Gradient Boosting for Quantitative Finance

Jesse Davis*, Laurens Devos†, Sofie Reyners‡, Wim Schoutens§

February 28, 2023

Abstract

In this paper, we discuss how tree-based machine learning techniques can be used in the context of derivatives pricing. Gradient boosted regression trees are employed to learn the pricing map for a couple of classical, time-consuming problems in quantitative finance. In particular, we illustrate this methodology by reducing computation times for pricing exotic derivative products and American options. Once the gradient boosting model is trained, it is used to make fast predictions of new prices. We show that this approach leads to speed-ups of several orders of magnitude, while the loss of accuracy is very acceptable from a practical point of view. In addition to the predictive performance of these methods, we acknowledge the importance of interpretability of pricing models. For both applications, we therefore look under the hood of the gradient boosting model and elaborate on how the price is constructed and interpreted.

1 Introduction

Financial institutions in the derivatives business have to deal with daily tasks requiring more and more computational power. Managing derivative portfolios involves calculating risk measures, determining hedging positions, computing value adjustments, etc. Researchers in quantitative finance have therefore developed a wide range of models and techniques to perform these computations as accurately as possible. As models become more complex and flexible, however, implementing them requires more computational budget. For instance, pricing (exotic) derivatives with a sophisticated model often involves time-consuming numerical methods, like Monte Carlo simulations. This leads to severe problems when those prices need to be computed in real-time. Both model calibration and pricing should therefore be conducted as efficiently as possible. Further examples can be found in the context of risk management, where portfolios are evaluated for different market scenarios in order to quantify a set of risk measures. This again boils down to the same task: pricing a lot of (similar) derivative products. Although a lot of research has been carried out on speeding up state-of-the-art pricing methods, the fight against computation time is not over yet.

Meanwhile, the field of machine learning has been developing at a high pace. Deep learning algorithms, gradient boosting machines, Gaussian process regression models, random forests, ... now have highly efficient implementations, allowing for a fast evaluation of the resulting models. This is exactly what one needs. Indeed, if we can train a

*KU Leuven, Department of Computer Science, Celestijnenlaan 200A, B-3001 Leuven, Belgium.
†KU Leuven, Department of Computer Science, Celestijnenlaan 200A, B-3001 Leuven, Belgium.
‡KU Leuven, Department of Mathematics, Celestijnenlaan 200B, B-3001 Leuven, Belgium.
§KU Leuven, Department of Mathematics, Celestijnenlaan 200B, B-3001 Leuven, Belgium.

machine to learn the pricing map, it can be applied to compute new prices very efficiently. In practice, this is accomplished by summarizing the (parametric) pricing function into a training set, which is the fuel to train the machine learning model. The trained model can then be called whenever a real-time price is required. The idea of learning the pricing map has already been studied in the literature by a couple of authors. [10] train Gaussian process regression models for pricing vanilla and exotic derivatives with models beyond Black-Scholes. [34] deploy this model for a direct calibration of interest rate models, whereas [9] use Gaussian processes in the context of XVA computations. Many other models rely on artificial neural networks. [25] and [14] train neural nets to learn prices of respectively vanilla options and basket options in the Black-Scholes model, while [27] and [24] employ them for pricing vanilla options according to more advanced models. [21] on the other hand directly trains a neural network to return the calibrated model parameters. We refer to [32] for a recent overview.

To the best of our knowledge, the use of tree-based machine learning methods has not been investigated deeply in the literature. Although a single decision tree may not provide enough flexibility, there are several reasons why an ensemble of decision trees could be interesting in the context of option pricing. Compared to neural networks, for instance, tree ensembles are easier to configure, faster to train and easier to interpret. Tree-based methods can moreover be constrained in a straightforward manner in order to satisfy monotonic relations. In the class of tree ensemble methods, gradient boosted decision trees have shown to be widely successful, see for instance [15], [11], [20] and [31], and are often deployed as solutions to real-world problems. While boosting models excel in their ability to achieve accurate predictions, bagging methods like random forests aim to decrease variance of the estimator rather than decreasing its bias. Random forests might therefore be a good choice for modeling noisy data, while gradient boosting models are well-suited to accurately fit complex patterns.

This article contributes by studying gradient boosted regression tree models to learn the pricing map. An empirical study points out how these models can be deployed and interpreted. The remainder of the paper is organized as follows. In Section 2, the key features of a gradient boosting model are described. Section 3 explains how to apply gradient boosting models in the context of derivative pricing. Numerical experiments illustrate the predictive and computational performance of this approach. In Section 4, we enhance the transparency of these models by elaborating on how predictions are constructed and interpreted. Section 5 comments on the advantages and disadvantages of gradient boosting models compared to other machine learning methods. Section 6 concludes the paper.

2 Gradient boosting machines

The concept of a gradient boosting machine (GBM) was first introduced by Friedman in 2001 [16], and has since become a state-of-the-art machine learning technique with a number of competitive implementations, e.g. XGBoost [6], LightGBM [26] and BitBoost [12]. Since recent gradient boosting implementations have many specific features to improve the accuracy and efficiency of the algorithm, theoretical results are often dedicated to one specific aspect of the boosting model. For instance, [4] concentrates on the convergence of gradient boosting classifiers as the number of training instances increases, while [3] proves the convergence of gradient boosting (from an optimization point of view) as the number of trees tends to infinity. The latter analysis was based on

the conceptual frameworks proposed by [29] and [16]. Other papers focus on optional elements in the algorithm like regularization through early stopping, see e.g. [38]. In this section, we limit the discussion to the main ideas of a gradient boosting model.

Consider a (training) set of observations $(X, \boldsymbol{y}) = \{(\boldsymbol{x_i}, y_i) \mid i = 1, \ldots, n\}$, where each $\boldsymbol{x_i}$ is an input vector of dimension d and y_i is the corresponding output. A GBM is an ensemble method, meaning that multiple base models are combined to produce one powerful model that captures the relation between inputs and outputs. In particular, the composite model F_K takes an additive form,

$$F_K(\boldsymbol{x}) = c + \sum_{k=1}^{K} \nu \, h_k(\boldsymbol{x}), \tag{1}$$

where each $h_k(\boldsymbol{x})$ is a base model, c is a constant and ν is a shrinkage parameter. The base models are parametric functions of one particular family, usually decision trees. In this paper we employ regression trees. Following the notation in [19], we can represent a regression tree that is fit on a set $(X, \boldsymbol{z}) = \{(\boldsymbol{x_i}, z_i) \mid i = 1, \ldots, n\}$ as

$$h_k(\boldsymbol{x}) = \sum_{j=1}^{J_k} \bar{z}_{j,k} \mathbb{1}_{R_{j,k}}(\boldsymbol{x}) \tag{2}$$

where $\mathbb{1}_A(x)$ is the indicator function that equals one if $x \in A$ and zero otherwise, $R_{1,k}, \ldots, R_{J_k,k}$ are distinct, non-overlapping regions in the input space and $\bar{z}_{1,k}, \ldots, \bar{z}_{J_k,k}$ are the averages of the outputs $\boldsymbol{z}$ in the respective regions. The function h_k thus represents a regression tree with J_k leaf nodes, having leaf scores $\bar{z}_{1,k}, \ldots, \bar{z}_{J_k,k}$. The resulting gradient boosting model F_K is a collection of scaled regression trees, and hence takes the form of a piecewise constant function.

A gradient boosting machine is trained in a stage-wise manner, meaning that the regression trees are fit one by one while previously fitted trees are left unchanged. In particular, the model is built as follows. Start with an initial constant guess c. In the kth iteration, add a new decision tree, scaled by the shrinkage factor ν, while keeping the previous model F_{k-1} unchanged,

$$F_0(\boldsymbol{x}) = c \tag{3}$$

$$F_k(\boldsymbol{x}) = F_{k-1}(\boldsymbol{x}) + \nu \, h_k(\boldsymbol{x}). \tag{4}$$

The shrinkage parameter $0 < \nu \leq 1$ is used to avoid individual trees from being too important in the final ensemble. The smaller the shrinkage parameter ν, the more trees are needed to obtain an accurate fit on the training data. The shrinkage parameter is therefore often referred to as the learning rate. The kth decision tree h_k is constructed in a typical greedy top-down manner and minimizes a loss function $\mathcal{L}$. The new tree is constructed such that it maximally improves the current best additive model F_{k-1},

$$h_k = \arg\min_{h} \sum_{i=1}^{n} \mathcal{L}(y_i, F_{k-1}(\boldsymbol{x_i}) + h(\boldsymbol{x_i})). \tag{5}$$

To make this optimization problem feasible in practice, the key idea of Friedman [16] was to let the trees predict the negative gradient values,

$$g_{i,k} = - \left[\frac{\partial \mathcal{L}(y_i, \hat{y})}{\partial \hat{y}} \right]_{\hat{y}=F_{k-1}(\boldsymbol{x_i})}. \tag{6}$$

The tree h_k is then fit to these negative gradients by a least-squares optimization,

$$h_k = \arg\min_h \sum_{i=1}^{n} (g_{i,k} - h(\boldsymbol{x}_i))^2. \tag{7}$$

When the ensemble is trained by a gradient boosting algorithm, the outputs $\boldsymbol{z}$ in Equation (2) thus correspond to the negative gradients $\boldsymbol{g_k} = (g_{1,k}, \ldots, g_{n,k})$. The range of the resulting function F_K is therefore not necessarily restricted to the interval $[\min(\boldsymbol{y}), \max(\boldsymbol{y})]$.

To further improve the predictive performance of GBMs, a technique called DART [35] was introduced with the goal of reducing over-specialization. Over-specialization refers to the problem of trees of later iterations affecting only small numbers of instances, leaving predictions of other instances unchanged. This problem reduces the effectiveness of those trees and can cause overfitting. To combat this issue, a shrinkage parameter can be used, but this merely delays the effects of over-specialization. DART – a dropout technique for GBMs inspired by similar ideas used in the context of neural networks [23] – is a more structural solution. DART mutes a random selection of previously constructed trees, and computes the gradients to be fit by the next tree considering only the non-muted trees. This process diminishes the influence of the early trees and more evenly distributes the learning across all trees in the ensemble. The probability of muting a tree is called the drop rate, and we have set it to 10%.

In the numerical experiments in the next sections, GBMs are trained in the LightGBM[1] framework with DART option in Python. We use a squared loss function, i.e.

$$\mathcal{L}(y_i, \hat{y}) = \frac{1}{2}(y_i - \hat{y})^2, \tag{8}$$

so that the negative gradients $g_{i,k}$ are equal to the residuals $y_i - F_{k-1}(\boldsymbol{x_i})$. Below, we briefly discuss some important characteristics of GBMs that were not introduced yet, but will be used in the remainder of the paper.

Histogram-based splitting: Feature histograms are built to avoid repeatedly scanning all data instances when choosing the tree splits. The algorithm instead only has to evaluate the bins of the histogram. The maximum number of bins can be chosen or tuned.

Stochastic gradient boosting [17]: In some boosting iterations, a random subsample of the training data is drawn without replacement. The base model is fit on this subsample instead of the complete training set. The relative size of the sample and the frequency of training on a subsample are respectively called the bagging fraction and frequency. We fix the bagging frequency at 1, i.e. each iteration uses a new subsample, while the bagging fraction is tuned. An additional random effect can be imposed by randomly sampling a subset of the features on which a tree can split.

Regularization: An L_2-regularization term with weight λ is included in the loss function to penalize trees with high leaf scores. This avoids that trees are focusing too much on learning specific residuals and hence are overfitting the training data. Larger λ-values

[1]The main reasons for choosing the LightGBM framework are its ability to scale well to large datasets and its fast training routine. We expect that a similar study performed with e.g. XGBoost will result in a similar predictive performance. However, note that the parameter definitions slightly differ among different implementations, and hence an alternative hyperparameter search is needed.

correspond to stronger penalties. Another regularization technique consists in imposing a lower bound on the number of observations in each leaf node of a tree.

