameter settings, to achieve its best performance usually requires extensive hyperparameter tuning (the maximum depth of the tree, the number of trees, the learning rate, and the subsampling rate), which can lead to overfitting.

In addition, healthcare data tend to have noisy labels, inaccurate data, small sample sizes, and a large number of features. The development of a "hyperparameter robust" tree-based boosting algorithm that can achieve near-best performance under a broad range of hyperparameter configurations is necessary to mitigate these issues.

4. Short Description

PaloBoost is a Stochastic Gradient TreeBoost model that uses novel regularization techniques to guard against overfitting and is robust to hyperparameter settings.

PaloBoost uses the out-of-bag samples (the samples not included from the subsampling process) to perform gradient-aware pruning and estimate adaptive learning rates. Unlike other Stochastic Gradient TreeBoost models that use the out-of-bag samples to estimate test errors, PaloBoost treats the samples as a second batch of training samples to prune the trees and adjust the learning rates.

As a result, PaloBoost can dynamically adjust tree depths and learning rates to achieve faster learning at the start and slower learning as the algorithm converges.

5. Pseudo-Code

F - a function that minimizes the empirical risk associated with a loss function $\boldsymbol{L}(\cdot,\cdot)$ over $\boldsymbol{N}$ pairs of target $\boldsymbol{y}$, and input features $\boldsymbol{x}$;
$\boldsymbol{J}$ - Tree size;
$\boldsymbol{M}$ - Number of trees;
$\boldsymbol{q}$ - The subsampling rate;
$\boldsymbol{v}$ - The learning rate.

1. $F_0 = arg\ min_\beta \sum_{i=1}^{N} L(y_i, \beta)$
2. **for** $m = 1\ to\ M$
3. $\quad \{y_i, x_i\}_1^{N'} = Subsample(\{y_i, x_i\}_1^N, rate = q)$
4. $\quad for\ i = 1\ to\ N'$
5. $\quad\quad z_i = -\frac{\partial}{\partial F(x_i)} L(y_i, F(x_i))\Big|_{F=F_{m-1}}$
6. $\quad$ **end for**
7. $\quad \{R_{jm}\}_1^J = RegressionTree(\{z_i, x_i\}_i^{N'})$
8. $\quad$ **for** $j = 1\ to\ J$
9. $\quad\quad \gamma_{jm} = arg\ min_\gamma \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma)$
10. $\quad$ **end for**
11. $\quad \{R_{jm}, \gamma_{jm}\}_1^{J'} = GAP\left(\{R_{jm}, \gamma_{jm}\}_1^J, v_{max}\right)$
12. $\quad$ **for** $j = 1\ to\ J'$
13. $\quad\quad v_{jm} = ALR(\gamma_{jm}, v_{max})$
14. $\quad$ **end for**
15. $\quad F_m(x) = F_{m-1}(x) + \sum_{j-1}^{J} v_{jm}, \gamma_{jm} \mathbb{1}(x \in R_{jm})$
16. **end for**

6. **Algorithm Explanation**

- First, the intercept term F_0, is fit using data set. (Line 1)
- Next, we initialize number of trees (M) and repeat the following actions for each estimator (Line 2)
- Splitting the dataset randomly (Line 3)
- Calculate the gradients and train the tree on them (Line 5-7)
- Minimize the loss function within the disjoint region R_{jm} (Line 9)
- Then we prune trees using the algorithm Gradient-Aware Pruning. In reduced-error pruning, the errors on OOB samples are compared between children and parent nodes. If the child node does not decrease the error, the node is pruned, thereby reducing the complexity of the tree. (Line 11)
- Further, for each region, the learning rate is selected using the algorithm Adaptive Learning Rate. To enforce stability and maintain similarity with existing SGTB implementations, estimated learning rates are capped with the maximum learning rate hyper parameter v_{max}. As a result, the effective region-specific learning rate ranges between zero and the maximum specified learning rate. (Line 13)
- Update of the function *F* (Line 15)

7. Illustration

In this section, we will illustrate the operation of the PaloBoost algorithm on a simulated dataset of 10000 samples adopted from Friedman [4, 5] to introduce noisy features (ie, functions with zero importance). Table 1 lists the characteristics associated with the dataset. The dataset is preprocessed: 1) no imputation of missing values, 2) minimal outlier detection, 3) adding polynomial (or interacting) features for better predictive performance [1].

Task	Missing Data	Target Stats	Samples	Features
Regression	None	$\sigma(y) = 6.953$	10000	55

Tab. 1. Friedman Data Set

Figure 1 shows the performance of the PaloBoost algorithm compared to some other algorithms based on the coefficient of determination (R^2) over boosting iterations for the three different learning rates and three tree depths.

The curves are drawn by averaging the curves from the 10 random train-test runs. The higher the R^2 values, the more accurate the predictions are.

- The results show that PaloBoost demonstrates the best and most stable predictive performance for all hyperparameter configurations.
- If at a learning rate of 0.1 there is not such a strong difference, then at higher values of the learning rate the algorithm demonstrates much better results.
- While other SGTB implementations show significant degradation in predictive performance with iteration, PaloBoost shows a gradual decrease in predictive performance even at high learning rates.
- PaloBoost shows the best performance on average up to 50 iterations.

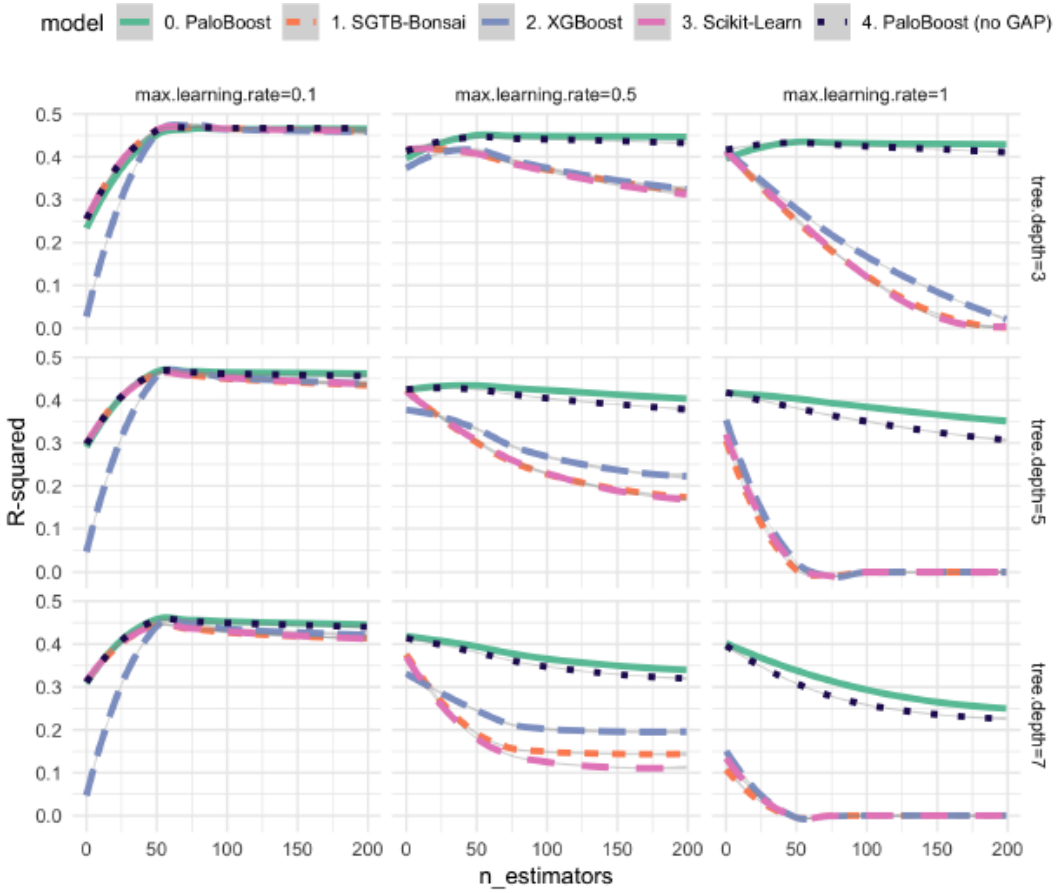


Fig. 1. Predictive performance over boosting iterations:

- Scikit-Learn, the de-facto machine learning library in Python that has been widely adopted.
- XGBoost, the battle-tested library that has won many data science challenges.
- Bonsai-SGTB, an SGTB implementation using the Bonsai framework.
- PaloBoost wiht/whithout GAP, Gradient-aware pruning (Removing the regions with higher variances of node estimates).

In the figure, also included the performance of PaloBoost without GAP (Gradient-aware pruning).

- We can observe that the effect of GAP is more noticeable in baseline trees with higher depths, such as depths 5 and 7. GAP improves overall performance at various hyperparameter settings.

Figure 2 shows the performance differences between the best and last boosting iterations. In each cell, boxplots show the performance differences from different hyperparameter configurations. The smaller the performance differences, the less performance degradation from their best performances.

- With different configurations of PaloBoost hyperparameters, there are minimal performance differences.
- PaloBoost can maintain its best performance even with a large number of iterations.
- Combining the conclusions of figures 1 and 2, it can be noted that Paloboost consistently achieves high performance with different variations in hyperparameters.

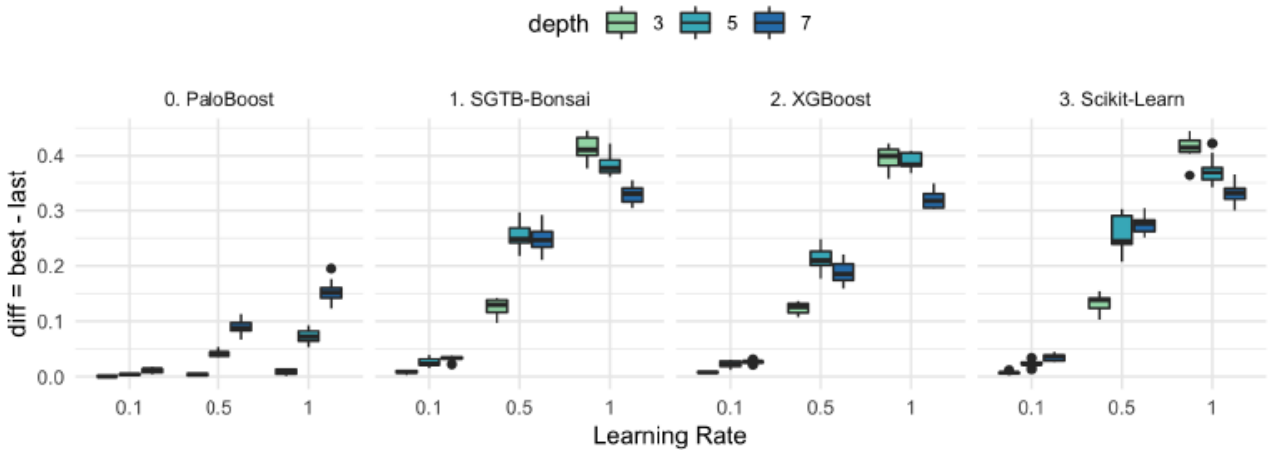


Fig. 2. Performance differences between the best and last boosting iterations.

The gradual decline of predictive performance in PaloBoost can be better understood by analyzing the pruning rate and average learning rate at each stage. Figure 3 and 4 display the average learning rates and the pruning rates over the same iterations, respectively. On top of the original data in gray lines, overlaid LOESS (locally estimated scatterplot smoothing) curves to visualize.

- It can be seen from the figure 3 that as the number of stages increases, the learning rate greatly decreases and aadaptively adjusting. At all iterations, the learning rate is below the specified maximum rate.
- Figure 4 shows that with an increase in the number of iterations, the tree pruning speed increases.
- Thus, these two mechanisms guard against overfitting even when adding more boosting iterations.

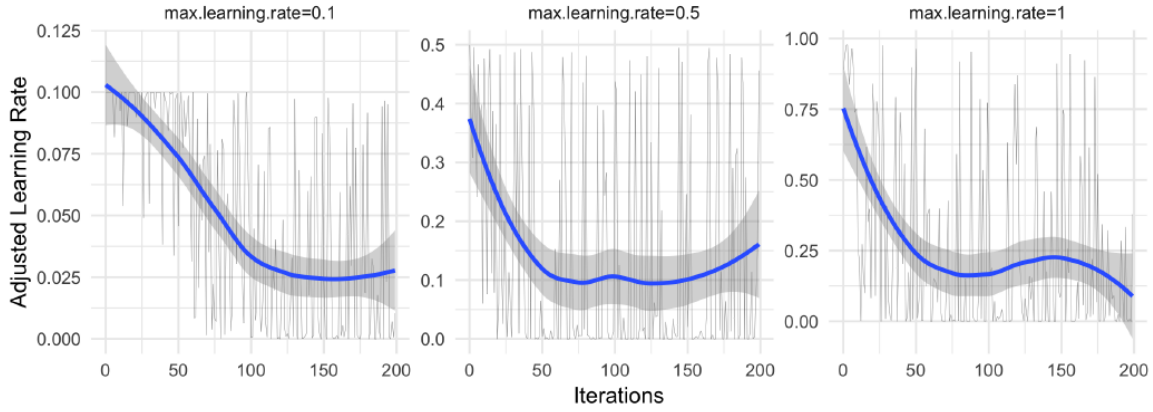


Fig. 3. The average learning rates over boosting iterations.