Table 1 lists the hyperparameters that still have to be tuned for the data sets of interest. Three tuning strategies are often used in practice: grid search, random search [2] and Bayesian optimization ([30], [33]).

- **Grid search:** Specify for each hyperparameter a finite set of potential values. For each combination of these values, i.e. each hyperparameter configuration in the grid, a model is built and evaluated.
- **Random search:** For each hyperparameter, m values are independently sampled from a uniform distribution over a prespecified domain. This results in a set of m randomly chosen parameter settings. As discussed in [2], a random search is more efficient than grid search in high-dimensional hyperparameter spaces. The value of m is typically chosen based on the computational budget available.
- **Bayesian optimization:** Hyperparameter sets are chosen sequentially, i.e. previous trials are used to determine which set of hyperparameters should be investigated in the next iteration. This is done by approximating the unknown objective function, e.g. the function that maps the hyperparameter space to the mean squared prediction error on the validation set, by a probabilistic model.

Note that the outcome of all methods above still depends on the manually chosen intervals of candidate hyperparameters. Since choosing a different method mainly impacts the computation time of the tuning procedure, which can be done at any time in advance whenever there is computational budget available, we decide to perform a thorough grid search. Table 1 presents the parameter search space[2] that we use for both applications in this paper, which requires fitting 2304 different GBM models. We tune the parameters

number of trees (K)	4000, 5000, 6000, 10000
shrinkage parameter (ν)	0.1, 0.15, 0.2, 0.25
maximum number of leaves	8, 16, 24, 32
minimum number of instances per leaf	4, 8, 16, 24
L_2-regularization (λ)	0.8, 1.2, 1.6
bagging fraction	0.3, 0.5, 0.7
maximum number of histogram bins	255

Table 1: Hyperparameter grid for both applications in this study.

via a train-validate-test approach rather than a cross-validation strategy. The reason for this choice is twofold. First, the latter is computationally more demanding. Second, since we are not limited by the amount of data, we can rely on an extended validation set without losing valuable training instances. The optimal parameter configuration is selected by evaluating the corresponding models in a mean squared error sense, in order to place a stronger penalty on large pricing errors. The errors are assessed on training and validation sets each consisting of 10 000 instances.

[2]The maximum number of histogram bins is fixed at 255 throughout the grid search. Once the other hyperparameters were chosen, we checked whether increasing the number of histogram bins resulted in a lower mean squared prediction error on the validation set.

3 Boosting derivative pricing

When pricing complex derivative products, typically no analytic pricing formula exists and one often has to resort to time-consuming numerical methods. In this section, we employ gradient boosting models in order to speed up numerical pricing methods. The main idea of the procedure is as follows.

1. Construct a training set $(X, \boldsymbol{y})$ that reflects the targeted pricing function, applied to the product to be priced. The $n \times d$ matrix X is filled with n uniformly sampled values of the d input parameters. For each of the n parameter combinations, the corresponding model price is calculated with the selected pricing function. These values are stored in $\boldsymbol{y}$. Similarly, a validation and test set are built for the purposes of model tuning and evaluation.

2. Build a GBM model. This comprises both tuning the hyperparameters and training the final model.

3. Apply the trained model in order to quickly estimate the product's price for a new market situation.

The first two steps in this procedure, the training phase, can be done at any time in advance, whenever there is computational budget available. Once the training phase is completed, the final model can be deployed to price the product in real-time, according to the current market situation. Although training is assumed to be done offline, it is worth mentioning that LightGBM offers an extremely fast training framework, which allows for an extensive hyperparameter tuning and hence for a more flexible model. Once the model is trained with the optimal hyperparameter configuration, its pricing accuracy is measured in terms of both average and worst-case prediction errors, i.e. using the maximum absolute error (MAE), average absolute error (AAE), maximum relative percentage error (MRPE) and the average relative percentage error (ARPE) as defined below,

$$\text{MAE} = \max\left\{|y_{true}(i) - y_{pred}(i)|,\ i = 1, \ldots, n\right\}, \tag{9}$$

$$\text{AAE} = \frac{1}{n}\sum_{i=1}^{n} |y_{true}(i) - y_{pred}(i)|, \tag{10}$$

$$\text{MRPE} = \max\left\{\frac{|y_{true}(i) - y_{pred}(i)|}{y_{true}(i)},\ i = 1, \ldots, n\right\}, \tag{11}$$

$$\text{ARPE} = \frac{1}{n}\sum_{i=1}^{n} \frac{|y_{true}(i) - y_{pred}(i)|}{y_{true}(i)}, \tag{12}$$

where y_{true} and y_{pred} refer respectively to the benchmark price and the predicted price[3] On the other hand, the computational performance is measured by comparing the prediction time of the GBM model with the time needed for computing the benchmark prices in the test set.

[3]Note that the predictions y_{pred} are point predictions. Estimating prediction uncertainty in a gradient boosting framework is not trivial and is still actively studied in the machine learning literature. Gradient boosting methods with probabilistic forecasting, e.g. NGBoost [13], do not yet obtain the same predictive performance compared to standard implementations without the probabilistic component.

In the remainder of this section, we demonstrate the methodology for pricing two types of derivative products: exotic derivatives and American put options. We introduce feature engineering procedures and monotonic constraints, tailored to the case of option pricing. Training and test sets for both derivative products are available on `https://doi.org/10.5281/zenodo.4277661`, as well as the Python scripts for training and evaluation of the models.

3.1 Pricing exotic derivatives

We explain the methodology for exotic derivative pricing by means of a bonus certificate. This is a path-dependent derivative product that generates payoffs only at the expiration date T. The payoff of a bonus certificate with bonus level B and knock-out level H is given by

$$\text{payoff} = \begin{cases} S_T & \text{if } \min\limits_{0\leq t\leq T} S_t \leq H \\ \max(B, S_T) & \text{else} \end{cases} \tag{13}$$

where S_t denotes the value of the underlying asset at time t. Suppose that we want to price this product, under the assumption that the underlying asset price behaves as described by the Heston model, [22]. As a benchmark pricing method, we use a Monte Carlo simulation based on 500 000 sample paths for the value of the underlying asset price, constructed by taking daily steps (1/250) up to the maturity date.

3.1.1 Training set-up

The goal is to train a gradient boosting model that mimics the Monte Carlo pricing function. A training set $(X, \boldsymbol{y})$ is constructed as follows. The input matrix X is filled with n observations of the $d = 10$ input parameters: bonus level (B), knock-out level (H), time to maturity (T), interest rate (r), dividend yield (q) and the five Heston parameters, denoted by κ, ρ, θ, η and v_0.[4] The observations for the input parameters are obtained by uniformly sampling according to the ranges in Table 2. For each row of X, the corresponding Heston model price is calculated via a Monte Carlo simulation. In particular, the price dynamics of the underlying asset are simulated with a Milstein path discretization,

$$\begin{aligned} S_{t+\Delta t} &= S_t \left(1 \;+\; (r-q)\Delta t \;+\; \sqrt{v_t \Delta t}\; \epsilon_t^1\right) \\ v_{t+\Delta t} &= v_t \;+\; \left(\kappa(\eta - v_t) - \frac{\theta^2}{4}\right)\Delta t \;+\; \theta\sqrt{v_t \Delta t}\; \epsilon_t^2 \;+\; \frac{\theta^2}{4}\Delta t \left(\epsilon_t^2\right)^2 \end{aligned}$$

where ϵ_t^1 and ϵ_t^2 are correlated standard normal random numbers (with correlation ρ) and $\Delta t = 1/250$. Negative variances v_i are avoided by reflection, i.e. $v_t = |v_t|$. The random numbers are generated once and are used for the calculation of all n prices in the training set. For the validation and test set, two different sets of random numbers are generated. Note that both the bonus level and the knock-out level are expressed as a percentage of the spot price of the underlying (S_0). All calculations are performed with $S_0 = 1$ and prices are to be interpreted relative to S_0.

Although the training set $(X, \boldsymbol{y})$ could perfectly be used to train a GBM model, we can first extend the input matrix X by means of feature engineering. Adding new input

[4] κ = rate of mean reversion, ρ = correlation asset price - volatility, θ = vol-of-variance, η = long run variance, v_0 = initial variance.

Training set				
$B:\ 1.05 \to 1.55$	$H:\ 0.55 \to 0.95$	$T:\ 11M \to 1Y$	$r:\ 2\% \to 3\%$	$q:\ 0\% \to 5\%$
$\kappa:\ 0.2 \to 1.6$	$\rho:\ -0.95 \to -0.25$	$\theta:\ 0.15 \to 0.65$	$\eta:\ 0.01 \to 0.25$	$v_0:\ 0.01 \to 0.25$
Test/validation set				
$B:\ 1.1 \to 1.5$	$H:\ 0.6 \to 0.9$	$T:\ 11M \to 1Y$	$r:\ 2\% \to 3\%$	$q:\ 0\% \to 5\%$
$\kappa:\ 0.3 \to 1.5$	$\rho:\ -0.9 \to -0.3$	$\theta:\ 0.2 \to 0.6$	$\eta:\ 0.02 \to 0.25$	$v_0:\ 0.02 \to 0.25$

Table 2: Input parameter ranges for sampling the training, validation and test set.

parameters, which are transformations of the original inputs, allows us to incorporate domain knowledge about the relation to be learned. More specifically, this helps the algorithm by offering additional, deliberately chosen variables on which the splitting criteria can be built. For instance, for a bonus certificate with a high knock-out level H, the probability of breaching this level - and hence not generating a bonus - is large. The impact of the bonus level B on the price thus depends on the value of H, and the ratio of both levels might provide useful information regarding the price. Including this variable in the training set might result in a better predictive performance, or allow for a simpler model that achieves a similar accuracy. For each observation

$$\boldsymbol{x_i} = (B_i,\ H_i,\ T_i,\ r_i,\ q_i,\ \kappa_i,\ \rho_i,\ \theta_i,\ \eta_i,\ v_{0i}), \qquad i = 1, \ldots, n \tag{14}$$

we compute the values of

$$\frac{H_i}{B_i}, \quad H_i T_i, \quad e^{-r_i T_i}, \quad e^{-q_i T_i}, \quad r_i - q_i, \quad \frac{e^{(r_i - q_i)T_i}}{B_i}, \quad \rho_i \theta_i, \quad \kappa_i \theta_i \quad \text{and} \quad \frac{v_{0i}}{\eta_i} \tag{15}$$

and add them as new input variables. As the splitting procedure of decision trees functions as a feature selection mechanism, generated features that are not useful are ignored and do not affect the predictive performance. Therefore, feature engineering can be seen as a method to extend the space in which trees can be built. The augmented input matrix $\tilde{X}$ consists of 19 columns and $(\tilde{X}, \boldsymbol{y})$ serves as the alternative training set. A test and validation set are generated similarly by sampling according to slightly narrower parameter ranges as given in Table 2 and then adding combined features based on these values. The reason for narrowing the parameter ranges is that we want to reduce extrapolation of sensitive parameters, as this may give rise to larger approximation errors.

While feature engineering only steers the model in the right direction, it does not enforce any structural constraints. In the context of option pricing, however, monotonic behavior with respect to certain parameters might be required. The gradient boosting framework allows to impose monotonic price constraints on the aggregate model when fitting the trees. For any input variable, a monotone relation with the output variable - all other variables kept constant - can be enforced by not creating splits that violate the monotonic constraint. For a bonus certificate, we can impose positive monotonic constraints for B, meaning that the price should increase with the bonus level, and negative monotonic constraints for H and q. Note that monotonic constraints do not only help the model by including more information about the pricing function, they also enhance the interpretability and trustworthiness of the model. The drawback of this approach is that monotonic constraints can only be enforced for individual input variables - the implementation assumes independence - and constraints are hence often violated when feature interactions are added. In the analysis that follows, we therefore do not combine feature engineering with monotonic constraints.

Tuning the hyperparameters of the gradient boosting model - either with the extended input matrix $\tilde{X}$ or the original matrix X, with or without enforcing monotonic constraints - results in the optimal configuration as displayed in Table 3.

	$(\boldsymbol{X}, \boldsymbol{y})$	**FE**	**MC**
number of trees (K)	10000	10000	10000
shrinkage parameter (ν)	0.2	0.2	0.2
maximum number of leaves	16	16	8
minimum number of instances per leaf	24	24	8
L_2-regularization (λ)	0.8	0.8	0.8
bagging fraction	0.5	0.5	0.7
maximum number of histogram bins	255	255	255

Table 3: Optimal hyperparameter configuration for pricing bonus certificates in three settings: training on the original training set $(X, \boldsymbol{y})$ without monotonic constraints, including feature engineering (FE) or including monotonic constraints (MC).

Feature engineering has (in this empirical study) no impact on the optimal hyperparameter setting for pricing bonus certificates. However, note that the final configurations still depend on the chosen parameter grid in Table 1. Adding monotonic constraints changes three parameters - the parameter setting that is optimal in the previous two settings gets only the 408th best validation score in the grid when monotonic constraints are imposed. In all settings, the final model consists of 10 000 trees. This is rather large, but it can be justified by the use of DART. We further want to highlight the optimal lower bound on the number of instances per leaf node, equalling 24 for the models without monotonic constraints. This means that at least 24 prices in the training set are averaged to make a prediction with one of the 10 000 trees. Hence, it serves as a regularization technique to account for the random noise in the Monte Carlo pricer.

3.1.2 GBM's price predictions

Given the optimal hyperparameter settings, three GBM models are trained on three differently sized training sets. The empirical performance of these models is summarized in Table 4.

	$(\boldsymbol{X}, \boldsymbol{y})$			**with feature engineering**			**with monotonic constraints**		
Training set size	10 000	100 000	1 000 000	10 000	100 000	1 000 000	10 000	100 000	1 000 000
In-sample prediction									
MAE	9.10e-04	2.46e-03	6.42e-03	7.91e-04	2.24e-03	5.96e-03	2.98e-03	1.71e-02	2.36e-02
AAE	1.53e-04	4.08e-04	5.74e-04	1.41e-04	3.72e-04	5.41e-04	3.73e-04	9.67e-04	1.12e-03
MRPE	7.26e-04	2.21e-03	5.73e-03	6.98e-04	1.99e-03	5.02e-03	3.04e-03	1.79e-02	2.34e-02
ARPE	1.39e-04	3.68e-04	5.12e-04	1.28e-04	3.34e-04	4.80e-04	3.42e-04	8.76e-04	1.01e-03
Out-of-sample prediction									
MAE	1.37e-02	5.53e-03	3.98e-03	1.46e-02	5.21e-03	4.33e-03	2.44e-02	1.46e-02	7.95e-03
AAE	1.07e-03	6.31e-04	5.73e-04	1.06e-03	5.88e-04	5.33e-04	1.41e-03	8.71e-04	8.05e-04
MRPE	1.15e-02	4.48e-03	3.51e-03	1.11e-02	4.12e-03	3.29e-03	1.89e-02	1.15e-02	6.55e-03
ARPE	9.71e-04	5.74e-04	5.21e-04	9.59e-04	5.33e-04	4.83e-04	1.28e-03	7.96e-04	7.38e-04
Computation time									
Prediction time (s)	43.38	42.58	42.38	42.14	42.18	42.09	33.03	31.68	31.92
Speed-up	× 5690	× 5797	× 5825	× 5858	× 5852	× 5864	× 7472	× 7792	× 7734

Table 4: Performance of GBM models for pricing bonus certificates in the Heston model. Out-of-sample predictions and their computation time[5] correspond to a test set of 100 000 instances.

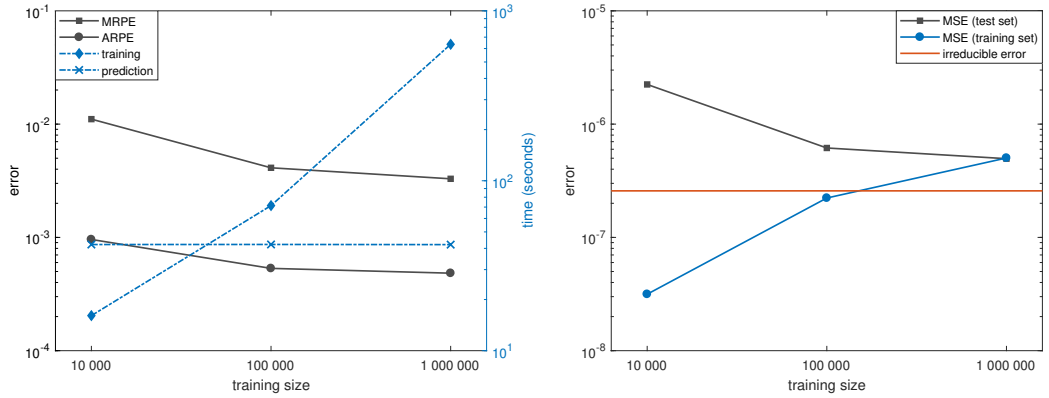
In this setting, feature engineering seems to have only little impact on the out-of-sample predictive performance of the model. In particular, the reduction of the average pricing errors is of order 10^{-5}, while the maximum errors have not decreased in absolute terms. The maximum relative errors have been reduced for each of the three models, with an error reduction of order 10^{-4}. Imposing monotonic constraints leads to a considerably worse predictive out-of-sample performance compared to the standard setting. Indeed, the maximum prediction errors have both approximately doubled, while the increase in average pricing error is of order 10^{-4} for each of the three training sets. This could be explained by the fact that monotonic constraints reduce flexibility[6] of the model. On the other hand, the prediction times of these models are remarkably shorter. More precisely, they have been reduced by approximately 25%, due to the alternative hyperparameter setting - trees have fewer leaves and are thus less complex.

Recall that the hyperparameters of these models were optimized based on a training set of only 10 000 instances. To train the models on larger samples, we did not alter any of the hyperparameters, which might seem counterintuitive. For instance, one would expect that the minimum number of instances per leaf should increase with the number of training instances. However, we did not obtain a better performance when increasing this lower bound and holding all other parameters constant. If we increase the lower bound from 24 to 240 for the model with feature engineering trained on 100 000 instances, we find an out-of-sample MAE of $5.05 \cdot 10^{-3}$, AAE of $6.02 \cdot 10^{-4}$, MRPE of $4.24 \cdot 10^{-3}$ and ARPE of $5.45 \cdot 10^{-4}$. Hence, there is not a one-to-one relation between the minimum number of instances per leaf and the number of training instances.

In the remainder of the discussion we will focus on the models trained on the extended set $(\tilde{X}, \boldsymbol{y})$. In terms of average prediction errors, the GBM model trained on 10 000 instances already achieves decent results. The model is capable of predicting 99 511 of the 100 000 test prices within a 0.5% range of their benchmark value. For the remaining 489 prices the maximum prediction error reaches higher levels, up to a worst-case relative prediction error of 1.11%. Training the GBM model on more observations improves both the maximum and the average predictive performance, while the time needed to make these predictions stays nearly constant. Prediction time indeed does not depend on the number of training instances when the hyperparameter configuration is fixed. For the model trained on 100 000 instances, the worst-case out-of-sample relative error has been reduced to 0.41%. Predicting 100 000 (test) prices takes about 42 seconds, which is almost 6000 times faster than the benchmark pricer. We thus achieve a tremendous speed-up in pricing, while giving up only little accuracy. In Figure 1a, the predictive and computational performance of the three models are summarized visually. Accuracy results are measured out-of-sample and correspond to the relative errors in Table 4. There is a significant improvement when training on 100 000 instances instead of 10 000, while further augmenting the training set to 1 million instances has little impact. The right y-axis in Figure 1a refers to the computation time related to the models, i.e. the time needed for training each model and making 100 000 predictions with it. While prediction time stays roughly constant, training time increases with the number of instances in the training set. However, recall that training needs to be done only once. Figure 1b compares in-sample and out-of-sample mean squared prediction

[5]Machine specifications: Intel(R) Core(TM) i7-7600U CPU @ 2.80GHz, 2901 Mhz, 2 Core(s), 4 Logical Processor(s).

[6]There is evidence that the current implementation for monotonic constraints in LightGBM is too strict, e.g. [1].



(a) Predictive and computational performance. Errors are measured out-of-sample.

(b) In-sample and out-of-sample MSE vs. irreducible error.

Figure 1: GBM performance for bonus certificates in function of the training set size.

errors (MSE) of the three models. The out-of-sample errors decrease with the size of the training set, whereas the in-sample errors increase. The horizontal red line refers to the irreducible error, defined as the variance of the noise (ϵ) in the Monte Carlo pricer. Here, we estimate the irreducible error as the average squared standard error for all Monte Carlo simulated prices in the test set, i.e.

$$Var(\epsilon) \approx \frac{1}{100\ 000} \sum_{i=1}^{100\ 000} \frac{\sigma_i^2}{500\ 000} \tag{16}$$

where $\frac{\sigma_i}{\sqrt{500\ 000}}$ is the standard error corresponding to the Monte Carlo procedure that computes the ith price in the test set. If a model has an in-sample MSE which is lower than the irreducible error, the model is capturing patterns caused by the randomness in the Monte Carlo pricer and is hence overfitting the training data. Based on the test set, we obtain an estimated irreducible MSE of $2.58 \cdot 10^{-7}$. Figure 1b shows that the GBM model trained on a small training set is clearly overfitting, despite all regularization techniques that were used. When the training set size increases, the in-sample and out-of-sample errors converge and the problem seems to be solved. Since the out-of-sample errors are close to the red line, the errors in Table 4 are largely caused by the noise in the benchmark pricing method.