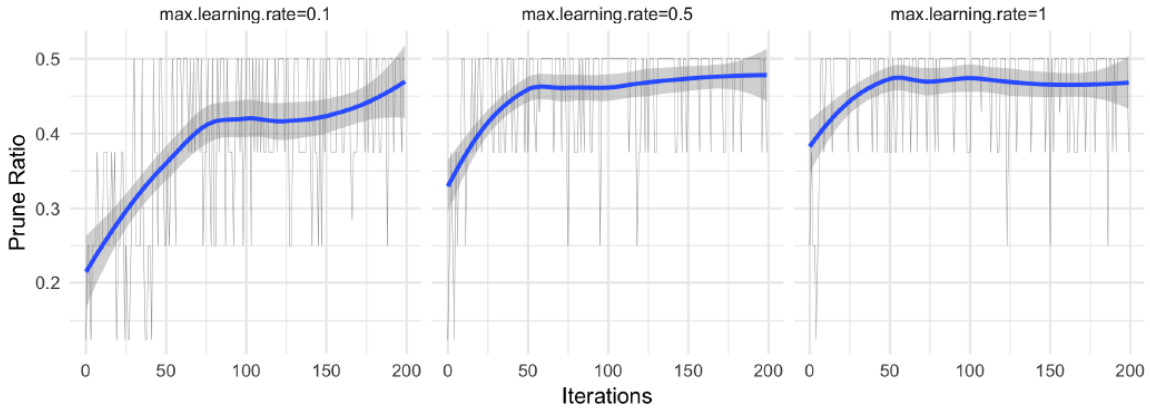


Fig. 4. The average pruning rates over boosting iterations.

8. Strengths

- Uses novel regularization techniques to guard against overfitting and is robust to hyper parameter settings;
- Unlike other Stochastic Gradient Tree Boost models that use the out-of-bag samples to estimate test errors, PaloBoost treats the samples as

a second batch of training samples to prune the trees and adjust the learning rates. As a result, PaloBoost can dynamically adjust tree depths and learning rates to achieve faster learning at the start and slower learning as the algorithm converges;

- PaloBoost is robust to overfitting and is less sensitive to the hyper parameters;
- PaloBoost produces robust predictive performance over various real-world healthcare data. It is because healthcare data often exhibit: 1) noisy labels, 2) complex data types, 3) missing data, 4) small sample sizes, and 5) a large number of features.

9. Drawbacks

- Presence of trees with negligible ($\nu \sim 0$) learning rates in PaloBoost. Given that these trees have minimal impact on the overall performance, they can be potentially removed. This should yield a more compact-sized tree boosting model while offering similar predictive performance;
- If a dataset has accurate labels and has easy-to-predict target variables, PaloBoost's adaptive learning and pruning may slow down its learning speed. PaloBoost sometimes requires more iterations to achieve their best performance.

10. Citations

There are no citations for the PaloBoost algorithm article. We found one citation to the pre-print version of the article [9], but this citation is completely irrelevant to the work and performance of the algorithm itself.

1. *Abhibhav Sharma, Buddha Singh. (2020). AE-LGBM: Sequence-based novel approach to detect interacting protein pairs* via *ensemble of autoencoder and LightGBM. Computers in Biology and Medicine.*

   To evaluate the predictive performance of our ensemble model, we performed 10-fold cross-validation, considering that complex gradient boosting models are prone to overfit on a small training sample.

Algorithm 2

1. Algorithm Name

Accelerated Gradient Boosting (AGB)

2. Reference

Biau, G., Cadre, B., & Rouvière, L. (2019). Accelerated Gradient Boosting. Machine Learning, 108(6), 971-992.

3. Motivation for the algorithm (or which problems it tries to solve?)

Gradient tree boosting are generally regarded as one of the best off-the-shell prediction algorithms we have today.

In a different direction, the pressing demand of the machine learning community to build accurate prediction mechanisms from massive amounts of high dimensional data has greatly promoted the theory and practice of accelerated first-order schemes. In this respect, one of

The most effective approaches among first-order optimization techniques is the so-called Nesterov's accelerated gradient descent.

It can be suggested that by combining gradient boosting and Nesterov's accelerated descent, an algorithm can be obtained with the advantages of both approaches:

- Gradient boosting: predictive performance;
- Nesterov's accelerated descent: significant acceleration in the rate of convergence (especially when we are faced with large-scale data).

4. Short Description

"AGB (Accelerated gradient boosting), a new tree boosting algorithm that incorporates Nesterov's mechanism into Friedman's original procedure (Gradient Boosting).

Gradient tree boosting is a prediction algorithm that sequentially produces a model in the form of linear combinations of decision trees, by solving an infinite-dimensional optimization problem."

The idea of Nesterov's momentum optimization, or Nesterov accelerated gradient, is to measure the gradient of the loss function not at the local position, but slightly ahead in the direction of the moment. In other words, "Nesterov's descent performs a simple step of gradient to go from y_t to x_{t+1}, and then it slides it a little bit further than x_{t+1} in the direction given by the previous point x_t".

5. Pseudo-Code

AGB - Building the ensemble

Input: T - number of iterations ($T \geq 1$),
k - number of terminal nodes in the trees ($k \geq 1$),
ν - shrinkage parameter ($\nu \in (0, 1)$),
S - a labeled training set, $S = (< x_1, y_1 >, ..., < x_m, y_m >)$.

1. **Initialize** $F_0 = G_0 = arg\ min_Z \sum_{i=1}^{n} \psi(z, Y_i), \lambda_0 = 0, \gamma_0 = 1.$
2. **for** $t = 0$ to $(T - 1)$ **do**
3. For $i = 1, ..., m$, **compute** the negative gradient instances
$$Z_{i,t+1} = -\nabla C_m(G_t), (X_i).$$
4. **Fit** a regression tree to the pairs $(X_i, Z_{i,\,t+1})$, giving terminal nodes $R_{j,\,t+1}$, $1 \leq j \leq k$.
5. For $j = 1, ..., k$, **compute**
$$\omega_{j,t+1} \in arg\ min_{\omega>0} \sum_{X_i \in R_{j,t+1}} \psi(Gt(Xi) + \omega, Yi).$$
6. **Update**
   (a) $F_{t+1} = G_t + \nu \sum_{j=1}^{k} \omega_{j,t+1} \mathbb{1}_{R_{j,t+1}}.$
   (b) $G_{t+1} = (1 - \gamma_t)F_{t+1} + \gamma_t F_t.$
   (c) $\lambda_t = \frac{\sqrt{1+4\lambda_{t-1}^2}}{2}, \lambda_{t+1} = \frac{1+\sqrt{1+4\lambda_t^2}}{2}.$
   (d) $\gamma_t = \frac{1-\lambda_t}{\lambda_{t+1}}.$
7. **end for**

AGB - Regressor an instance

8. **Return output** $\sum_t^T a_t F_t.$

6. Algorithm Explanation

The algorithm has two inner functional components, $(Ft)_t$ and $(Gt)_t$, which correspond respectively to the vectorial sequences $(x_t)_t$ and $(y_t)_t$ of Nesterov's acceleration scheme:

$$x_{t+1} = y_t - \omega \nabla f(y_t)$$

$$y_{t+1} = (1 - \gamma_t) x_{t+1} + \gamma_t x_t,$$

where ω is the step size, $\lambda_0 = 0$, $\lambda_t = \frac{\sqrt{1+4\lambda_{t-1}^2}}{2}$, and $\gamma_t = \frac{1-\lambda_t}{\lambda_{t+1}}$.

The sequence $(G_t)_t$ is internal to the procedure while the linear combination output by the algorithm after T iterations is F_T .

Line 1 initializes to the optimal constant model. As in Friedman's original approach, the algorithm selects at each iteration, by least-squares fitting, a particular tree that is in most agreement with the descent direction (the "gradient"), and then performs an update of Gt. The essential difference is the presence of the companion function sequence $(Gt)_t$, which slides the iterates $(Ft)_t$ according to the recursive parameters λt and γ_t (lines 6 (b)-(d)).

The next logical step is to perform a linear search to find the step size and update the model accordingly, as shown in lines 5 and 6 (a).

7. Illustration

In this section, we will illustrate how AGB works on both simulated and real datasets.

For each of the simulated models, two designs were considered for $X = (X_1, \dots, X_d)$: Uniform over$(-1, 1)^d$ ("Uncorrelated design") and Gaussian with mean 0 and $d \times d$ d covariance matrix $\mathbf{\Sigma}$ such that $\sum_{ij} = 2^{-|i-j|}$("Correlated design"). The following five models cover a wide spectrum of regression problems. Models 1–3 come from [6].

Models 2 and 3 are additive, while Model 1include some interactions. Model 3 can be seen as a sparse high-dimensional problem.

<u>Model 1</u>: $n = 1000, d = 100,\ Y = X_1 X_2 + X_3^2 - X_4 X_7 + X_8 X_{10} - X_6^2 - Z_{0,0.5}$.

Model 2: $n = 800, d = 100,\ Y = -\sin(2X_1) + X_2^2 + X_3 - \exp(-X_4) + Z_{0.5}$.

Model 3: $n = 1000, d = 500,\ Y = X_1 + 3X_3^2 - 2\exp(-X_5) + X_6$.

Z_{μ,σ^2} - Gaussian random variable with mean μ and variance σ^2.

Also the real-world datasets Community & Crime and Wine (Tab. 2) from the UCI Machine Learning repository [8] considered

Data Set	Task	Samples	Features
Crime	Regression	1993	102
Wine	Regression	1559	11

Tab. 2. Main characteristics of the real-life data sets.

In the boosting algorithms, the validation set is used to select the number of components of the model, i.e., the number of iterations performed by the algorithm. Thus, denoting by F_T the boosting predictor after T iterations fitted on D_{train}, we select the T^* is chosen that minimizes

$$\frac{1}{\#Dval} \sum_{i \in Dval} \psi(F_T(X_i), Y_i).$$

For both standard gradient tree boosting and AGB, regression trees with two terminal nodes were fitted. Five fixed values for the shrinkage parameter ν (1e - 05, 0.001, 0.01, 0.1, and 0.5) were considered, and an arbitrary (large) limit T = 10000 iterations was fixed for the standard boosting and T = 2500 for AGB. All results are averaged over 100 replications for simulated examples, and over 20 independent permutations of the sample for the real-life data.

Figure 5 shows the training and validation errors for Friedman's boosting and AGB, as a function of the number of iterations.

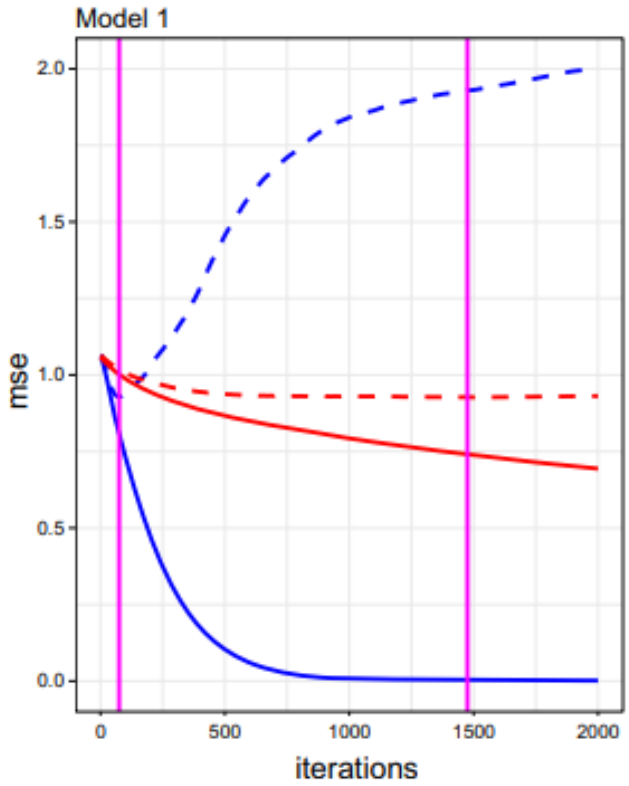


Fig. 5. Training (solid lines) and validation (dashed lines) errors for standard gradient boosting (red) and AGB (blue) for Model 1. Shrinkage parameter ν is fixed to 0.01 (Color figure online).

"As it is generally the case for gradient boosting, the validation error decreases until predictive performance is at its best and then starts increasing again. The vertical magenta line shows the optimal number of iterations T, selected by minimizing. We see that the validation rates at the optimal T^* are comparable for AGB and the original algorithm. However, AGB outperforms gradient boosting in terms of number of components of the output model, which is much smaller for AGB. This is a direct consequence of Nesterov's acceleration scheme."