Instead of summarizing the predictive performance into one number for the entire test set, we now look at the predictions individually and the distribution of their errors. Figure 2 shows histograms of the out-of-sample relative errors for the three models, i.e.

$$\frac{BC_{pred}(i) - BC_{true}(i)}{BC_{true}(i)}, \qquad i = 1, \ldots, 100\ 000, \tag{17}$$

where BC_{true} refers to the prices calculated according to the benchmark Monte Carlo method, while BC_{pred} are the prices predicted by the GBM model. The histogram clearly narrows when the training set expands from 10 000 to 100 000 observations, while the improvement is less pronounced when the training set is enlarged once more. In Figure 3, we inspect the predictions of the GBM model trained on 100 000 instances in more detail. In the left panel, the predicted prices are plotted against the benchmark prices. Prices range from 0.95 to 1.45 and their absolute prediction error does not seem to

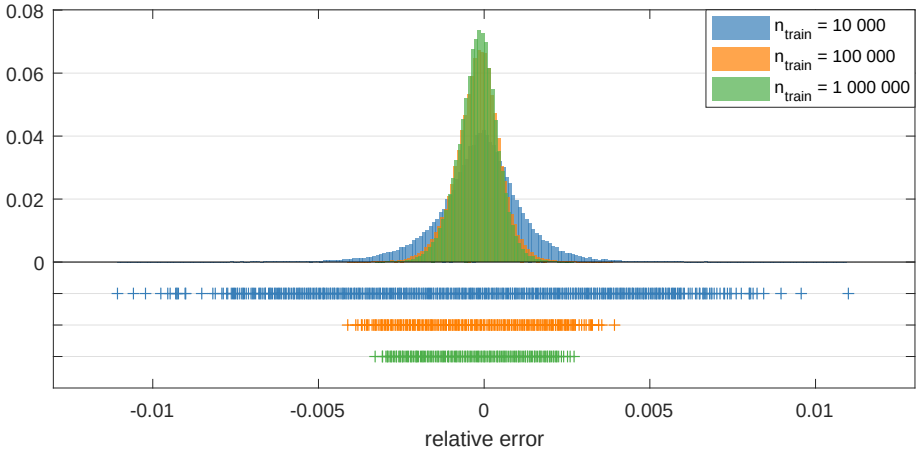
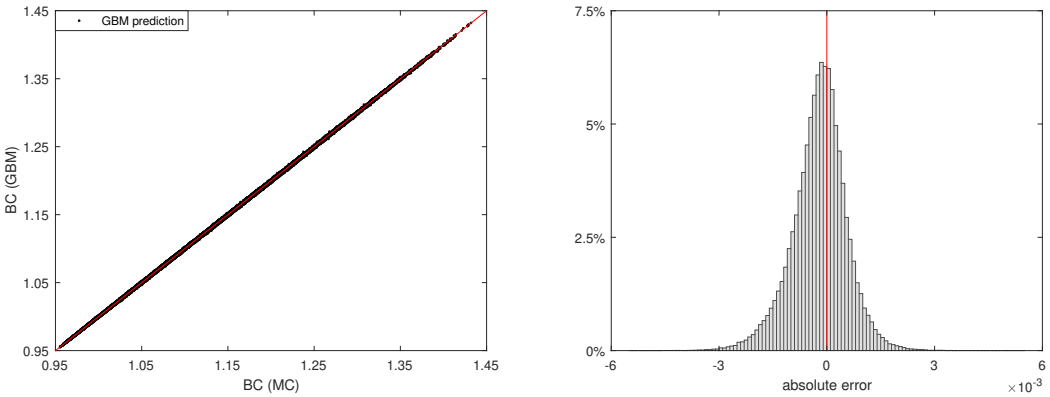


Figure 2: Out-of-sample relative errors of bonus certificates predicted by GBM models trained on respectively 10 000, 100 000 and 1 000 000 instances.

depend on the price level. Figure 3b displays the absolute prediction errors as defined by the numerator in Equation (17). Note that this histogram is not exactly centered around zero. In fact, 60.39% of the predicted prices are lower than their benchmark value. This might be attributed to the right-skewed distribution of training outputs, and to the fact that wider parameter ranges in the training set result on average in higher prices than those in the test set.

3.2 Pricing American options

The second application concerns the pricing of American options. These products are often the reference instruments for calibration, since most liquid options on single stocks are of American type. However, few closed form formulas exist for pricing American options, resulting in a slow calibration procedure. In this section, we focus on pricing American put options according to the binomial tree model [8], where we discretize the



(a) Out-of-sample predictions.

(b) Absolute prediction errors.

Figure 3: Out-of-sample predictive performance of the GBM model for bonus certificates, trained on 100 000 instances.

lifetime of the options in time steps of 1/750.

3.2.1 Training set-up

We first generate a training set $(X, \boldsymbol{y})$ that reflects the binomial tree pricing function for American put options. The input matrix X is filled with n uniformly sampled observations of the five input parameters: strike price (K), time to maturity (T), volatility (σ), interest rate (r) and dividend yield (q). The sampling ranges for training, test and validation set are stated in Table 5. Note again that we implicitly assume the initial value of the underlying asset (S_0) to be equal to 1. For each input observation, the price

Training set				
$K:\ 0.4 \to 1.6$	$T:\ 1.5M \to 12.5M$	$\sigma:\ 5\% \to 105\%$	$r:\ 2\% \to 3\%$	$q:\ 0\% \to 5\%$
Test/validation set				
$K:\ 0.5 \to 1.5$	$T:\ 2M \to 1Y$	$\sigma:\ 10\% \to 100\%$	$r:\ 2\% \to 3\%$	$q:\ 0\% \to 5\%$

Table 5: Input parameter ranges for sampling training, validation and test set.

of the American put option is calculated with a binomial tree and stored in $\boldsymbol{y}$. In order to avoid too many options with zero prices in the data sets, we delete those instances with prices below 0.0001 and replace them by newly sampled values of the input parameters and their corresponding higher price. As indicated in Figure 4a, the training set still contains a lot of low prices. For this reason, we will use the logarithm of the prices as training outputs. Training on log-prices instead of raw prices allows the model to distinguish more carefully between the prices that are close to zero, and it moreover avoids negative price predictions.

Next, we augment the input matrix X with the following transformations of the original input variables:

$$\sigma\sqrt{T}, \quad KT, \quad K\sigma, \quad e^{-rT}, \quad r-q \quad \text{and} \quad \frac{e^{(r-q)T}}{K}. \tag{18}$$

Apart from the time-scaled volatility $\sigma\sqrt{T}$ and the forward moneyness $\frac{e^{(r-q)T}}{K}$, we also include interactions of the strike price with the time to maturity (KT) and the volatility ($K\sigma$), as the impact of the strike level on the option price depends both on how long the option still exists and on how volatile the underlying asset price is. This feature engineering procedure results in a broader input matrix $\tilde{X}$ with 11 feature columns.

Alternatively, we train a GBM model on the set $(X, \log(\boldsymbol{y}))$ subject to five monotonic constraints. A positive association is imposed between respectively K, T, σ, q and the price of the option, while r is enforced to have a negative impact on the option price. Table 6 states the optimal hyperparameter configuration for the GBM model pricing American put options. In this case, three different optimal parameter settings are found. However, in each configuration the number of trees and the maximum number of histogram bins are large, while the minimum number of price observations in a leaf node is set to the lowest possible value, 4. Hence, very specific branches in the trees are not penalized when evaluating the resulting model on the validation set, which is probably related to the lack of noise in the American option prices.

	$(\boldsymbol{X}, \log(\boldsymbol{y}))$	**FE**	**MC**
number of trees (K)	10000	10000	10000
shrinkage parameter (ν)	0.1	0.1	0.25
maximum number of leaves	32	24	8
minimum number of instances per leaf	4	4	4
L_2-regularization (λ)	1.2	0.8	1.2
bagging fraction	0.5	0.5	0.7
maximum number of histogram bins	2047	2047	2047

Table 6: Optimal hyperparameter setting for pricing American put options in three settings: training on the original training set $(X, \log(\boldsymbol{y}))$ without monotonic constraints, including feature engineering (FE) or including monotonic constraints (MC).

3.2.2 GBM's price predictions

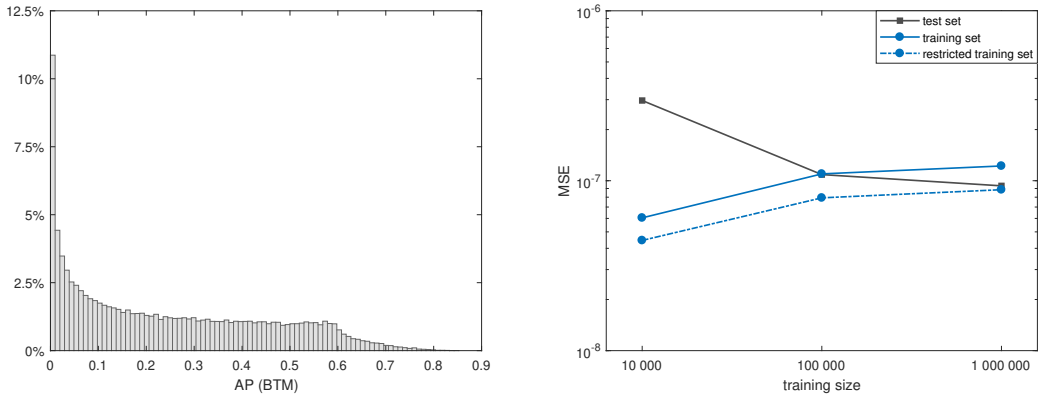
We build GBM models on the original training set $(X, \log(\boldsymbol{y}))$ - with and without monotonic constraints - and on the augmented set $(\tilde{X}, \log(\boldsymbol{y}))$. In each case, three models are trained on respectively 10 000, 100 000 and 1 000 000 training instances. Table 7 summarizes their performance in terms of predictive power and computation time. Prediction errors are computed on the price level, i.e. after exponentiating the raw model predictions. For the sake of completeness, we report the main results for models trained on $(X, \boldsymbol{y})$ in Appendix A.

	$(\boldsymbol{X}, \log(\boldsymbol{y}))$			**with feature engineering**			**with monotonic constraints**		
Training set size	10 000	100 000	1 000 000	10 000	100 000	1 000 000	10 000	100 000	1 000 000
In-sample prediction									
MAE	2.67e-03	4.98e-03	6.70e-03	2.25e-03	5.23e-03	7.53e-03	4.90e-02	4.54e-02	5.90e-02
AAE	2.06e-04	2.91e-04	3.09e-04	1.55e-04	2.15e-04	2.27e-04	1.33e-03	1.27e-03	1.22e-03
MRPE	1.58e-02	2.97e-02	5.64e-02	1.99e-02	2.05e-02	5.04e-02	2.68e-01	1.89e-01	2.54e-01
ARPE	1.10e-03	1.83e-03	2.36e-03	8.74e-04	1.44e-03	1.83e-03	5.82e-03	6.03e-03	6.22e-03
Out-of-sample prediction									
MAE	5.72e-03	3.76e-03	2.98e-03	5.89e-03	3.89e-03	4.39e-03	2.00e-02	1.65e-02	1.45e-02
AAE	5.58e-04	3.12e-04	2.87e-04	3.89e-04	2.28e-04	2.10e-04	1.22e-03	9.81e-04	8.94e-04
MRPE	1.57	3.48e-01	1.28e-01	8.27e-01	3.71e-01	6.94e-02	1.14	3.15e-01	1.32e-01
ARPE	1.39e-02	3.71e-03	2.56e-03	7.69e-03	2.36e-03	1.86e-03	1.21e-02	6.51e-03	5.51e-03
Computation time									
Prediction time (s)	65.57	63.58	63.65	57.13	58.07	57.39	32.97	33.04	32.69
Speed-up	× 61	× 62	× 62	× 69	× 68	× 69	× 120	× 120	× 121

Table 7: GBM performance for American put options with respect to the binomial tree method. Out-of-sample predictions and their computation time correspond to a test set of 100 000 observations.

First of all, we inspect the effect of feature engineering on the resulting out-of-sample model performance. We observe that worst-case absolute prediction errors have become larger by incorporating the engineered features. For the two smallest training sets, the MAE has increased by slightly more than one basis point. When training on 1 million instances, feature engineering leads to an unexpected increase in MAE of 14 basis points. However, the other three pricing measures report a higher accuracy due to the feature engineering procedure. For instance, the worst-case relative pricing error is compressed below 7%, while it equalled 12.8% for the standard model trained on $(X, \log(\boldsymbol{y}))$. Feature engineering has in this case also impact on computation time of the resulting model. For each of the three models, the prediction time has decreased by roughly 10%. The feature

engineering procedure itself has no direct impact on the prediction time, but the resulting hyperparameter setting has. The main driver is in this case the maximum number of leaves which has been reduced from 32 to 24. This results in a smaller model and hence a faster computation of predictions. Enforcing monotonic constraints results in overall larger prediction errors, both in-sample and out-of-sample. While the out-of-sample MAE has increased by more than 10^{-2}, the MRPE has barely been affected by imposing monotonic constraints. The average absolute errors have increased by 6 to 7 basis points due to the constraints. On the other hand, predictions are computed twice as fast as with the standard model, again due to the reduced number of leaves in the trees.



(a) Training set prices.

(b) Mean squared errors.

Figure 4: Left: histogram of the option prices in the training set. Right: in-sample and out-of-sample predictive performance of the three GBM models.

The remainder of the discussion focuses on the models built on $(\tilde{X}, \log(\boldsymbol{y}))$. The absolute errors in Table 7 show that the worst-case prediction error is of order 10^{-3} for the three models, while the average prediction error equals approximately two basis points when the training set is large enough. Again, we observe that in-sample errors increase when the training set expands, while out-of-sample predictive performance improves. The same evolution can be seen in the mean squared prediction errors in Figure 4b. For large training sets, the in-sample MSE becomes even larger than its out-of-sample counterpart, due to the slightly different sampling ranges of the input parameters. The average option price is indeed higher in the training set than in the test set. Ignoring the training instances with input parameters that exceed the test bounds in Table 5 leads to an adjusted in-sample MSE as depicted by the dashed line in Figure 4b. Since (adjusted) in-sample and out-of-sample mean squared errors converge to a similar value, the GBM model does not seem to suffer from overfitting when the training set is large enough. Focusing again on computation time, we observe that each model needs approximately 57 seconds to predict the 100 000 prices in the test set. This is roughly 70 times faster than calling the binomial tree model. However, note that this speed-up is computed for a test set containing maturities that range from two months to one year. Since the computation time in the binomial tree model grows quadratically with the maturity, the effective speed-up for long-term options will be even higher than the values reported in Table 7.

Figure 5 visualizes the distribution of the out-of-sample prediction errors for each of

the three models. The prediction errors are in these histograms defined as

$$AP_{pred}(i) - AP_{true}(i), \qquad i = 1, \ldots, 100\ 000, \tag{19}$$

where AP_{true} refers to the price calculated with the benchmark binomial tree model and AP_{pred} to the price predicted by the GBM model. We recognize an overall improvement

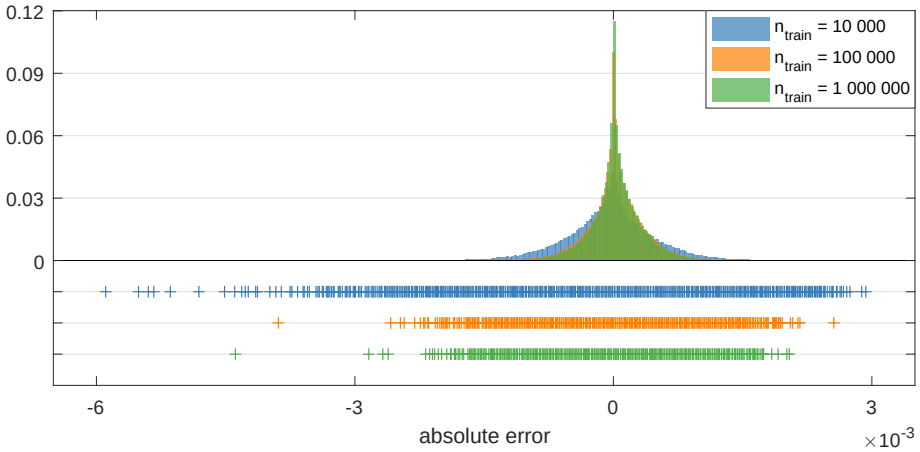
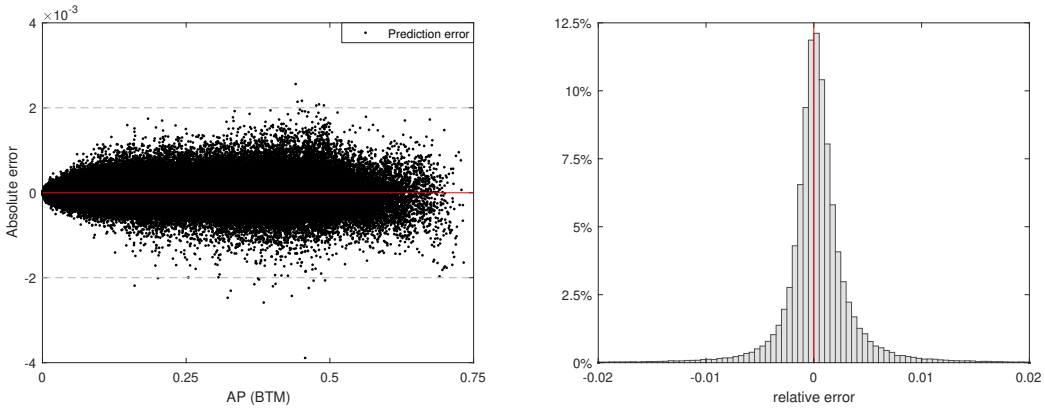


Figure 5: Out-of-sample prediction errors for GBM models trained on sets of 10 000, 100 000 and 1 000 000 instances.

in predictive performance when extending the training set from 10 000 to 100 000 observations. Training on 1 million observations improves the model only a little. In each case, the error distribution has a long left tail in which options are undervalued. The skewness of the error distribution might be induced by training the GBM models on the logarithm of the option prices instead of the raw prices. Figure 6 focuses on the out-of-sample predictive performance of the model trained on 100 000 observations. Figure 6a plots the absolute prediction errors defined by Equation (19) against the benchmark prices calculated with the binomial tree model. Absolute prediction errors are smaller for low prices, which could again be attributed to the log-transformation. Figure 6b presents the relative prediction errors, i.e.

$$\frac{AP_{pred}(i) - AP_{true}(i)}{AP_{true}(i)}, \qquad i = 1, \ldots, 100\ 000. \tag{20}$$

Note that Figure 6b does not contain all relative prediction errors, as 1.03% of the predicted prices deviate more than 2% from the benchmark price. This might seem like a deal-breaker to implement a GBM model in practice. Therefore, we take a closer look at the instances for which those errors occur. If a pattern can be observed, one could raise a red flag when a prediction with these characteristics is made. As expected, high relative errors only occur when the predicted price is low. Indeed, all price predictions higher than 0.025 ($\times S_0$) have relative errors well below 2%. Alternatively, the instances with poor relative prediction errors can be characterized as those for which either the predicted price is below 0.008 or the interaction $\sigma\sqrt{T}K$ is below 0.16. Hence, higher accuracy on low prices can be achieved when an extra condition is imposed on the volatility and strike price. A better overall relative performance for low prices might be obtained by training the model with a loss function tailored to this objective, e.g. a mean relative error loss function.



(a) Absolute prediction errors.

(b) Relative prediction errors.

Figure 6: Out-of-sample predictive performance of the GBM model for American put options, trained on 100 000 instances.

Another source of errors could be extrapolation, due to the randomly sampled training sets. We therefore examine the predictions of test instances located outside of the convex hull of the training set with 100 000 instances. Table 8 shows that the relative errors of these predictions are on average higher than for the predictions of instances that belong to the convex hull. We moreover find that 22.32% of these predictions deviate

	MAE	**AAE**	**MRPE**	**ARPE**
outside of convex hull	1.94e-03	1.87e-04	3.71e-01	1.53e-02
inside convex hull	3.89e-03	2.29e-04	2.56e-01	2.29e-03

Table 8: Out-of-sample error measures separated by the location of test instances: inside the convex hull (99 404 instances) vs. outside of the convex hull (596 instances) of the training set.

(relatively) more than 2% from the true price, whereas only 0.90% of the predictions inside the convex hull exceed this target. Hence, the largest relative errors stem from extrapolation, however, restricting the test set to the convex hull of the training set is not sufficient to exclude all relative errors above this target. Finally, Table 8 indicates that extrapolation is not responsible for the largest absolute errors.

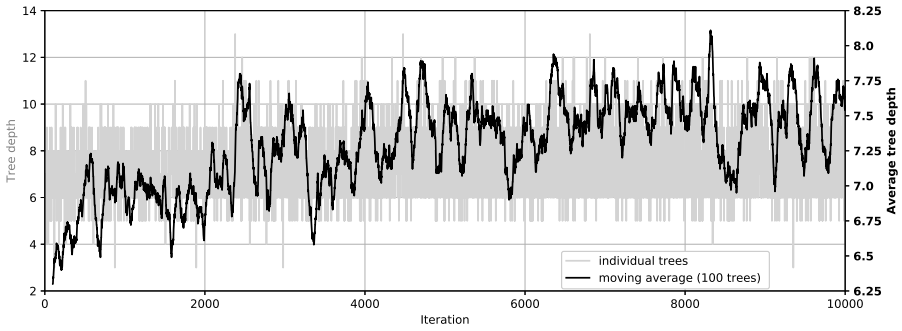
4 Boosted predictions explained

In this section we shift our attention from predictive performance to interpretability of the pricing models. For both applications, we look under the hood of the gradient boosting model and try to reveal how the price is constructed and interpreted. We focus on the models built with feature engineering procedure, unless specified otherwise.

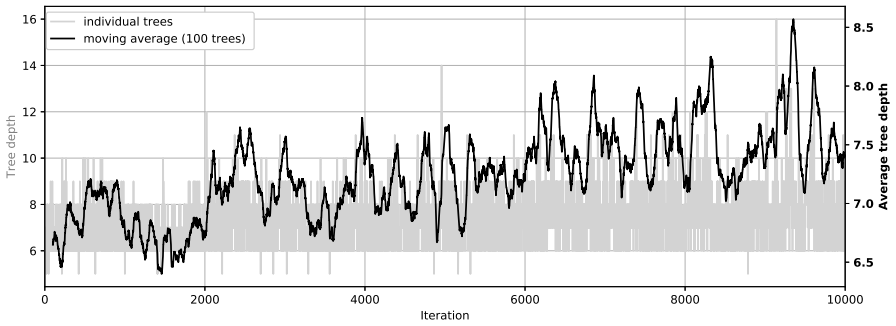
4.1 Model structure

We examine the structure of each tree in the ensemble by considering the tree depth, number of leaves in each tree and the number of training observations in each leaf. We focus on the models trained on 100 000 observations.

First of all, we inspect how the depth of trees evolves when the number of trees in the ensemble grows. Note that we did not impose an explicit upper bound on tree depth when training the model. The depth is thus only limited by the maximum number of leaves in each tree minus one. Figure 7 shows that the trees in the GBM model for bonus certificates have depth between 3 and 13, while the model for American put options has trees with depth between 5 and 16. When considering the individual tree depths plotted in grey, there seems not to be a clear difference between early fitted trees and trees that are included in the end. When averaging the depth over the 100 previously fitted trees, one obtains a smoother curve (in black) that shows that later trees are deeper than those included in the early iterations. This holds for both models.



(a) Bonus certificates.

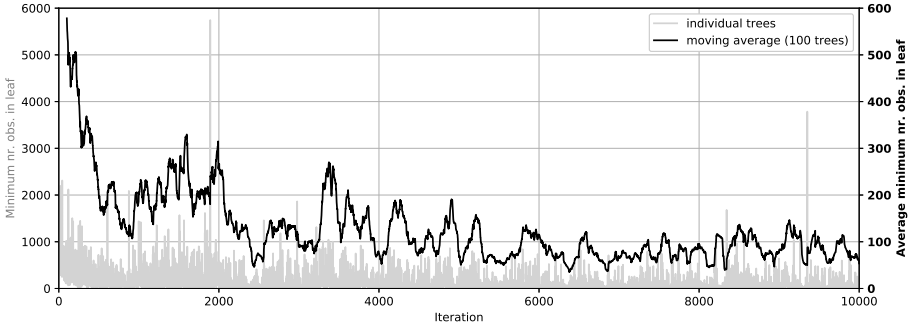


(b) American put options.