Figures 6, 7, and 8 show the relationship between predictive performance, the number of iterations, and the shrinkage parameter. On the left side of each figure, box plots of test errors are shown for the selected F_{T^*}, i.e.,

$$\frac{1}{\#Dval} \sum_{i \in Dval} \psi(F_{T^*}, (X_i), Y_i),$$

as a function of the shrinkage parameter ν. The right sides depict the boxplots of the optimal number of components T^*.

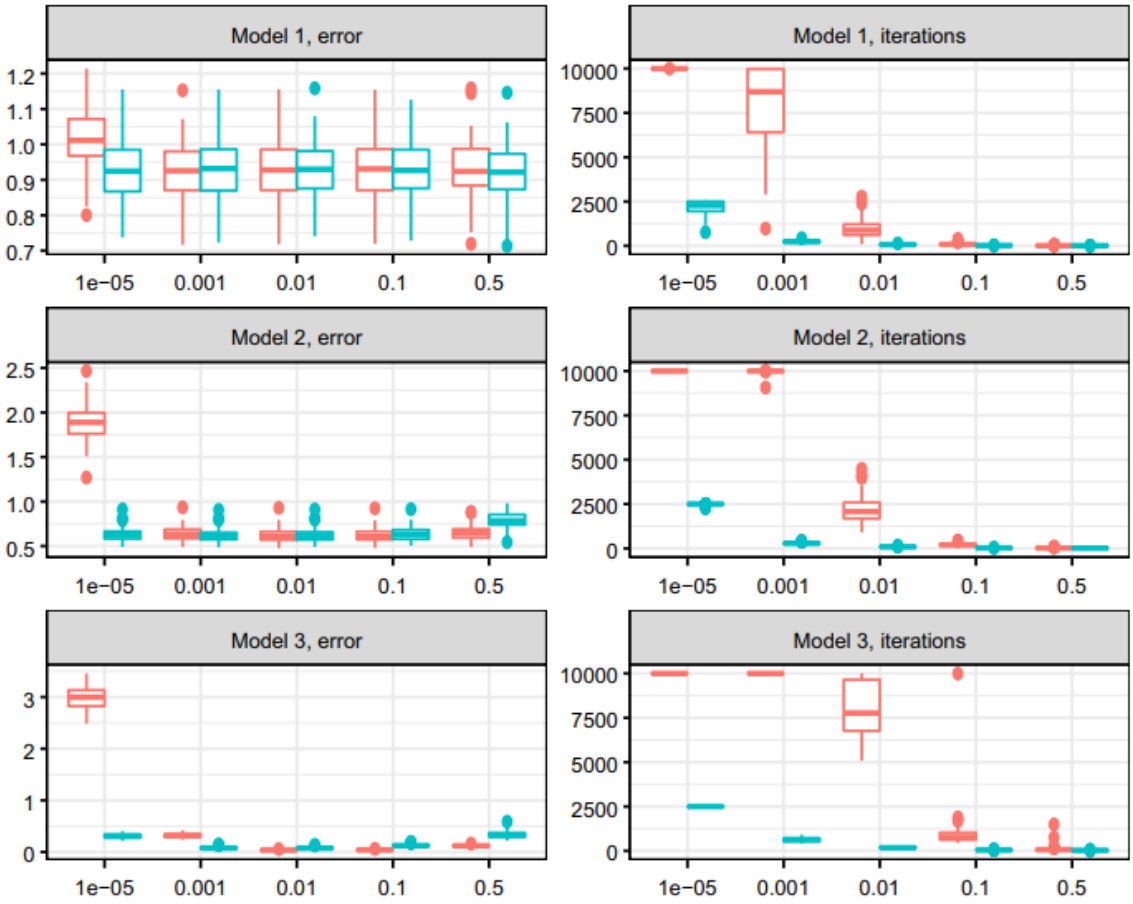


Fig. 6. Boxplots of the test error (left) and selected numbers of iterations (right), as a function of the shrinkage parameter ν for standard gradient boosting (red, left) and AGB (blue, right). Results are presented for simulated models with uncorrelated (Color figure online).

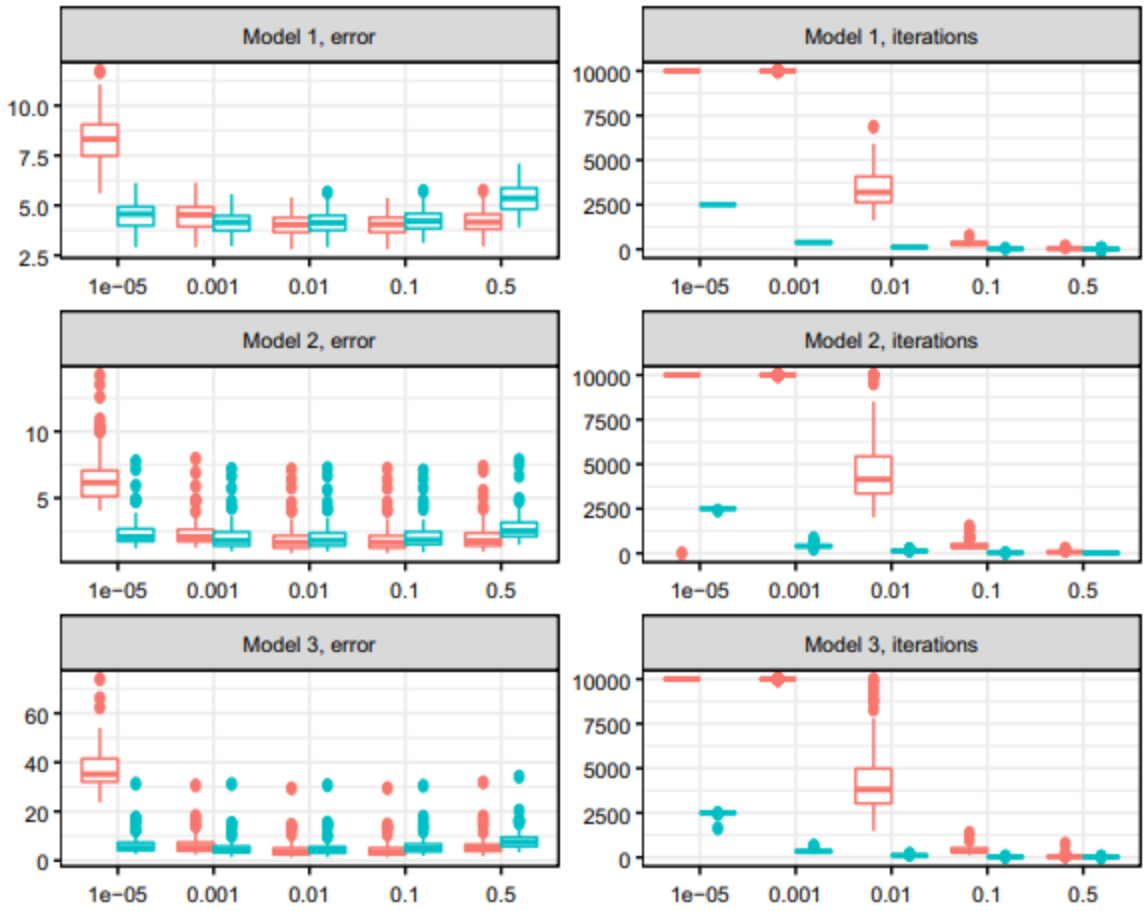


Fig. 7. Boxplots of the test error (left) and number of selected iterations (right) as a function of the shrinkage parameter ν, for standard gradient boosting (red, left) and AGB (blue, right). Results are presented for simulated models with correlated design (Color figure online).

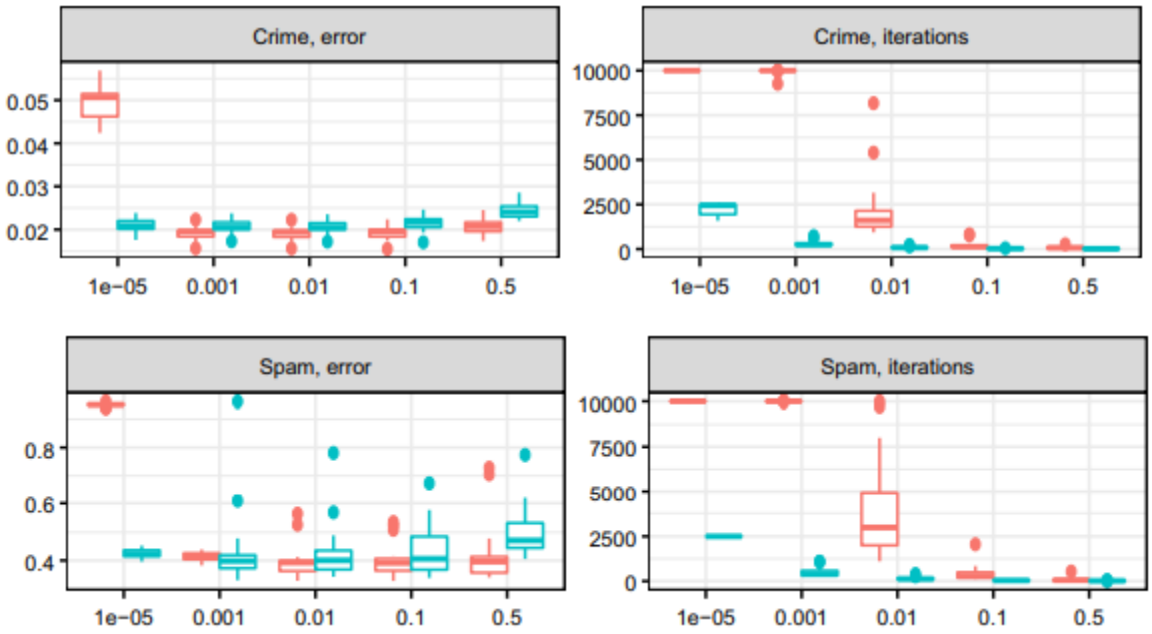


Fig. 8. Boxplots of the test error (left) and number of selected iterations (right) as a function of the shrinkage parameter ν, for standard gradient boosting (red, left) and AGB (blue, right). Results are presented for real-life data sets (Color figure online).

- Smaller values of the shrinkage parameter v favor better test error. Can be seen in all date sets, both real and simulated, the best test errors are achieved when $v < 0.1$.
- With a shrinkage parameter $v > 0.01$, a significantly larger number of iterations is required to achieve the optimal T.
- In the case of the standard boosting at $v > 0.01$, 10000 iterations may be required.
- In the case of the AGB up to 2500 at the lowest value of the shrinkage parameter $v = 1e - 05$.
- The accelerated algorithm allows to avoid this problem since, for each value of v, the optimal model is achieved after a number of iterations considerably smaller than with standard boosting.

8. Strengths

1. «AGB is much less sensitive to the shrinkage parameter and outputs predictors that are considerably more sparse in the number of trees, while retaining the exceptional performance of gradient boosting.»

2. In the case of the accelerated algorithm for each value of the shrinkage parameter, the optimal model is achieved after a number of iterations considerably smaller than with standard boosting.
3. AGB uses far less components in the output model.

9. Drawbacks

"On the other hand, Nesterov's accelerated descent is provably faster than gradient descent when the gradient used is accurate (Bubeck 2015, Chapter 3). However, when the gradient is not accurate (e.g., in a stochastic setting), then Nesterov's descent is prone to accumulating error and diverging. This type of situation is analyzed in Devolder et al. (2014), who prove that the superiority of fast gradient methods over the classical ones is no longer absolute when an inexact oracle is used. Therefore, the benefits of Nesterov's technique may be lost, or reduced, in some inexact gradient settings. This is of course the case in our boosting problem, since the gradient direction is highly inexact due to the least squares approximation."

10. Citations

1. *J Hartmann. Classification Using Decision Tree Ensembles. Available at SSRN 3484009, 2019*

   Several derivatives of AdaBoost include gradient boosting, XGBoost, and accelerated gradient boosting. The main advantage of gradient boosting is its speed relative to traditional algorithms.

2. *F Sigrist. Gaussian Process Boosting. arXiv preprint arXiv:2004.02653, 2020*

   As in finite dimensional optimization, functional gradientdescent can also be acceleratedusing momentum. For instance, Biau et al. propose to use Nesterov acceleration for gradient boosting.

3. *X Ju, Y Sun, S Vaswani, M Schmidt. Accelerating boosting via accelerated greedy coordinate descent*

Another avenue is to exploit the close connection between boosting and greedy coordinate descent (GCD) methods. In particular, it has previously been observed that Adaboost is equivalent to coordinate minimization under the Gauss-Southwell rule, and LogitBoost is inspired by agreedy selection rule of a Taylor approximation of the loss function. Similar observations have also beenmade about gradient boosted machines . This has inspired the development of several "accelerated" boosting schemes, such as MultiBoost, GradBoost, LPBoost, and Accelerated GradientBoosting (AGB).