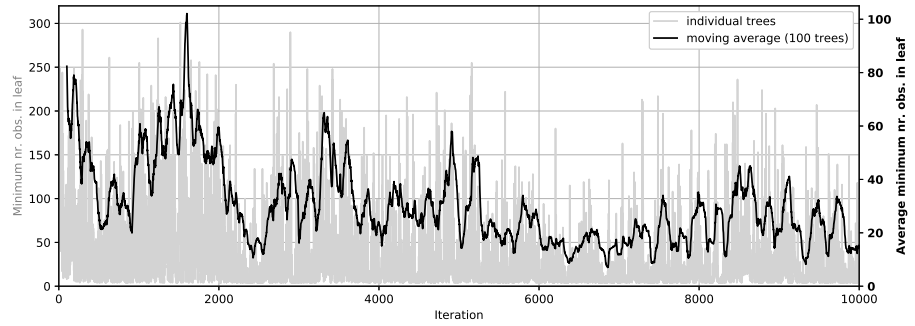
Figure 7: Tree depth for both GBM models trained on 100 000 instances.

Tree depth is linked to the number of leaves in the tree, on which we explicitly imposed upper bounds of 16 and 24 leaves for respectively bonus certificates and American put options. In the model for bonus certificates, 9976 trees have the maximum allowed 16 leaves. Most of the 33 smaller trees are fitted in the early iterations. On the other hand, each of the 10 000 trees in the model for American put options has the maximum allowed 24 leaves. Combining the information regarding tree depth and number of leaves, we can conclude that trees in the bonus certificates model are more asymmetric, where subsequent splitting happens often one side of the tree, whereas trees in the American put model are more shallow relative to the number of leaves and are hence wider and more balanced.

In Figure 8, we present the minimum number of observations per leaf for each tree in the ensemble, i.e. we only consider the leaf that contains the smallest number of training instances. We observe that this number is often higher than the required lower bounds of 24 and 4 for bonus certificates and American put options, respectively. When considering the moving average over the 100 previously fitted trees, we detect for both models a decreasing trend in the minimum number of observation per leaf node. This shows that later fitted trees need to separate smaller subsets of the training data to find relevant tree splits.



(a) Bonus certificates.

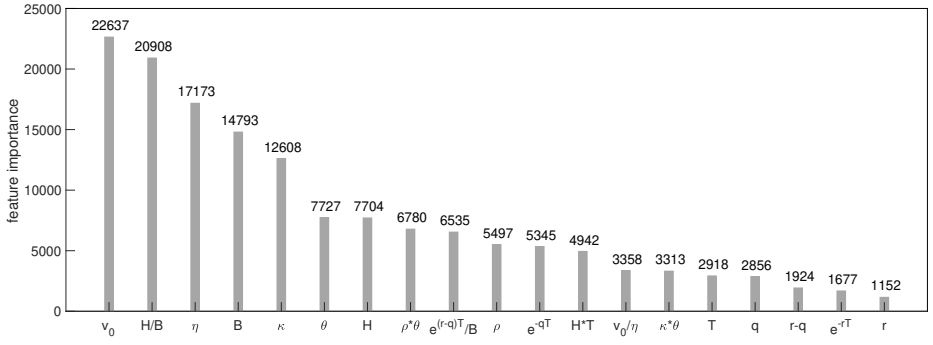


(b) American put options.

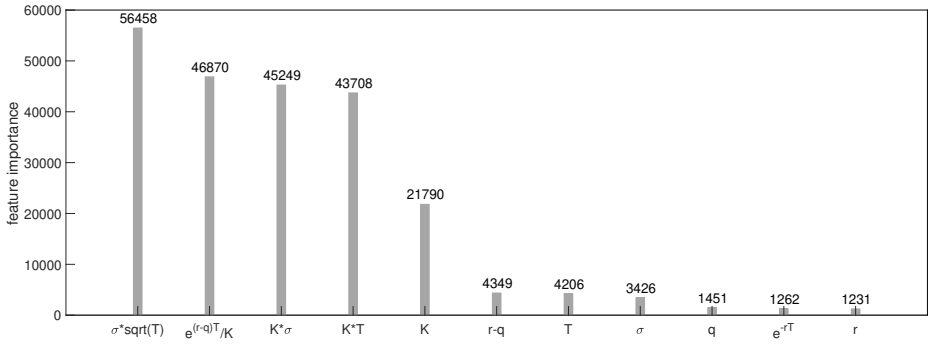
Figure 8: Minimum number of observations per leaf in each tree for both GBM models trained on 100 000 instances.

4.2 Feature importance

Feature importance plots reveal how much importance a model attaches to each input variable in the data. A simple way to measure feature importance in tree-based models consists in counting the number of times a variable is used in the model, i.e. how many times a regression tree in the ensemble makes a split on this variable. Figure 9 presents the split feature importances for the GBM models for bonus certificates and American put options, both trained on 100 000 observations. As stated in Table 3 and 6, both models consist of 10 000 regression trees, respectively with maximum 16 and 24 leaves. The aggregate feature importance is therefore bounded by $(16 - 1) \times 10\,000 = 150\,000$



(a) Bonus certificates.

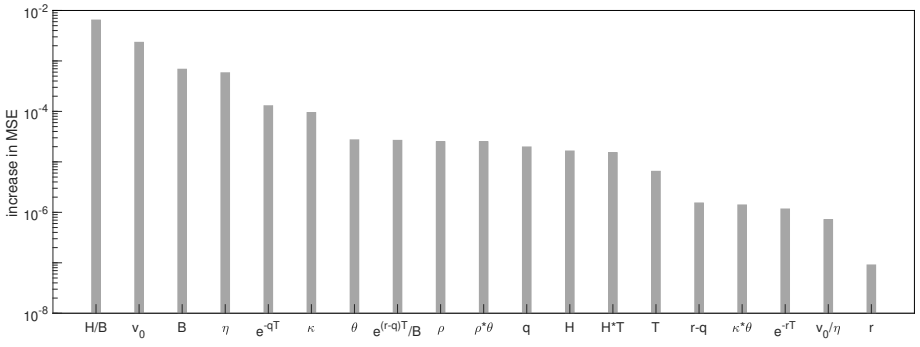


(b) American put options.

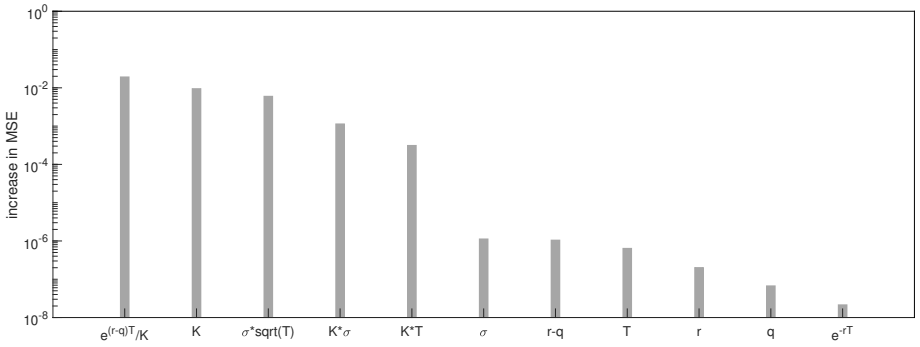
Figure 9: Feature importance plots for the models trained on 100 000 instances, measured by the number of times each feature is used in the model.

splits for the first model and 230 000 for the latter. Figure 9a shows that the initial variance v_0, the ratio of knock-out and barrier level H/B, the long run variance η, the bonus level B and the speed of mean reversion κ are the five variables on which the model relies most to make its predictions. In particular, 88 119 (of the 149 847) splits are based on one of these five variables. Similarly, Figure 9b shows that the time-scaled volatility $\sigma\sqrt{T}$, the moneyness $e^{(r-q)T}/K$, the strike K and the interactions $K\sigma$ and KT are the most important drivers for predicting prices of American put options. These variables account for 214 075 of the 230 000 splits in the model. Note that four of these five variables are constructed by feature engineering.

Alternatively, the important features can be identified by directly looking at the feature's impact on the predictions made by the model, i.e by computing the increase in prediction error when this feature is not available. Permutation feature importance, introduced by [5], makes a feature irrelevant by permuting its observed values. A feature is important if the permutation increases the error, as in this case the model relies on the feature for making predictions. Figure 10 shows the permutation feature importances, measured by the increase in out-of-sample mean squared error, for the GBM models trained on 100 000 instances. Although the importance plots for bonus certificates in Figure 9a and Figure 10a do not result in exactly the same order, they largely agree on the most important variables. For American put options, we observe that the five most important variables in Figure 10b agree with those in Figure 9b. Moreover, they again have a significantly higher importance value than the remaining input variables.



(a) Bonus certificates.



(b) American put options.

Figure 10: Out-of-sample permutation feature importance plots for the GBM models trained on 100 000 instances.

4.3 Feature impact and prediction smoothness

Partial dependence plots [16] and individual conditional expectation curves [18] are graphical tools for gaining insight in the effect of a subset of input variables on the prediction of a (black box) model. A partial dependence function indicates how the average predicted value changes when a subset of features $\boldsymbol{x_S}$ varies, i.e.

$$f_S^{PD}(\boldsymbol{x_S}) = \frac{1}{n}\sum_{i=1}^{n} F_K(\boldsymbol{x_S}, \boldsymbol{x_i^*}) \tag{21}$$

where F_K is the model, n the number of training instances and $\boldsymbol{x_i^*}$ the ith observation in the training set excluding the values of $\boldsymbol{x_S}$. A partial dependence plot is built by varying the values of $\boldsymbol{x_S}$ over their domain. Since this results in an average curve, the plot could be misleading when input variables interact with each other. Individual conditional expectation (ICE) plots try to overcome this issue by disaggregating the average in Equation (21). For the ith observation in the training set, the ICE function is given by

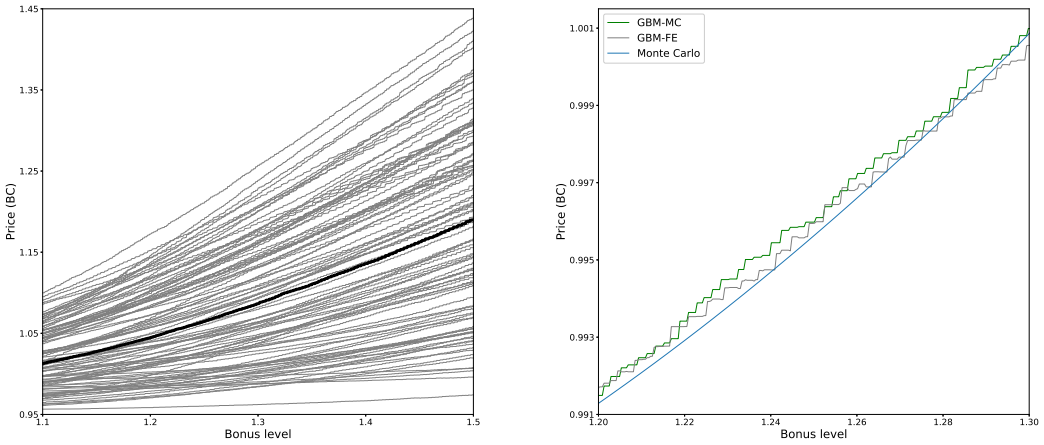
$$f_{i,S}^{ICE}(\boldsymbol{x_S}) = F_K(\boldsymbol{x_S}, \boldsymbol{x_i^*}). \tag{22}$$

Averaging all n ICE curves produces again the partial dependence plot. Figure 11a displays in black the partial dependence plot regarding the bonus level B in the GBM

model for bonus certificates. In fact, we consider the subset

$$\boldsymbol{x_S} = \left(B,\ \frac{H}{B},\ \frac{e^{(r-q)T}}{B} \right), \tag{23}$$

as the prediction curves have no meaning when these input variables are not linked. We let B vary over the interval $[1.1, 1.5]$ and adapt the other two features using the values of H, r, q and T in the training set. Figure 11a shows that the average impact on the price is increasing with B. In grey, we added the ICE curves for 100 randomly selected training observations. All these curves seem to be increasing as well. A monotonic relation was indeed expected, since a higher bonus level B should result in a higher payoff. The grey



(a) ICE and partial dependence curves.