Algorithm 3

1. Algorithm Name

Regularized Greedy Forest (RGF)

2. Reference

Rie Johnson, Tong Zhang (2014). "Learning Nonlinear Functions Using Regularized Greedy Forest". IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE

3. Motivation for the algorithm (or which problems it tries to solve?)

“Boosted decision trees” is regarded as the most effective off-the-shelf nonlinear learning method for a wide range of application problems." One boosted tree approach include bagging and random forests. In this context, we may view boosted decision tree algorithms as methods to learn decision forests by applying a greedy algorithm (boosting) on top of a decision tree base learner. This indirect approach is sometimes referred to as a wrapper approach (in this case, wrapping boosting procedure over decision tree base learner); the boosting wrapper simply treats the decision tree base learner as a black box and it does not take advantage of the tree structure itself.

The advantage of such a wrapper approach is that the underlying base learner can be changed to other procedures with the same wrapper; the disadvantage is that for any specific base learner which may have additional structure to explore, a generic wrapper might not be the optimal aggregator. Due to the practical importance of boosted decision trees in applications, it is natural to ask whether one can design a more direct procedure that specifically learns decision forests without using a black-box decision tree learner under the wrapper."

4. Short Description

Regularized Greedy Forest (RGF) is a direct decision forest learning algorithm. Unlike the traditional boosted decision tree approach, RGF works directly with the underlying forest structure. RGF integrates two ideas: one is to include tree-structured regularization into the learning formulation; and the other is to employ the fully-corrective regularized greedy algorithm.

5. Pseudo-Code

1. $F \leftarrow \{\}$.
   repeat
2. $F \leftarrow$ the optimum forest that minimizes $Q(F)$ among all the forests that can be
   obtained by applying one step of structure-changing operation to the current
   forest *F*.
3. **if** *some criterion is met* **then** optimize the leaf weights in F to minimize loss $Q(F)$.
   until *some exit criterion is met*;
   Optimize the leaf weights in F to minimize loss $Q(F)$.
   return $h_F(x)$

F represent a forest (fig.9), and each node ν of F is associated with (b_ν, α_ν). Here b_ν is the basis function that this node represents; α_ν is the *weight* or coefficient assigned to this node. The additive model of this forest F is:

$$h_F(x) = \sum_{\nu \in F} \alpha_\nu b_\nu(x),$$

where $\alpha_\nu = 0$ for any internal node ν.
The regularized loss is a function of decision forest:

$$Q(F) = L(h_F(X), Y) + R(h_F),$$

where $L(h_F(X), Y) -$ loss function, and $R(h_F) -$ regularization:

$$R(h_T) = \lambda \sum_{\nu \in T} \frac{\alpha_\nu^2}{2} = \lambda \sum_{\nu \in L_T} \frac{\alpha_\nu^2}{2},$$

where λ is a constant for controlling the strength of regularization.

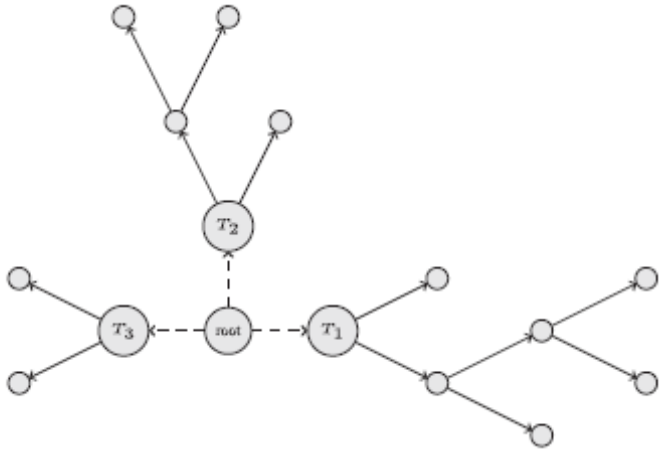


Fig. 9. Decision forest: T_1, …, T_K - decision trees.

6. Algorithm Explanation

The training objective of RGF is to build a forest that minimizes $Q(F)$ defined. Since the exact optimum solution is difficult to find, we greedily select the basis functions and optimize the weights. RGF essentially has two main components as follows.

• Fix the weights, and change the structure of the forest (which changes basis functions) so that the loss $Q(F)$ is reduced the most (Line 2).

• Fix the structure of the forest, and change the weights so that loss $Q(F)$ is minimized (Line 3).

7. Illustration

In this section, we will illustrate how the algorithm works by comparing it with GBDT and several ensemble tree methods.

Evaluation on synthesized datasets

First was studied the performance of the methods in relation to the complexity of target functions using synthesized datasets. To synthesize datasets, first was defined the target function by randomly generating 100 leaf regression trees and then randomly generated data points and applied the target function to them to assign the output/target values.

The results shown in Table 3 are in the root mean square error (RMSE) averaged over three runs. In each run, randomly chosen 2K data points were used for training and the number of test data points was 20K. The parameters were chosen by 2-fold cross validation on the training data. Since the task is regression, the loss function for RGF and GBDT were set to square loss. RGF used here is the most basic version, which does L2 regularization with one parameter λ for both forest growing and weight correction. λ was chosen from $\{1,0.1,0.01\}$. The tree size (in terms of the

number of leaf nodes) and the shrinkage parameter were chosen from {5,10,15,20,25} and {0.5,0.1,0.05,0.01,0.005,0.001}, respectively. Table 3 shows that RMSE achieves smaller error than GBDT on all types of datasets.

	RGF-L_2	GBDT	RGF min-penalty			
			w/sib. constraint		w/o sib. constraint	
5-leaf data	**0.2200**	0.2597	0.1885	(0.0315)	0.1890	(0.0310)
10-leaf data	**0.3480**	0.3968	0.3270	(0.0210)	0.3266	(0.0214)
20-leaf data	**0.4578**	0.4942	0.4545	(0.0033)	0.4538	(0.0040)

Table 3. Regression results on synthesized datasets. RMSE. Average of 3 runs, each of which used randomly drawn 2K training data points.RGF-L2outperforms GBDT. RGF min-penalty (with or without the sibling constraint) further improves accuracy; the numbers in parentheses are accuracy improvements over RGF-L2.

RGF with min penalty regularizer with the sibling constraints further improves RMSE over RGF-L2 by 0.0315, 0.0210, and 0.0033 on the 5-leaf, 10-leaf, and 20-leaf synthesized datasets, respectively. RGF with min penalty regularizer without the sibling constraints also achieved the similar performances. Based on the amount of improvements, min penalty regularizer appears to be more effective on simpler targets. Figure 10 plots RMSE in relation to the model size in terms of the number of basis functions or leaf nodes. RGF produces better RMSE at all the model sizes.

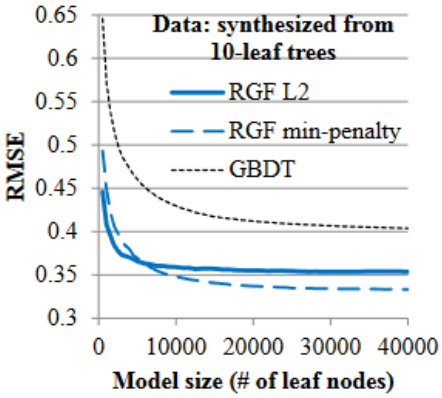


Fig. 10: Regression results in relation to model size. One particular run on the data synthesized from 10-leaf trees.

Evaluation on real dataset

All datasets except Houses [7] were taken from the UCI repository [8]. All the results are the average of 3 runs, each of which used randomly drawn 2K training data points. (table 4)

Name	Dim	Regression tasks
CT slices	384	Target: relative location of CT slices
California Houses	6	Target: log(median house price)
YearPredictionMSD	90	Target: year when the song was released

Table 4: Real-world Datasets. The numbers in parentheses indicate the dimensionality after converting categorical attributes to indicator vectors.

All the parameters were chosen by 2-fold cross validation on the training data. The RGF tested here is RGF-L2 with the extension in which the processes of forest growing and weight correction can have regularization parameters of different values, which called λ_g ('g' for 'growing') and λ, respectively. The value of λ was chosen from {10,1,0.1,0.01} with square loss, and from {10,1, 0.1, 0.01,1e−10,1e−20,1e−30} with logistic loss and exponential loss. λ_g was chosen from {λ, λ/100}. The tree size for GBDT was chosen from {5,10,15,20,25}, and the shrinkage parameter was from {0.5,0.1,0.05,0.01,0.005,0.001}.

In addition to GBDT, were also tested two other tree ensemble methods: random forests and Bayesian additive regression trees (BART). We performed random forest training with the number of randomly drawn features in $\{\frac{d}{4},\frac{d}{3},\frac{d}{2},\frac{3d}{5},\frac{7d}{10},\frac{4d}{5},\frac{9d}{10},\sqrt{d}\}$, where d is the feature dimensionality; the number of trees set to 1000; and other parameters set to default values. BART is a Bayesian approach to tree ensemble learning.

Table 5 shows the regression results in RMSE. RGF achieves lower error than all others.

	RGF-L_2	GBDT	RandomF.	BART
CT slices	**7.2037**	7.6861	*7.5029*	8.6006
California Houses	**0.3417**	0.3454	*0.3453*	0.3536
YearPredictionMSD	**9.5523**	9.6846	9.9779	*9.6126*

Table 5: Regression results. RMSE. Average of 3 runs, each of which used randomly-drawn 2K training data points. The best and second best results are in bold and italic, respectively.

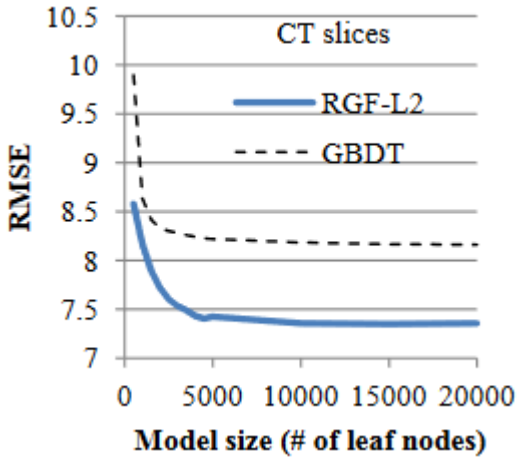


Fig. 11: RMSE/Accuracy in relation to model size. One particular run on the representative datasets.

Fig.11 shows the RMSE of RGF and GBDT in relation to the model sizes on the representative datasets. Similar to Fig. 10 (on the synthesized data), RGF is more accurate than GBDT at all model sizes; in other words, to achieve similar accuracy, RGF only requires a smaller model than GBDT.

8. Strengths

- Method achieves higher accuracy and smaller models than gradient boosting models;

- RGF does not require the tree size parameter needed in GBDT. With RGF, the size of each tree is automatically determined as a result of minimizing the regularized loss;
- RGF produces better RMSE at all the model sizes; in other words, to achieve similar RMSE, RGF requires a smaller model than GBDT;
- RGF is more accurate than GBDT (and AdaBoost) at all model sizes; in other words, to achieve similar accuracy, RGF only requires a smaller model than GBDT;
- RGF can achieve performance superior to GBDT even in the most competitive situation.

9. Drawbacks

- Compared with GBDT, computation of RGF involves additional complexity mainly for fully-corrective weight updates; however, running time of RGF is linear in the number of training data points.

10. Citations

1. *Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. InProceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pages 785–794. ACM, 2016*

   XGBoost incorporates a regularized model to prevent overfitting. This this resembles previous work on regularized greedy forest, but simplifies the objective and algorithm for parallelization.

2. *Adam-Bourdarios, C., Cowan, G., Germain, C., Guyon, I., Kégl, B., and Rousseau, D. (2014). The higgs boson machine learning challenge. InNIPS 2014 Workshop on High-energy Physics andMachine Learning, volume 42, page 37.*

   The solution of the second winner, Tim Salimans, pushes decision trees further away from greediness. It is based on the Regularized Greedy Forest (RGF) algorithm, a variation on gradient boosting that decouples structure search and optimization. The final solution

combines multiple RGF through stacking, with a linear regression as the learning model.

3. *Botezatu, M. M., Giurgiu, I., Bogojeska, J.,Andwies-Mann, D. Predicting disk replacement towards reliable data centers. InProceedings of the 22nd ACM SIGKDD InternationalConference on Knowledge Discovery and Data Mining(2016),ACM, pp. 39–48.*

   Using the regularized greedy forests (RGF) approach that is a powerful, non-linear classification method delivers better quality predictions compared to other tree ensemble based methods such as gradient boosted decision trees (GBDT) or random forests and also outperforms other classification methods such as SVM, or logistic regression.

4. *Huan Zhang, Si Si, and Cho-Jui Hsieh. Gpu-acceleration for large-scale tree boosting.arXiv preprint arXiv:1706.08359, 2017.*

   This algorithm has shown superb performance in regression, classification, and ranking tasks.

5. *Jean-Michel Begon, Arnaud Joly, and Pierre Geurts. Globally induced forest: A prepruning compression scheme. In Doina Precup and Yee Whye Teh (eds.),Proceedings of the 34th InternationalConference on Machine Learning, volume 70 ofProceedings of Machine Learning Research, pp.420–428, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR.*

   As an important difference, however, splits in RGF are globally optimized based on the current forest predictions, while splits in GIF are optimized locally and only the nodes and their weights are chosen globally. This local optimization, to-get her with the learning rate and candidate subsampling, acts as the main regularizer for GIF, while RGF uses explicit regularization through the objective function.

6. *T.R. Lekhaa, "Efficient Crop Yield and Pesticide Prediction for Improving Agricultural Economy using Data Mining Techniques", International Journal of Modern Trends in Engineering and Science (IJMTES), 2016, Volume 03, Issue 10*

   Regularized Greedy Forest is an additive decision tree algorithm in which a series of boosted decision trees are created and additively form a forest as a single predictive model. A globally optimized decision tree is created in RGF whereas locally optimized decision tree is created in GBDT. RGF takes advantage of tree structure as it works on fully corrective regularized steps where as GBDT does not take advantage of tree structure as it works on partially regularized steps. RGF works faster and more accurate than GBDT for regression problem.

7. *J. Zhang, K. Zhou, P. Huang, X. He, M. Xie, B. Cheng,Y. Ji, and Y. Wang. Minority disk failure prediction based on transfer learning in large data centers of hetero-geneous disk systems. IEEE Transactions on Parallel and Distributed Systems, 31(9):2155–2169, 2020.*

   To avoid overfitting, the GBRT algorithm trains many tree stumps as week learners, rather than full, high variance trees. Moreover, the Regularized Greedy Forests approach is a powerful, non-linear classification method. It is a variation of GBRT in which the structure search and optimization are decoupled and it utilizes the concept of structured sparsity to perform greedy search directly over the forest nodes based on the forest structure.

8. *Samat, E. Li, W. Wang, S. Liu, C. Lin, and J. Abuduwaili, "Meta-xgboost for hyperspectral image classification using extended mser-guided morphological profiles,"Remote Sens., vol. 12, no. 12, p. 1973,2020.*

   Although regularized greedy forest (RGF) decision tree-based ensemble learning methods can provide state-of-the-art results based

on many standard classification and ranking benchmarks, gradient-boosting decision trees (GBDT) have recently gained considerable interest due to their superb performance and flexibility in incorporating different loss functions.

9. *M. Nalenz, "Horseshoe rulefit: Learning rule ensembles via bayesianregularization," 2016.*

   Reguralized Greedy Forest use regularization to limit the complexity of Ensemble of Tree models. In this result regularization increases predictive performance while limiting the model complexity. This questions thestated inverse relationship between accuracy and model complexity.

Experimental Results

In this part we will compare 4 algorithms: The three algorithms discussed in the previous sections - PaloBoost, Accelerated Gradient Boosting (AGB), Regularized Greedy Forest (RGF); and a well-known state-of-the-art gradient boosting on decision trees library – CatBoost. These algorithms solve the regression problem. The comparison will be carried out on 100 datasets.

As measures of the quality of algorithms, we will measure the following parameters:

- MSE
- Mean absolute error
- R-squared score
- Explained variance
- Training time
- Inference time (Predict on 1000 rows)

These parameters are measured in outer 10 folds cross validation. We use an inner 3-fold cross-validation to tune the hyperparameters.

The hyperparameter tuning is done using randomized_search(CatBoost) for CatBoost and RandomizedSearchCV (Scikit-learn) for all other algorithms; and includes 50 iterations.

Below are the hyperparameter for the algorithms:

PaloBoost:

```
params_PaloBst={
'distribution':["gaussian"],
'learning_rate':np.arange(0.01,1,0.05),
'n_estimators':np.arange(100,500,50),
'subsample':np.arange(0.1,1,0.1)}
```

Accelerated Gradient Boosting (AGB):

```
params_AcceleratedBoost={'
```

```
descent':['proximal','gradient'],
'loss':['ls','lad'],
'learning_rate':np.arange(0.01,1,0.05),'
n_estimators':np.arange(100,500,50)}
```

Regularized Greedy Forest (RGF):

```
params_RGFR={'l2':np.arange(0.01,1,0.05),
'max_leaf':range(1000,10000,200),
'learning_rate':np.arange(0.01,1,0.05)}
```

CatBoost:

```
params_CatBoost = {
'learning_rate':np.arange(0.05, 0.31, 0.05),
'depth': [4, 6, 10],
'l2_leaf_reg': [1, 3, 5, 7, 9]}
```

Below is a part of the table with the obtained measurements:

Dataset name	Algorithm	MSE	Mean absolute error	Median absolute error	R2 score	Explained variance	Training time	Inferenc e time	Hyper-Parameters	Cross Valid ation
Acorns.csv	PaloBst	8.137915	8.137915	8.137915	8.137915	8.137915224	271.529	132.1399	{'distribution': 'gaussian', 'learning_rate': 0.1, 'n_estimators': 100, 'subsample': 0.7}	1
Acorns.csv	RGFR	8.919651	8.919651	8.919651	8.919651	8.91965146	7.650091	3.241837	{'l2': 0.1, 'max_leaf': 500, 'learning_rate': 0.5}	1
Acorns.csv	Accelerated	6.357838	6.357838	6.357838	6.357838	6.357838176	33.13243	0.466029	{'descent': 'gradient', 'loss': 'ls', 'learning_rate': 0.1, 'n_estimators': 100}	1
Acorns.csv	CatBoost	25.65528	25.65528	25.65528	25.65528	25.65527875	84.60958	0	{'loss_function': 'RMSE', 'silent': True, 'depth': 4, 'l2_leaf_reg': 5, 'learning_rate': 0.25}	1
Acorns.csv	PaloBst	5.402222	5.402222	5.402222	5.402222	5.402222222	342.2682	125.0212	{'distribution': 'gaussian', 'learning_rate': 0.1, 'n_estimators': 100, 'subsample': 0.7}	2
Acorns.csv	RGFR	12.48043	12.48043	12.48043	12.48043	12.48042795	5.094645	1.597921	{'l2': 0.1, 'max_leaf': 500, 'learning_rate': 0.5}	2
Acorns.csv	Accelerated	6.14607	6.14607	6.14607	6.14607	6.146070299	35.07403	1.006881	{'descent': 'gradient', 'loss': 'ls', 'learning_rate': 0.1, 'n_estimators': 100}	2
Acorns.csv	CatBoost	21.37481	21.37481	21.37481	21.37481	21.37481212	86.3191	0	{'loss_function': 'RMSE', 'silent': True, 'depth': 4, 'l2_leaf_reg': 1, 'learning_rate': 0.150	2
Acorns.csv	PaloBst	24.67162	24.67162	24.67162	24.67162	24.67161882	350.8861	0	{'distribution': 'gaussian', 'learning_rate': 0.1, 'n_estimators': 100, 'subsample': 0.7}	3
Acorns.csv	RGFR	31.28025	31.28025	31.28025	31.28025	31.2802456	5.769379	2.485553	{'l2': 0.1, 'max_leaf': 500, 'learning_rate': 0.5}	3
Acorns.csv	Accelerated	35.03548	35.03548	35.03548	35.03548	35.03548018	33.90393	1.259883	{'descent': 'gradient', 'loss': 'ls', 'learning_rate': 0.1, 'n_estimators': 100}	3
Acorns.csv	CatBoost	31.89976	31.89976	31.89976	31.89976	31.89975684	83.52129	0	{'loss_function': 'RMSE', 'silent': True, 'depth': 4, 'l2_leaf_reg': 1, 'learning_rate': 0.150	3
Acorns.csv	PaloBst	5.769213	5.769213	5.769213	5.769213	5.769212773	352.9349	0	{'distribution': 'gaussian', 'learning_rate': 0.1, 'n_estimators': 100, 'subsample': 0.7}	4
Acorns.csv	RGFR	13.84628	13.84628	13.84628	13.84628	13.8462796	7.869381	2.67897	{'l2': 0.1, 'max_leaf': 500, 'learning_rate': 0.5}	4
Acorns.csv	Accelerated	13.29991	13.29991	13.29991	13.29991	13.29991282	31.88163	2.493302	{'descent': 'gradient', 'loss': 'ls', 'learning_rate': 0.1, 'n_estimators': 100}	4
Acorns.csv	CatBoost	20.05149	20.05149	20.05149	20.05149	20.0514935	87.32808	0	{'loss_function': 'RMSE', 'silent': True, 'depth': 4, 'l2_leaf_reg': 1, 'learning_rate': 0.3}	4
Acorns.csv	PaloBst	18.99447	18.99447	18.99447	18.99447	18.9944679	334.0331	0	{'distribution': 'gaussian', 'learning_rate': 0.1, 'n_estimators': 100, 'subsample': 0.7}	5
Acorns.csv	RGFR	26.44203	26.44203	26.44203	26.44203	26.44203045	6.668255	3.060818	{'l2': 0.1, 'max_leaf': 500, 'learning_rate': 0.5}	5
Acorns.csv	Accelerated	24.97568	24.97568	24.97568	24.97568	24.97567524	30.62917	1.449426	{'descent': 'gradient', 'loss': 'ls', 'learning_rate': 0.1, 'n_estimators': 100}	5
Acorns.csv	CatBoost	22.88855	22.88855	22.88855	22.88855	22.88855251	85.44123	0	{'loss_function': 'RMSE', 'silent': True, 'depth': 4, 'l2_leaf_reg': 7, 'learning_rate': 0.3}	5

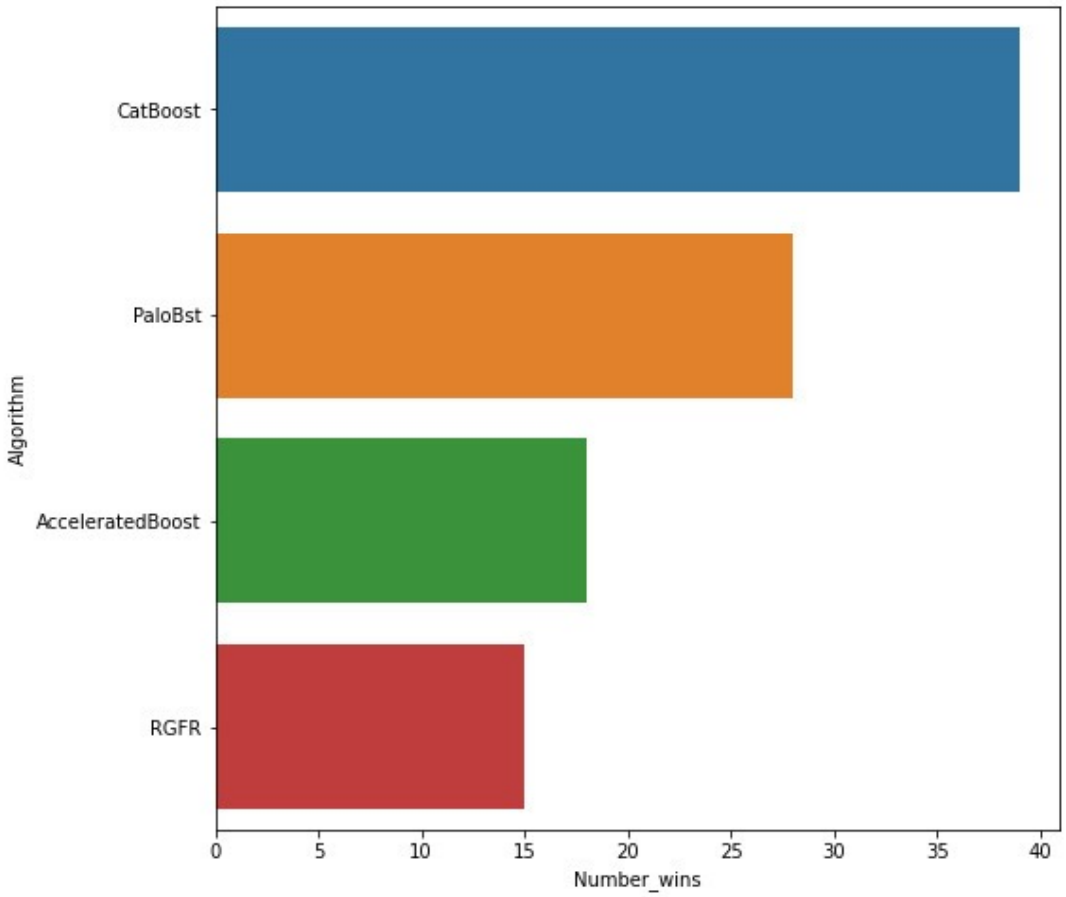
Also, the table is saved in a separate file "table_of_results" in the "results" folder in csv and xlsx formats.

Statistical significance of the results

In this section, we want to know which algorithm showed the best results on previous 100 data sets.

We calculated the number of victories for each algorithm (best MSE):

	Algorithm	Number_wins
1	CatBoost	39
0	PaloBst	28
3	AcceleratedBoost	18
2	RGFR	15



CatBoost won the most victories, but to make sure that the superiority of his performance is statistically significant, we will conduct a statistical test.