(b) ICE curves in detail.

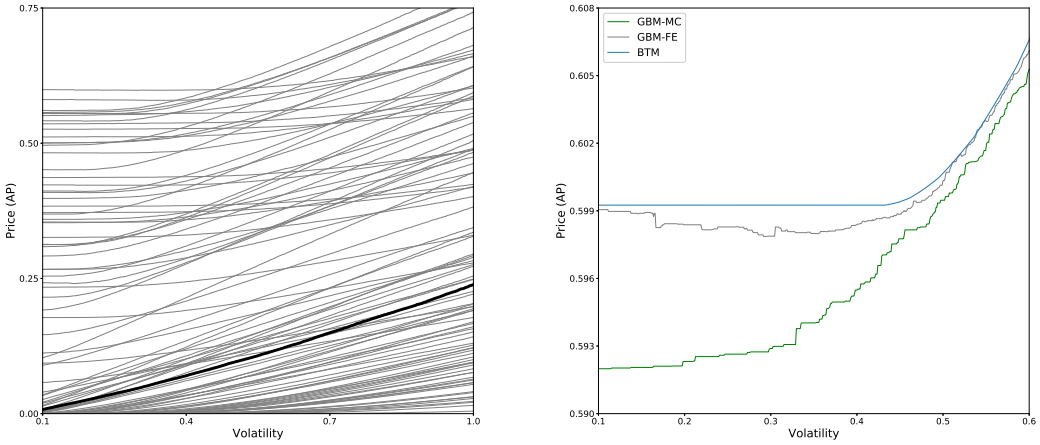
Figure 11: ICE curves (grey) and partial dependence plot (black) for the GBM model including feature engineering for predicting prices of bonus certificates. The ICE curve in green originates from the model with monotonic constraints.

line in Figure 11b shows one of the ICE curves in detail. The benchmark price curve computed by Monte Carlo simulations is added in blue to give an idea about the accuracy of the predictions. It turns out that the expected monotonic relation is not consistently satisfied. This can be tackled by imposing a marginal positive monotonic constraint for B, which enforces the price to increase with B. Indeed, the green line in Figure 11b shows the ICE curve for the same training instance, but drawn from the GBM model with monotonic constraints. The monotonic relation is satisfied, however, the price to be paid for this consistency is a loss of accuracy as demonstrated in Table 4. The stepwise curves in Figure 11b moreover affirm the typical rectangular structure of tree-based models. Since observations are assigned to a finite number of leaf nodes, prediction curves such as those in Figure 11 are piecewise constant functions of the input variables. As a consequence, one cannot directly use the pricing models for calculating price sensitivities, e.g. by finite-differencing. An alternative consists in building a separate GBM model for the sensitivities themselves. For instance, one can train a model on the set $(X, \boldsymbol{y})$, where X contains the same input parameters as the pricing GBM, while $\boldsymbol{y}$ contains the Delta of the corresponding option values, computed with a benchmark method.

Figure 12 presents the partial dependence plot and 100 randomly selected ICE curves for the volatility (σ) in the GBM model for American put options. In particular, the subset $\boldsymbol{x_S}$ equals

$$\boldsymbol{x_S} = \left(\sigma,\ \sigma\sqrt{T},\ K\sigma\right), \tag{24}$$

where σ varies from 10% to 100% and both T and K are extracted from the training set. An increasing relation can be seen for all curves in Figure 12a. Figure 12b shows



(a) ICE and partial dependence curves.

(b) ICE curves in detail.

Figure 12: ICE curves (grey) and partial dependence plot (black) for the GBM model including feature engineering predicting prices of American put options. The ICE curve in green originates from the model with monotonic constraints.

one arbitrary ICE curve in detail, which illustrates that the expected monotonic relation for σ is again violated. The reference price curve computed by the binomial tree model is depicted in blue. The predicted price decreases when σ goes from 10% to 40% and is slightly too low in this volatility range. The green line in Figure 12b shows that enforcing monotonic constraints solves the consistency issue. However, the prediction error has become significantly larger, as stated in Table 7.

5 Why gradient boosting?

After discussing the predictive, computational and behavioral characteristics of gradient boosting models for option pricing, the question remains how these models compare to other machine learning models proposed in the literature. In this section, we highlight some advantages and disadvantages of GBMs relative to other models, where we consider Gaussian process regression and neural networks as the main competitors.

5.1 Training and hyperparameter optimization

State-of-the-art boosting frameworks like LightGBM, BitBoost and XGBoost make it possible to train a gradient boosting model in a very efficient way. Indeed, the models in this paper were all trained in less than 15 minutes. As Figure 1a already demonstrated,

the training time depends on the number of instances in the training set, but increases at a fairly slow rate. Note that the training time also depends on the hyperparameter setting, e.g. the more trees to be fit, the longer the training time. The computation time of hyperparameter tuning was not included in Figure 1a, as this highly depends on the chosen parameter search space.

Most implementations of Gaussian process regression models have a training time complexity of $\mathcal{O}(n^3)$, n being the number of training instances, due to the inversion of an $n \times n$ matrix that is naively addressed via a Cholesky decomposition. Moreover, the memory requirement scales as $\mathcal{O}(n^2)$ since the matrix is stored in memory. For this reason, Gaussian process regression models in the financial literature have been trained on rather small data sets of maximum 20 000 observations. However, recent research (e.g. [36]) has extended the basic Gaussian process regression framework by partitioning and distributing kernel matrix multiplications, resulting in a reduced memory requirement of $\mathcal{O}(n)$. On the other hand, Gaussian process regression models have few hyperparameters that can elegantly be optimized via a maximum likelihood estimation.

Training a neural network is considered to be a computationally demanding task as well, and training complexity highly depends on the network architecture. Indeed, neural networks have numerous hyperparameters that need to be tuned carefully. Many of them are very similar to the GBM parameters, e.g. learning rate, dropout rate, batch size, etc. These parameters can be tuned relatively easy, in a similar fashion as explained in this paper. In a neural network, however, one additionally has to decide on the optimal network structure, i.e., number of layers, number of neurons in a layer, activation function etc., which is more complicated due to the complexity of the search space. The financial machine learning literature moreover indicates that neural networks require large training sets in order to obtain the desired accuracy. For instance, [27] use a training set of 1 million observations for pricing European call options and [14] rely on 5 to 500 million training observations for pricing basket options.

Although gradient boosting models require a hyperparameter optimization as well, the ease of training - in particular for large training sets - is one of the advantages of GBMs.

5.2 Predictive performance

Predictive performance of the machine learning models can be compared in different ways. Not only computation time, but also prediction accuracy and interpretation are important.

5.2.1 Efficiency

Once trained, a gradient boosting model can make predictions by evaluating and aggregating the fitted trees. This procedure is very fast, and its computation time does moreover not depend on the size of the training set. The same holds for neural networks, as prediction with these models - roughly speaking - boils down to evaluating a composition of nonlinear functions of linear combinations of the inputs. For the standard Gaussian process regression implementation, prediction time scales with $\mathcal{O}(n)$, where n is the number of training instances. Gaussian processes are hence not only computationally expensive in the offline training phase, but also when making real-time predictions.

5.2.2 Accuracy

Equally interesting is how the models compare in terms of accuracy of predicted prices. Unfortunately, this can only be addressed when the models are trained on exactly the same data sets, i.e. for the same derivative products, pricing models, methods and parameter ranges. Hence we cannot rely on the results in the literature for a trustworthy comparison. Following [10], we train a Gaussian process regression model on the smallest data set of 10 000 observations[7], both for bonus certificates (BC) and American put options (AP). The results are stated in Table 9.

	BC	**AP**	
Training set	$(\boldsymbol{X}, \boldsymbol{y})$	$(\boldsymbol{X}, \boldsymbol{y})$	$(\boldsymbol{X}, \log(\boldsymbol{y}))$
In-sample prediction			
MAE	8.90e-03	7.42e-03	8.59e-03
AAE	3.73e-04	3.03e-04	5.45e-04
MRPE	7.47e-03	25.43	2.82e-01
ARPE	3.38e-04	7.13e-02	3.77e-03
Out-of-sample prediction			
MAE	4.93e-03	6.65e-03	1.23e-02
AAE	3.52e-04	2.59e-04	5.75e-04
MRPE	4.47e-03	43.17	1.93e-01
ARPE	3.19e-04	7.94e-02	4.25e-03
Computation time			
Prediction time (s)	78.72	14.62	14.17
Speed-up	$\times$ 3136	$\times$ 271	$\times$ 280

Table 9: Performance of Gaussian process regression for both applications in this study. All models are trained on 10 000 observations and out-of-sample performance is measured on a test set of 100 000 observations.

For bonus certificates, we find that out-of-sample prediction errors of Gaussian process regression models are of the same order as those of the GBM models trained on 100 000 instances in Table 4. Taking into account the computation time, this means that GBM models predict prices with a similar accuracy in only half the time. When comparing Gaussian process regression models to GBM models trained on the data set of only 10 000 observations, we observe that the former provide an out-of-sample AAE that is about 7 basis points smaller than the AAE of GBM predictions. These smaller errors of Gaussian process regression for pricing bonus certificates can probably be attributed to the noise in the training data, which is explicitly modeled by Gaussian processes. Indeed, when inspecting their performance on pricing American put options, we find that GBM models trained on only 10 000 observations, both in Table 7 and Table 11, have a comparable to slightly better accuracy than the Gaussian process regression models in Table 9. Similarly as for GBM models, a Gaussian process regression model trained on the untransformed prices $\boldsymbol{y}$ results in very large relative errors - even larger than for GBM model - and training on log-transformed prices is preferred. Whereas a GBM model can easily be trained on a larger training set to improve predictive performance, this is not trivial for Gaussian process regression models. Hence, GBM models can

[7]The models have a squared exponential kernel function and a zero mean function. For bonus certificates, we apply the kernel variation with automatic relevance determination, i.e. with a characteristic length-scale parameter for each input dimension. Hyperparameters are found by maximizing the (log) marginal likelihood of the training observations.

achieve an average absolute error of approximately 2 basis points, while the AAE for a Gaussian process regression model equals almost 6 basis points.

Comparing these results with the predictive performance of neural networks would be interesting as well, but this requires an extensive hyperparameter tuning and would lead us too far. A thorough comparative analysis is left for future research.

5.2.3 Interpretation

Individual price predictions of gradient boosting models might be fairly accurate, however, the functional form of the resulting price approximator can be a concern for the interpretation of predicted prices. As already pointed out in Section 4.3, predicted price curves are piecewise constant and prices are insensitive to very small changes in the product parameters. While this complicates the computation of Greeks, it might not play a role when only price estimates are required or when conclusions have to be made on the portfolio level. On the other hand, both Gaussian process regression and neural networks provide smooth prediction curves and might be more appropriate when price sensitivities are required. Monotonic behavior of the resulting model is important as well. In most recent GBM implementations, the prediction function can be enforced to satisfy monotonic constraints for single input parameters. For Gaussian process regression models, one can impose shape-constraints by approximating Gaussian processes as proposed in [28] and illustrated for financial term structures in [7]. Constraining neural networks is possible as well, see for instance [37] for a recent method and an overview of the related literature. However, as we did not run these methods, the impact of monotonic constraints on the accuracy of predicted prices is not clear and might be restricting the predictive and computational performance of the models.