In the statistical test, we have chosen estimator measure MSE as the compared parameter. For each algorithm, the average value of this parameter was calculated for each dataset.

The resulting table was reduced to the following form:

Dataset name	AGB	CatBoost	PaloBoost	RGF
Acorns.csv	21.04687	22.8564	16.19627	20.10768
Admission_Predict_kaggle.csv	0.00505	0.005317	0.004638	0.005328
Automobile.csv	5489450	4949370	7060620	5927815
Bike Sharing in Washington_kaggle.csv	13964.18	7830.519	30777.81	5024.007
Concrete_Data_Yeh_kaggle.csv	21.43832	17.44357	48.59362	16.79441
Crash.csv	169872.5	178627.7	162164.8	182685.8
ERA.csv	2.525545	2.54087	2.510213	2.509711
EgyptianSkulls.csv	2769240	3086100	2456577	2785353
FacultySalaries.csv	6.259689	6.001416	6.86532	7.608284
ICU.csv	0.175584	0.14801	0.199874	0.195414
IMDB-Movie-Data_kaggle.csv	158.9466	179.607	157.3734	165.6822
MercuryinBass.csv	0.017734	0.017368	0.023316	0.029602
SMSA.csv	1946.101	1829.236	2125.154	2492.82
airfoil_self_noise_uci.csv	3.934706	2.597112	7.957101	2.484899
analcatdata_apnea1.csv	1744013	1169005	5539864	1151017
analcatdata_birthday.csv	113899.5	88792.46	330656.9	81140.91
analcatdata_chlamydia.csv	67986828	21626725	92873857	81947970
analcatdata_election2000.csv	1.26E+09	8.37E+08	1.62E+09	3.21E+09
analcatdata_galapagos.csv	5369.061	6582.816	8048.072	10741.35
analcatdata_gsssexsurvey.csv	0.821693	0.78969	0.662628	0.862132
analcatdata_gviolence.csv	91578.48	104365.6	95485.68	120461.9
analcatdata_michiganacc.csv	27.07519	24.98519	24.42225	25.79318
analcatdata_ncaa.csv	77.20501	73.49479	67.80456	72.68377
analcatdata_neavote.csv	1.242866	1.256132	13.04794	1.257392
analcatdata_negotiation.csv	0.890041	0.8962	0.798771	0.942495
analcatdata_olympic2000.csv	7.87E+11	7.02E+11	1.11E+12	1.27E+12
analcatdata_runshoes.csv	2.097502	1.40138	2.099787	2.891438
analcatdata_seropositive.csv	427.7887	248.3428	624.0726	635.4284
analcatdata_vehicle.csv	36246.91	30809.23	98742.19	41401.06
analcatdata_vineyard.csv	6.283618	4.333157	14.31989	4.912303
analcatdata_wildcat.csv	12.78904	13.20831	11.70599	11.49731
arsenic-female-bladder.csv	9.520639	3.773596	24.33075	7.242373

auto93.csv	38.83192	29.13049	43.42902	41.68945
autoHorse.csv	192.8077	155.4395	208.0179	152.1726
autoMpg.csv	8.181714	7.529361	8.653448	7.585243
auto_price.csv	6897135	6585968	8335105	6323088
baseball-team.csv	1.02E+10	9.31E+09	1.11E+10	1.24E+10
baskball.csv	0.00984	0.009793	0.008551	0.012184
bolts.csv	85.60697	59.62274	177.8857	302.7172
boston.csv	12.17343	10.56139	18.50463	10.89759
breastTumor.csv	104.5193	150.716	103.6604	108.7722
cholesterol.csv	2969.537	3475.056	2960.246	3284.408
chscase_census2.csv	0.266849	0.309517	0.260505	0.312526
cocomo_numeric.csv	112143.2	116490.6	223678.2	305229.6
cps_85_wages.csv	21.33213	24.77675	20.54499	21.68206
cpu_with_vendor.csv	2209.288	2082.453	7366.685	4152.49
detroit.csv	3497.182	3022.287	4818.698	7373.815
diabetes_numeric.csv	0.45711	0.518018	0.504639	0.630478
diggle_table_a1.csv	0.533198	0.517448	0.496998	0.782867
disclosure_x_tampered.csv	8.69E+08	1.23E+09	8.66E+08	9.91E+08
echoMonths.csv	177.2073	184.9245	257.1712	197.2879
fishcatch.csv	7186.708	4351.282	15504.33	31567.61
forest-fires.csv	2669.75	3597.055	2542.46	3473.267
fri_c2_100_25.csv	0.43048	0.446609	0.576292	0.406439
fruitfly.csv	307.7674	444.6735	268.156	374.2092
gascons.csv	185.4731	152.3484	309.8138	1802.251
happinessRank_2015.csv	0.003762	0.025575	0.05013	0.024155
house_prices_kaggle.csv	1.13E+09	1.01E+09	1.48E+09	1.07E+09
hutsof99_child_witness.csv	156.5748	107.0541	129.1821	241.0986
hutsof99_logis.csv	96.11812	93.46904	99.21737	98.63769
iq_brain_size.csv	232.9833	226.3851	173.3065	187.1366
kc1-numeric.csv	91.99089	62.7861	80.42534	95.54067
kdd_coil_7.csv	23.55665	20.75172	21.22518	22.32603
liver-disorders.csv	9.317069	10.97307	8.689708	10.42274
longley.csv	1558709	1696036	4452805	14673147
machine_cpu.csv	3940.795	4251.653	6650.881	5490.114
mbagrade.csv	0.185765	0.194429	0.13695	0.133172
mu284.csv	48.29703	50.06858	56.46537	48.13247
nasa_numeric.csv	1592868	517983.9	981658.7	758751.2
nflpass.csv	31.5139	33.61752	17.66881	15.4346
papir_2.csv	1.363536	1.236809	1.435087	3.318867
pbc.csv	0.572388	0.578814	0.557075	0.58877
pharynx.csv	104091	108790.2	167043.9	110027.3
places.csv	1050844	1114905	1020970	1152065
pollution.csv	2215.567	2183.609	2500.068	2506.855

pubexpendat.csv	2746.54	2256.046	3006.171	3941.311
pwLinear.csv	3.125786	3.796505	14.05822	3.245979
pyrim.csv	0.011574	0.008555	0.008725	0.011586
quake.csv	0.03652	0.042275	0.03663	0.037327
rmftsa_ladata.csv	3.668313	4.131506	4.30155	4.340269
sensory.csv	0.526774	0.642535	0.676397	0.488534
servo.csv	0.42352	0.362046	1.866914	0.400427
sleep.csv	0.395631	0.36217	0.371775	0.42559
sleuth_case1102.csv	17152.62	14518.67	14959.91	17455.03
sleuth_ex1221.csv	1979.551	1862.28	2039.408	2704.855
sleuth_ex1605.csv	113.3121	94.18921	149.1723	124.9336
sleuth_ex1714.csv	4315035	3714192	3541521	5502558
stock.csv	0.709372	0.56178	1.140599	0.549028
strikes.csv	323634.7	333708.7	303378.7	314335.8
student-performance.csv	1.760859	1.833133	1.924942	1.892526
tecator.csv	1.44989	1.889143	2.194301	2.560653
treepipit.csv	1.180042	0.839254	0.941622	0.840645
triazines.csv	0.020631	0.019333	0.018116	0.019378
vineyard.csv	9.500838	10.49669	9.528355	12.07272
visualizing_environmental.csv	11.29377	13.06018	10.29276	12.30506
visualizing_ethanol.csv	0.053015	0.060256	0.048109	0.056709
visualizing_slope.csv	2.111543	2.33112	7.321681	9.921553
wisconsin.csv	1284.1	1365.66	1252.323	1435.858
witmer_census_1980.csv	30.65259	25.12864	29.89496	55.03277
yacht_hydrodynamics.csv	1.181408	1.915138	18.36178	1.280094

The table is available for review in the folder "results" under the name "table_for_test_friedman", in two formats csv and xlsx.

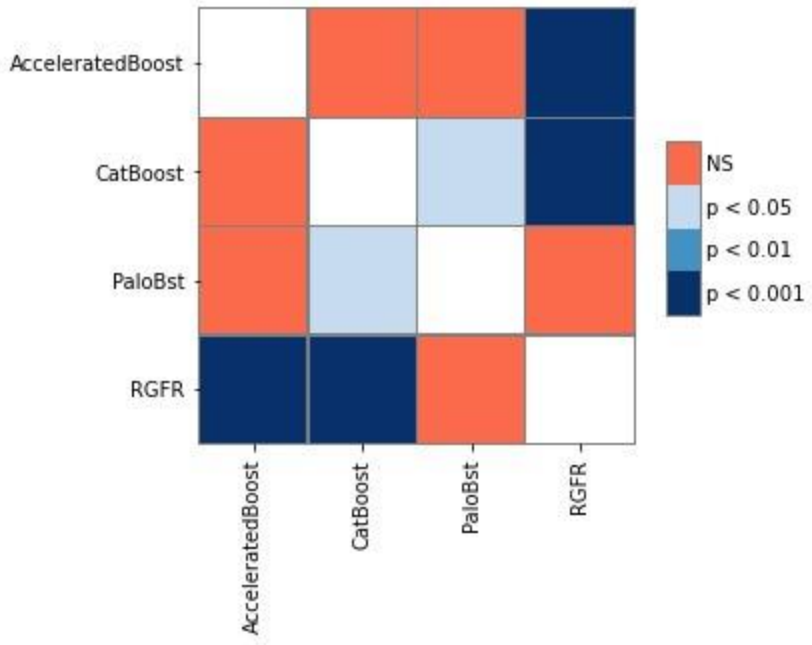
We ran a nonparametric Friedman test. It is available in the scipy library (scipy.stats.friedmanchisquare).

Based on the test results, we rejected the null hypothesis of statistical equality between the results of the algorithms. (p-value = 0.000016).

In this case, a post hoc test was performed, namely we calculated pairwise comparisons using Nemenyi post hoc test.

The following results were obtained:

	AcceleratedBoost	CatBoost	PaloBst	RGFR
AcceleratedBoost	-1.000000	0.900000	0.198718	0.001000
CatBoost	0.900000	-1.000000	0.049306	0.001000
PaloBst	0.198718	0.049306	-1.000000	0.220908
RGFR	0.001000	0.001000	0.220908	-1.000000



From pairwise comparison can bee seen that catboost has a significant difference with RGF and Paloboost (and no difference with AGB).

And in turn, AGB has a significant difference with RGF.

Meta-learning model

In this part, we will build a model that predicts the algorithm that will give the best result for each dataset according the given meta-features.

Our model solves the problem of binary classification (1 - the algorithm gave the best result, 0 - all the others).

We used XGBoost as a base model. We tried to optimize the hyperparameters, but the best results were obtained using the XGBoost out-of-the-box.

To evaluate the quality of the model, we used two metrics: ROC AUC and Accuracy.

The evaluation was carried out using Leave-one-out validation (training is carried out on (*n* -1) datasets, where *n* - number of datasets, and the evaluate on the one remaining).

For XGBoost to work, we have convert all categorical features to numeric.

The final table is available for review in the folder "results" under the name "table_for_meta", in two formats csv and xlsx.