6 Conclusion

In this paper, we demonstrated how tree-based machine learning methods are employed for pricing derivative products. We showed that gradient boosted decision tree models manage to speed up the pricing procedure of (exotic) derivative products with several orders of magnitude. In particular, we illustrated the approach for exotic and American options. Building a gradient boosted machine boils down to two steps. First, the hyperparameters need to be tuned. This introduces a lot of model flexibility which allows for a good fit. Secondly, the final model has to be trained according to the optimal hyperparameter setting. Once trained, the model is able to compute derivative prices according to any new market situation, as long as the parameters still belong to the initial ranges of the training set. Since gradient boosted models can be evaluated extremely fast, they allow for real-time derivative pricing. In Section 4, we discussed how predictions of tree-based models can be understood better. An analysis of the individual trees in the ensemble showed that trees tend to grow over time (in the sense of tree depth) and that the minimum number of observations in each leaf is often higher than the required lower bound. Feature importance plots showed on which parameters the models rely most, while partial dependence plots and individual conditional expectation curves illustrated how the predicted price moves when the product parameters are slightly shifted. These graphs revealed that monotonic behavior of the price with respect to certain input parameters was not always satisfied. We showed that this issue can be solved by explicitly imposing monotonic constraints on the model. Unfortunately, this

may reduce the accuracy of price predictions. In the last part of the paper, we briefly compared the main properties of gradient boosting models to the existing alternatives proposed in the literature, i.e. Gaussian process regression and neural networks. Gradient boosting models are preferred for their ability of training on large data sets with limited computational budget, while their main drawback is the piecewise constant structure of prediction curves. Finally, we want to point out that the same methodology can be applied to other (parametric) pricing models and products, as long as a thorough error analysis is carried out for each specific application.

Acknowledgements JD is supported by the KU Leuven Research Fund (C14/17/070). LD is supported by the KU Leuven Research Fund (C14/17/070) and by Research Foundation-Flanders (FWO).

References

[1] Auguste, C., Smirnov, I., Malory, S., and Grassl, T. (2019) A better method to enforce monotonic constraints in regression and classification trees. Tech. rep., available on GitHub: `https://github.com/microsoft/LightGBM/files/3457826/PR-monotone-constraints-report.pdf`.

[2] Bergstra, J. and Bengio, Y. (2012) Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, **13**, 281–305.

[3] Biau, G. and Cadre, B. (2017) Optimization by gradient boosting. ArXiv: 1707.05023.

[4] Blanchard, G., Lugosi, G., and Vayatis, N. (2003) On the rate of convergence of regularized boosting classifiers. *Journal of Machine Learning Research*, pp. 861– –894.

[5] Breiman, L. (2001) Random forests. *Machine Learning*, **45**, 5–32.

[6] Chen, T. and Guestrin, C. (2016) XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 785–794, ACM.

[7] Cousin, A., Maatouk, H., and Rullière, D. (2016) Kriging of financial term-structures. *European Journal of Operational Research*, **255**, 631–648.

[8] Cox, J. C., Ross, S. A., and Rubinstein, M. (1979) Option pricing: a simplified approach. *Journal of Financial Economics*, **7**, 229–263.

[9] Crépey, S. and Dixon, M. F. (2020) Gaussian process regression for derivative portfolio modeling and application to credit valuation adjustment computations. *Journal of Computational Finance*, **24**, 47–81.

[10] De Spiegeleer, J., Madan, D. B., Reyners, S., and Schoutens, W. (2018) Machine learning for quantitative finance: fast derivative pricing, hedging and fitting. *Quantitative Finance*, **18**.

[11] Decroos, T., Bransen, L., Van Haaren, J., and Davis, J. (2019) Actions speak louder than goals: Valuing player actions in soccer. *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1851—-1861, KDD '19, Association for Computing Machinery, New York, NY, USA.

[12] Devos, L., Meert, W., and Davis, J. (2019) Fast gradient boosting decision trees with bit-level data structures. *Proceedings of ECML PKDD*, Springer.

[13] Duan, T., Avati, A., Yi Ding, D., Thai, K., Basu, S., Ng, A., and Schuler, A. (2020) NGBoost: Natural gradient boosting for probabilistic prediction. *Proceedings of the 37th International Conference on Machine Learning*, PMLR 108, Vienna, Austria.

[14] Ferguson, R. and Green, A. (2018) Deeply learning derivatives. ArXiv: 1809.02233.

[15] Feurer, M., Eggensperger, K., Falkner, S., Lindauer, M., and Hutter, F. (2018) Practical automated machine learning for the AutoML Challenge 2018. *ICML 2018 AutoML Workshop*.

[16] Friedman, J. H. (2001) Greedy function approximation: a gradient boosting machine. *The Annals of Statistics*, **29**, 1189–1232.

[17] Friedman, J. H. (2002) Stochastic gradient boosting. *Computational Statistics & Data Analysis*, **38**, 367–378.

[18] Goldstein, A., Kapelner, A., Bleich, J., and Pitkin, E. (2015) Peeking inside the black box: visualizing statistical learning with plots of individual conditional expectation. *Journal of Computational and Graphical Statistics*, **24**, 44–65.

[19] Hastie, T., Tibshirani, R., and Friedman, J. (2009) *The Elements of Statistical Learning*. Springer, 2nd edn.

[20] Henckaerts, R., Côté, M.-P., Antonio, K., and Verbelen, R. (2020) Boosting insights in insurance tariff plans with tree-based machine learning methods. *North American Actuarial Journal*, **0**, 1–31.

[21] Hernandez, A. (2017) Model calibration with neural networks. *Risk*.

[22] Heston, S.L. (1993) A closed form solution for options with stochastic volatility with applications to bonds and currency options. *Review of Financial Studies*, **6**, 327–343.

[23] Hinton, G., Srivastava, N., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2012) Improving neural networks by preventing co-adaptation of feature detectors. ArXiv: 1207.0580.

[24] Horvath, B., Muguruza, A., and Tomas, M. (2021) Deep learning volatility: a deep neural network perspective on pricing and calibration in (rough) volatility models. *Quantitative Finance*, **21**, 11–27.

[25] Hutchinson, J. M., Lo, A. W., and Poggio, T. (1994) A nonparametric approach to pricing and hedging derivative securities via learning networks. *The Journal of Finance*, **49**, 851–889.

[26] Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., and Liu, T. (2017) LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems 30*, pp. 3146–3154.

[27] Liu, S., Oosterlee, C. W., and Bohte, S. M. (2019) Pricing options and computing implied volatilities using neural networks. *Risks*, **7**.

[28] Maatouk, H. and Bay, X. (2017) Gaussian process emulators for computer experiments with inequality constraints. *Mathematical Geosciences*, **49**, 557—582.

[29] Mason, L., Baxter, J., Bartlett, P. L., and Frean, M. R. (2000) Boosting algorithms as gradient descent. *Advances in Neural Information Processing Systems 12*, pp. 512–518.

[30] Mockus, J., Tiesis, V., and Zilinskas, A. (1978) The application of Bayesian methods for seeking the extremum. *Towards Global Optimization*, **2**, 117–129.

[31] Mohan, A., Chen, Z., and Weinberger, K. (2011) Web-search ranking with initialized gradient boosted regression trees. *JMLR Workshop and Conference Proceedings: Proceedings of the Yahoo! Learning to Rank Challenge*, **14**, 77–89.

[32] Ruf, J. and Wang, W. (2020) Neural networks for option pricing and hedging: a literature review. *Journal of Computational Finance*, **24**, 1–46.

[33] Snoek, J., Larochelle, H., and Adams, R. P. (2012) Practical Bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems 25*, pp. 2960–2968.

[34] Sousa, J. B., Esquível, M. L., and Gaspar, R. M. (2012) Machine learning Vasicek model calibration with Gaussian processes. *Communications in Statistics - Simulation and Computation*, **41**, 776–786.

[35] Vinayak, R. K. and Gilad-Bachrach, R. (2015) DART: Dropouts meet multiple additive regression trees. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, San Diego, CA, USA.

[36] Wang, K., Pleiss, G., Gardner, J., Tyree, S., Weinberger, K. Q., and Wilson, A. G. (2019) Exact Gaussian processes on a million data points. *Advances in Neural Information Processing Systems 32*, pp. 14648–14659.

[37] Wehenkel, A. and Louppe, G. (2019) Unconstrained monotonic neural networks. *Advances in Neural Information Processing Systems 32*, pp. 1545–1555.

[38] Zhang, T. and Yu, B. (2005) Boosting with early stopping: Convergence and consistency. *The Annals of Statistics*, **33**, 1538–1579.

A Pricing American options without log-transformation

In Section 3.2, a GBM model was built for pricing American put options by training on the logarithm of the option prices. Alternatively, the raw prices could be considered as model outputs. Table 10 and Table 11 below show the results of respectively the hyperparameter optimization and the corresponding model performance, if the untransformed prices are used as direct model output. In absolute terms, the pricing performance seems better than the results in Table 4, but the relative errors blow up. Some of the raw predictions made by the GBM model were negative. For the calculation of the accuracy results below, these negative prices are set to zero.

	$(\boldsymbol{X}, \boldsymbol{y})$	**FE**	**MC**
number of trees (K)	10000	10000	4000
shrinkage parameter (ν)	0.15	0.1	0.2
maximum number of leaves	8	16	8
minimum number of instances per leaf	16	4	4
L_2-regularization (λ)	1.6	1.6	0.8
bagging fraction	0.5	0.3	0.7
maximum number of histogram bins	2047	2047	2047

Table 10: Optimal hyperparameter setting for pricing American put options in three settings: training on the original training set $(X, \boldsymbol{y})$ without monotonic constraints, including feature engineering (FE) or including monotonic constraints (MC).

	$(\boldsymbol{X}, \boldsymbol{y})$			**with feature engineering**			**with monotonic constraints**		
Training set size	10 000	100 000	1 000 000	10 000	100 000	1 000 000	10 000	100 000	1 000 000
In-sample prediction									
MAE	1.34e-03	2.48e-03	3.61e-03	1.17e-03	1.62e-03	2.92e-03	2.32e-02	2.49e-02	2.51e-02
AAE	2.28e-04	3.09e-04	3.35e-04	1.12e-04	1.60e-04	1.72e-04	1.81e-03	2.02e-03	1.97e-03
MRPE	1.58	3.67	4.98	1.00	1.41	3.68	3.00	5.41	13.65
ARPE	2.55e-02	3.04e-02	3.22e-02	1.26e-02	1.39e-02	1.36e-02	1.03e-01	1.12e-01	1.13e-01
Out-of-sample prediction									
MAE	4.62e-03	2.55e-03	2.35e-03	4.57e-03	1.89e-03	1.89e-03	2.13e-02	1.61e-02	1.48e-02
AAE	5.44e-04	3.49e-04	3.28e-04	3.46e-04	1.87e-04	1.68e-04	2.01e-03	1.79e-03	1.66e-03
MRPE	7.59	5.41	2.37	4.76	1.50	2.24	4.06	3.52	3.02
ARPE	3.56e-02	2.70e-02	2.62e-02	1.73e-02	1.19e-02	1.08e-02	9.73e-02	9.54e-02	9.20e-02
Computation time									
Prediction time (s)	31.52	31.07	31.34	45.07	45.06	44.57	8.29	8.29	7.58
Speed-up	× 126	× 128	× 127	× 88	× 88	× 89	× 478	× 478	× 523

Table 11: GBM performance for American put options with respect to the binomial tree method. Out-of-sample predictions and their computation time correspond to a test set of 100 000 observations.