We got the following results:

ROC AUC: 0.6566666666666667

Accuracy: 0.7825

With the following hyperparameters:

```
Model parametrs: {
'base_score': 0.5,
'booster': 'gbtree',
'colsample_bylevel': 1,
'colsample_bynode': 1,
'colsample_bytree': 1,
'gamma': 0,
```

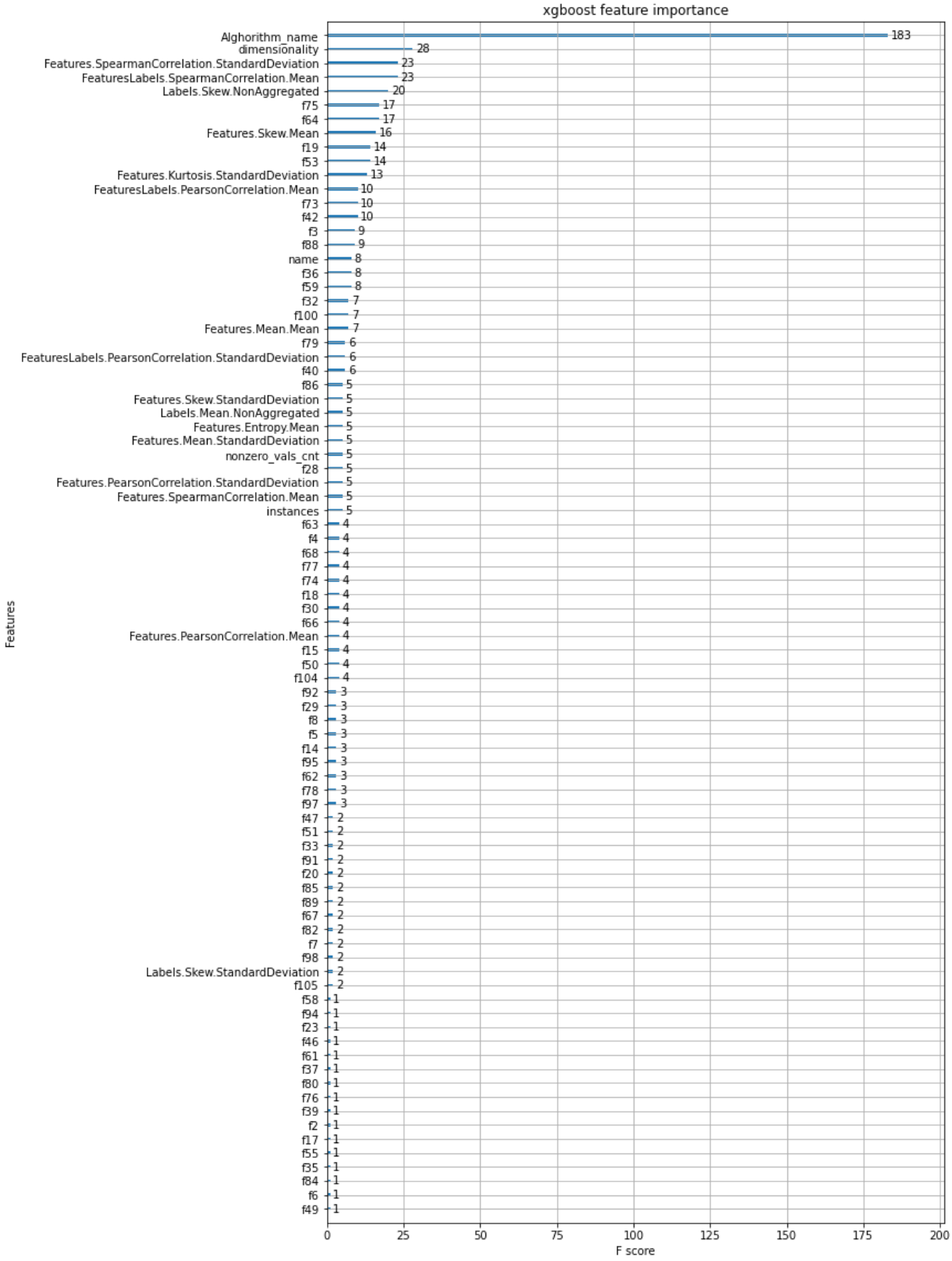
```
'learning_rate': 0.1,
'max_delta_step': 0,
'max_depth': 3,
'min_child_weight': 1,
'missing': None,
'n_estimators': 100,
'n_jobs': 1,
'nthread': None,
'objective': 'binary:logistic',
'random_state': 0,
'reg_alpha': 0,
'reg_lambda': 1,
'scale_pos_weight': 1,
'seed': None,
'silent': None, '
subsample': 1,
'verbosity': 1}
```

We find these to be satisfactory results.

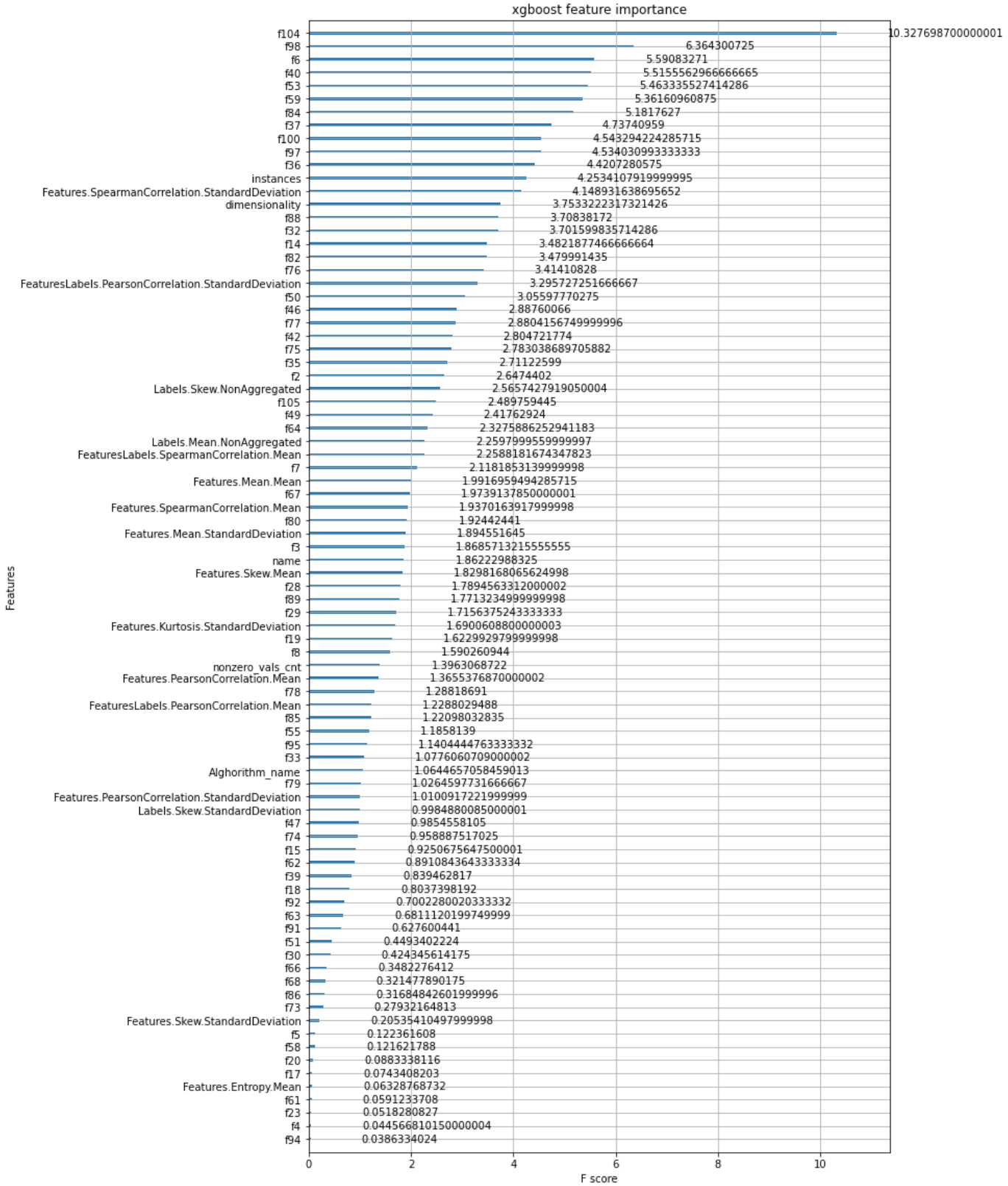
We used a pre-trained XGBoost model and its method feature_importance_ to rank the features by:

- weight
- gain
- cover

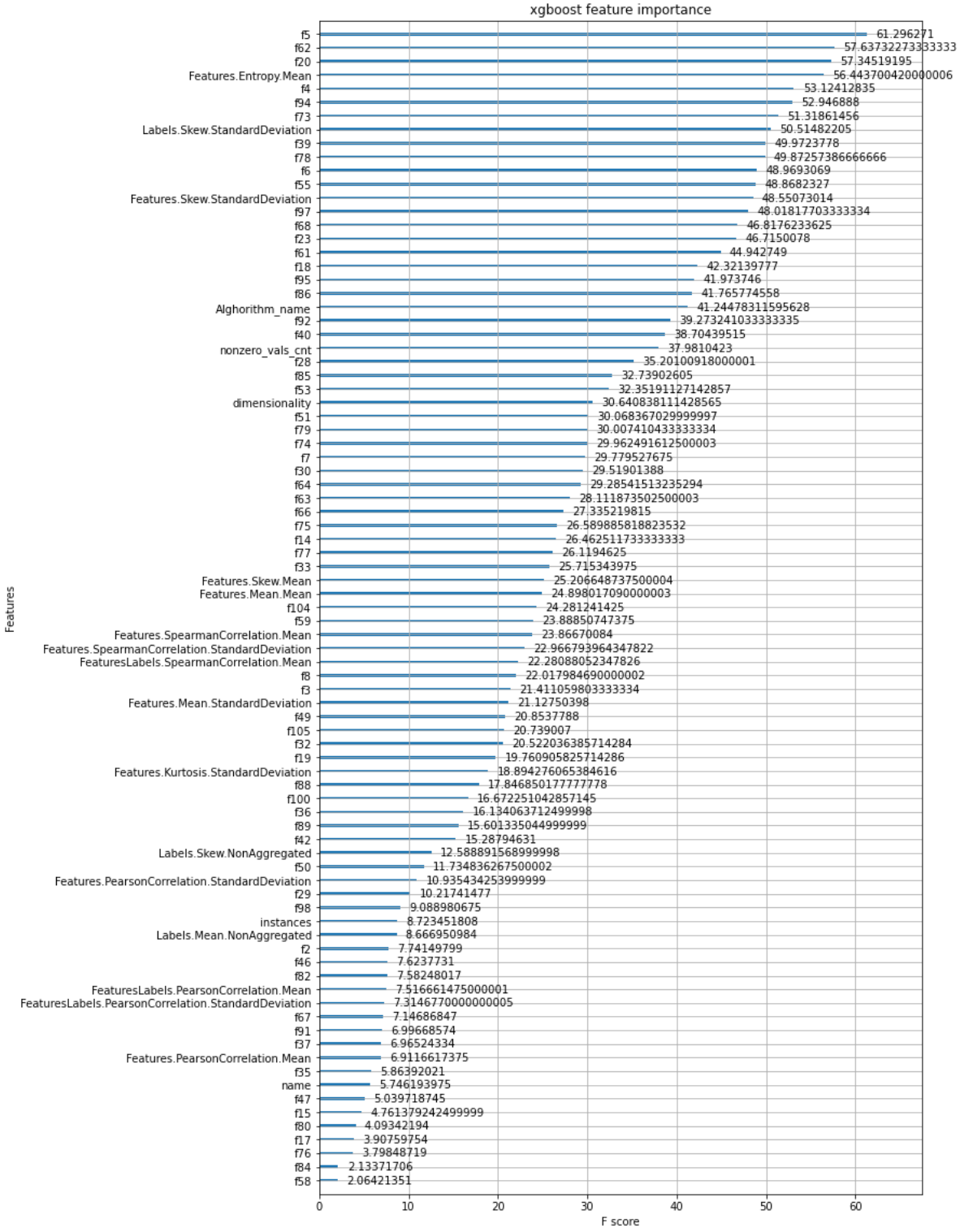
"Weight" is the number of times a feature appears in a tree:



”Gain” is the average gain of splits which use the feature:

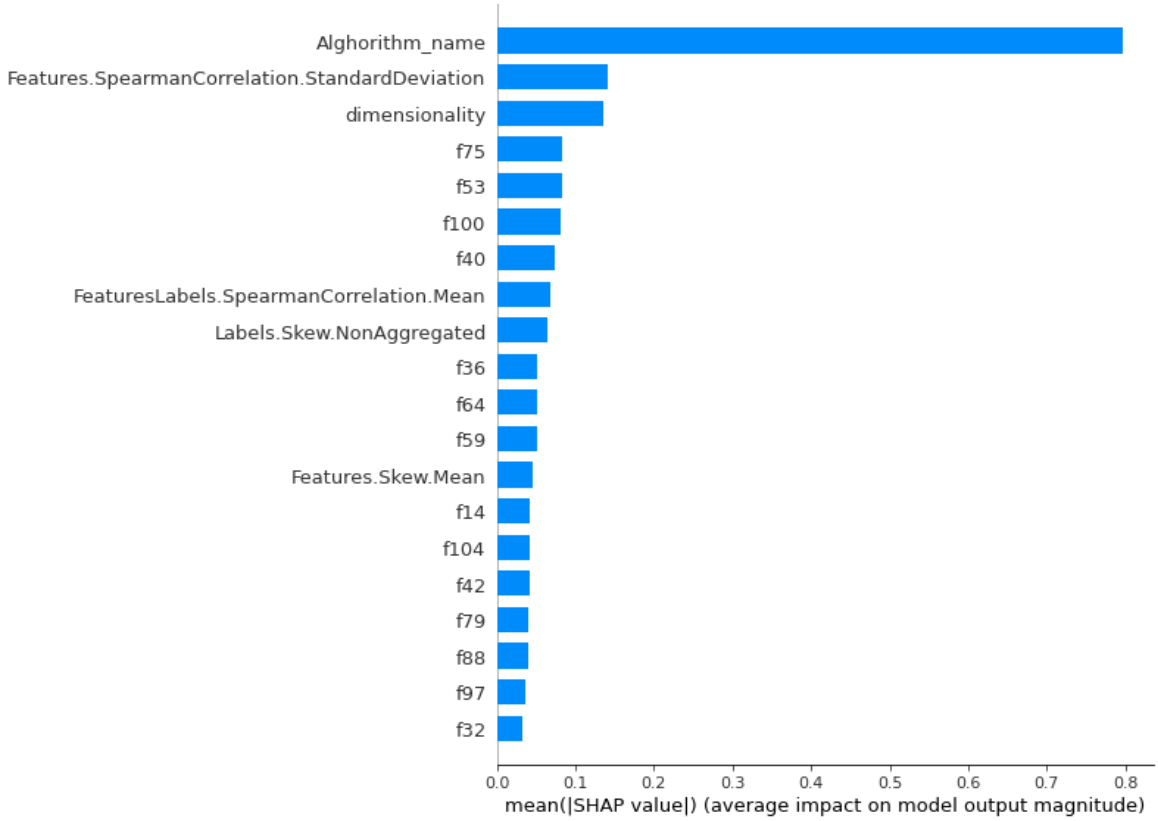


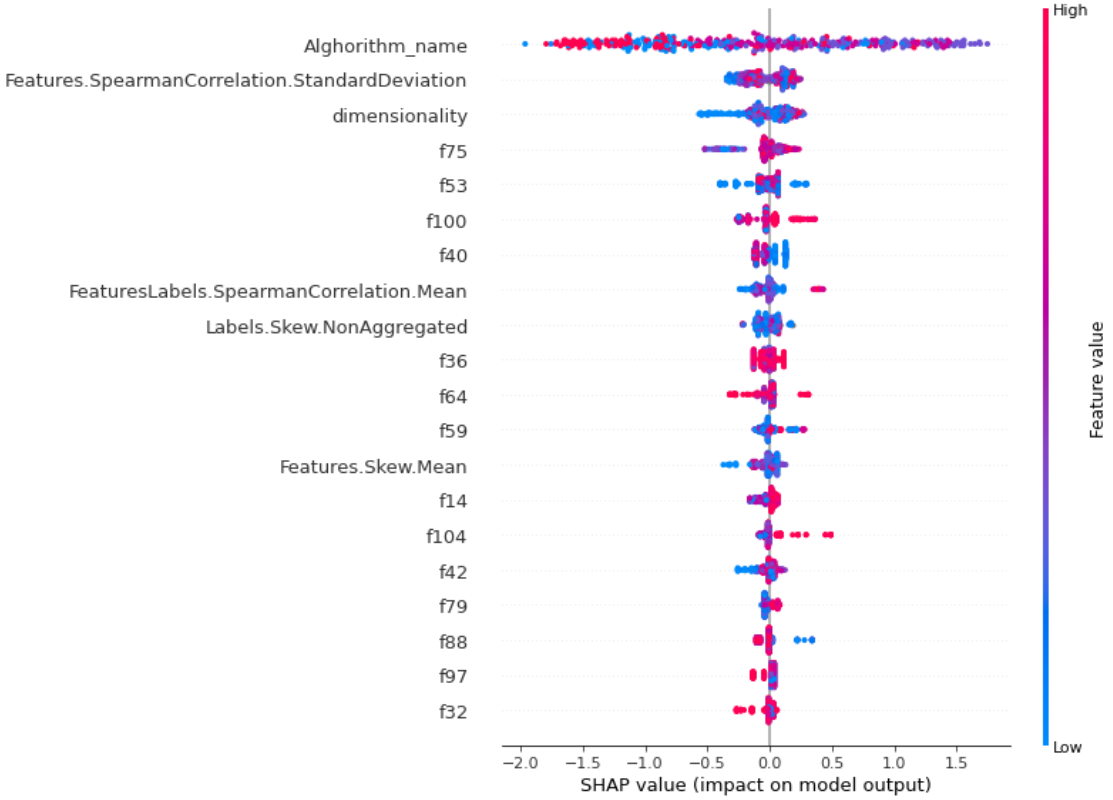
”Cover” is the average coverage of splits which use the feature where coverage is defined as the number of samples affected by the split:



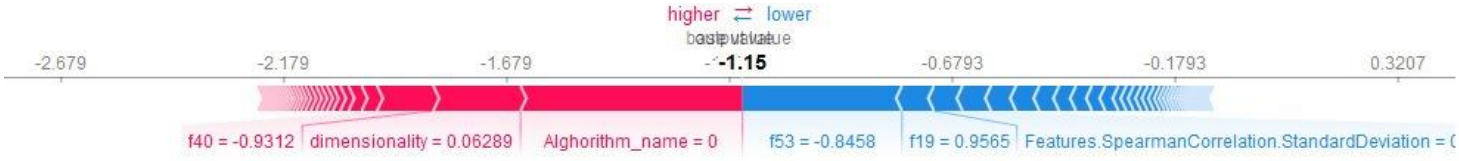
Pictures “weight.png”, “gain.png”, “cover.png” are available in the folder "results”.

Feature importance from SHAP

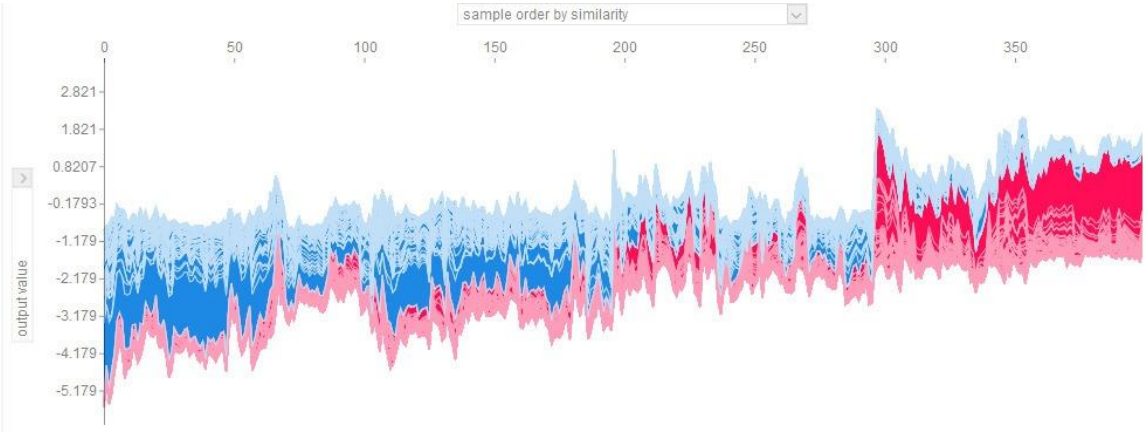




Visualization of single prediction



Visualization of many predictions



As we can see, the Algorithm_name feature brings the greatest weight to the model (it comes first according to the XGBoost and SHAP model).

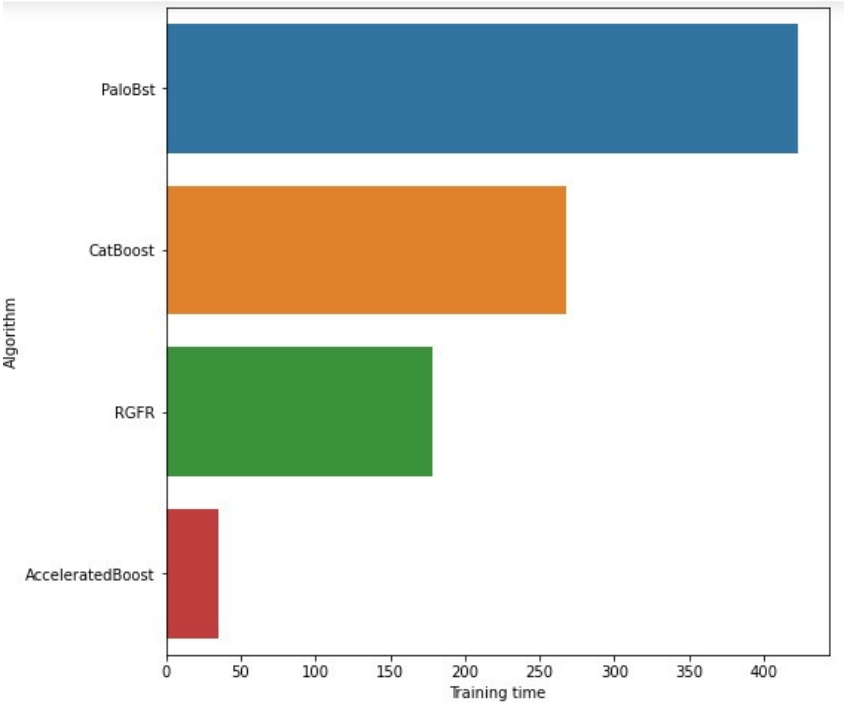
Then Features.SpearmanCorrelation.StandardDeviation and dimensionality follow in XGBoost , and in SHAPE they are reversed. In general, both models equally highlighted the same features as the most important three features, and then the differences begin.

Conclusions

- As part of this project, we researched 3 algorithms:
    - PaloBoost
    - Accelerated Gradient Boosting (AGB)
    - Regularized Greedy Forest (RGF)
- We also evaluated their work and compared their work with the work of the famous CatBoost library using the example of regression.
- The evaluation was carried out on 100 datasets of various sizes. We used MSE as an estimate.
- When using the ranking system and scoring, CatBoost showed the best result, followed by PaloBoost, AGB and RGF.
- Nevertheless, the statistical test showed that there is no significant difference in the MSE results between CatBoost and AGB, on the contrary, there is between CatBoost and PaloBoost.
- Another important point is time. We investigated both the inference time and the training time. The figures show the results of the training time:

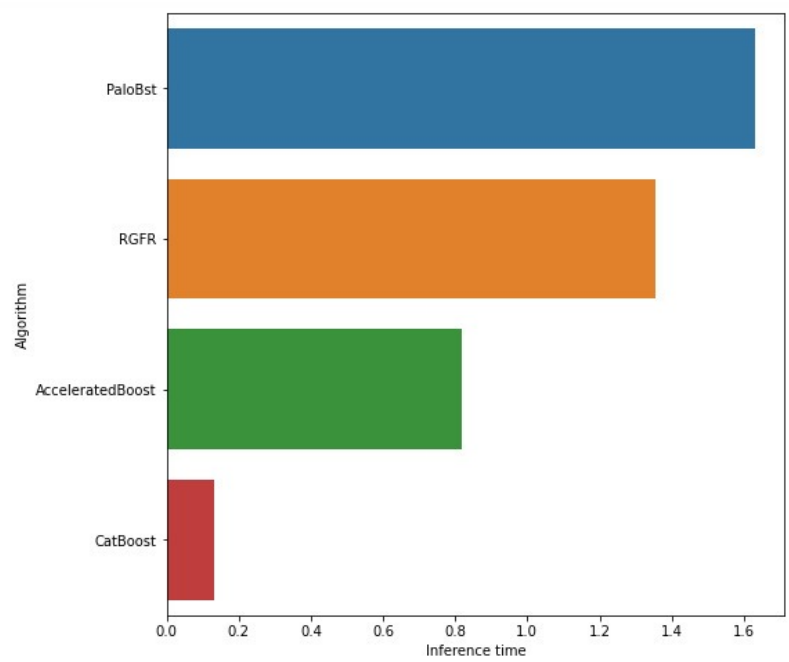
Average training time

	Algorithm	Training time
2	PaloBst	422.754634
1	CatBoost	267.777875
3	RGFR	178.179608
0	AcceleratedBoost	34.815685



- PaloBoost training takes the most time, followed by CatBoost. AGB takes the least amount of time.
- In the following figure, can be seen the results of the time of the predict (inference):

	Algorithm	Inference time
2	PaloBst	1.630968
3	RGFR	1.356760
0	AcceleratedBoost	0.818953
1	CatBoost	0.132337



- PaloBoost still takes the longest. CatBoost takes the least time, followed by AGB.
- We can conclude that AGB can compete with the well-known CatBoost.

Note: all the code is contained in the file “project.ipynb” (Jupyter Notebook)

If there are difficulties with opening, a copy of the file in the “html” format.

REFERENCES

[1] Yubin Park , Joyce Ho (2019). "Tackling Overfitting in Boosting for Noisy Healthcare Data." IEEE Transactions on Knowledge and Data Engineering.

[2] Biau, G., Cadre, B., & Rouvière, L. (2019). Accelerated Gradient Boosting. Machine Learning, 108(6), 971-992.

[3] Rie Johnson, Tong Zhang (2014). "Learning Nonlinear Functions Using Regularized Greedy Forest". IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE.

[4] J. H. Friedman, "Multivariate adaptive regression splines," The Annals of Statistics, vol. 19, no. 1, pp. 1–67, March 1991.

[5] J. H. Friedman, E. Grosse, and W. Stuetzle, "Multidimensional additive spline approximation," SIAM Journal on Scientific and Statistical Computing, vol. 4, no. 2, pp. 291–301, 1983.

[6] G. Biau, A. Fischer, B. Guedj, and J.D. Malley. COBRA: A combined regression strategy. Journal of Multivariate Analysis, 146:18–28, 2016.

[7] Houses (downloaded from http://lib.stat.cmu.edu).

[8] A. Frank and A. Asuncion. UCI machine learning repository [http://archive.ics.uci.edu/ml], 2010. University of California, Irvine, Schoolof Information and Computer Sciences.

[9] Y. Park, J.C. Ho. PaloBoost: an Overfitting-Robust TreeBoost with Out-Of-Bag Sample Regularization Techniques (2018).