Gradient Boosting for Quantitative Finance

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

October 2, 2019

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. Besides the predictive performance of machine learning methods, financial regulators attach more and more importance to the interpretability of pricing models. For both applications, we therefore look under the hood of the gradient boosting model and try to reveal 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 efficient 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,

*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.

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 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 the literature by a couple of authors. [5] train Gaussian process regression models for pricing vanilla and exotic derivatives with models beyond Black-Scholes. [18] deploy this model for a direct calibration of interest rate models, whereas [4] use Gaussian processes in the context of XVA computations. Many other models rely on artificial neural networks. [15] and [7] train neural nets to learn prices of respectively vanilla options and basket options in the Black-Scholes model, while [17] and [14] employ them for pricing vanilla options according to more advanced models. [11] on the other hand directly trains a neural network to return the calibrated model parameters.

To the best of our knowledge, the use of tree-based machine learning methods has not been investigated deeply in the literature. 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 try to open the black box machine learning models. Section 5 concludes the paper.

2 Gradient boosting machines

The concept of a gradient boosting machine (GBM) was first introduced by Friedman in 2001 [8], and has since become a state-of-the-art machine learning technique with a number of competitive implementations, e.g. XGBoost [2], LightGBM [16] and BitBoost [6]. We present the main ideas of the method in this section.

Consider a (training) set of observations $(X, \boldsymbol{y}) = \{(\boldsymbol{x_i}, y_i) \mid i = 1, \dots, n\}$, 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, i.e., a parametric function of a particular family, and ν is a shrinkage parameter or learning rate. This paper uses decision trees as base models.

The boosting procedure constructs each decision tree $h_k(\boldsymbol{x})$ in a stage-wise manner. 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

trees unchanged,

$$F_0(\boldsymbol{x}) = c \tag{2}$$
$$F_k(\boldsymbol{x}) = F_{k-1}(\boldsymbol{x}) + \nu\, h_k(\boldsymbol{x}). \tag{3}$$

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}, i.e.,

$$h_k = \arg\min_h \sum_{i=1}^{n} \mathcal{L}(y_i, F_{k-1}(\boldsymbol{x_i}) + h(\boldsymbol{x_i})). \tag{4}$$

To make this optimization problem feasible in practice, the key idea of Friedman [8] was to let the trees predict the negative gradient values,

$$g_{i,k} = -\left[\frac{\partial \mathcal{L}(y_i, \hat{y})}{\partial \hat{y}}\right]_{\hat{y}=F_{k-1}(\boldsymbol{x_i})}. \tag{5}$$

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

$$h_k = \arg\min_h \sum_{i=1}^{n} (g_{i,k} - h(\boldsymbol{x}_i))^2. \tag{6}$$

The procedure shown in (5) is very similar to the well-known steepest descent method. Equation (6) constructs the tree that is most correlated with $\boldsymbol{g_i}$. This can be interpreted as the steepest descent direction in an n dimensional data space evaluated at $F_{k-1}(X)$.

To further improve the predictive performance of GBMs, a technique called DART [19] 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 [13] – 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%.

We briefly discuss the most important features and their parameters of GBMs:

- *Histogram-based splitting*: 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* [9]: 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 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 in order to penalize trees with high leaf scores. Larger λ-values correspond to stronger penalties. As an additional regularization technique, one can impose a lower bound on the number of observations in each leaf node of a tree.

3 Boosting derivative pricing

When pricing complex derivative products according to models beyond the Black-Scholes model, typically no analytic pricing formula exists. One often has to resort to time-consuming numerical methods. Marking to market a portfolio of such products hence becomes computationally demanding. 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. This allows for an extensive hyperparameter tuning and hence for a more flexible model. The tuning follows 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 hyperparameters are tuned by performing a grid search on the parameter search space[1] in Table 1, which requires

[1]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 error for the validation set.

fitting 2304 different GBM models. 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. For the grid search we rely on training and validation sets each consisting of 10 000 instances.

number of trees (K)	4000, 5000, 6000, 10000
shrinkage parameter (ν)	0.1, 0.15, 0.2, 0.25
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 the two applications in this study.

Once the model is trained with the tuned hyperparameters, 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,

$$\text{MAE} = \max\left\{|y_{true}(i) - y_{pred}(i)|,\ i = 1, \ldots, n\right\}, \tag{7}$$

$$\text{AAE} = \frac{1}{n}\sum_{i=1}^{n} |y_{true}(i) - y_{pred}(i)|, \tag{8}$$

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

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

where y_{true} and y_{pred} refer respectively to the benchmark price and the predicted price. 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.

3.1 Pricing exotic derivatives

We illustrate 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 payoffs of a bonus certificate with bonus level B and knock-out level H are given by

$$\text{payoff} = \begin{cases} S_T & \text{if } \min\limits_{0 \le t \le T} S_t \le H \\ \max(B, S_T) & \text{else} \end{cases} \tag{11}$$

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, [12]. 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.[2] 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. 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.

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.

Although this training set $(X, \boldsymbol{y})$ could perfectly be used to train a GBM model, we first extend the input matrix X by means of feature engineering. Adding new input parameters, which are transformations of the original inputs, allows us to incorporate domain knowledge about the relation to be learned. For each observation

$$\boldsymbol{x_i} = (B_i,\ H_i,\ T_i,\ r_i,\ q_i,\ \kappa_i,\ \rho_i,\ \theta_i,\ \eta_i,\ v_{0i}), \qquad i = 1, \ldots, n \tag{12}$$

we compute the values of

$$\frac{H_i}{B_i}, \quad H_i T_i, \quad e^{-r_i T_i}, \quad e^{-q_i T_i}, \quad r_i - q_i, \quad \frac{e^{(r_i - q_i)T_i}}{B_i}, \quad \rho_i \theta_i, \quad \kappa_i \theta_i \quad \text{and} \quad \frac{v_{0i}}{\eta_i} \tag{13}$$

and add them as new input variables. If these combined inputs would not offer added value, the boosting framework will simply ignore them and rarely include split conditions on these variables. The augmented input matrix $\tilde{X}$ thus consists of 19 columns and $(\tilde{X}, \boldsymbol{y})$ serves as the final 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.

Tuning the hyperparameters of the gradient boosting model - either with the extended input matrix $\tilde{X}$ or the original matrix X - results in the optimal configuration as displayed in Table 3. Feature engineering has (in this empirical study) no impact on the optimal hyperparameter setting for pricing bonus certificates. However, note that this configuration highly depends on the chosen parameter grid in Table 1.

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

	feature engineering		no feature engineering	
	BC	AP	BC	AP
number of trees (K)	10000	10000	10000	10000
shrinkage parameter (ν)	0.2	0.1	0.2	0.15
number of leaves	16	16	16	8
minimum number of instances per leaf	24	4	24	16
L_2-regularization (λ)	0.8	1.6	0.8	1.6
bagging fraction	0.5	0.3	0.5	0.5
maximum number of histogram bins	255	2047	255	2047

Table 3: Optimal hyperparameter setting for the two applications in this study.

3.1.2 GBM's price predictions

Given the optimal hyperparameter setting, three GBM models are trained on differently sized training sets. The empirical performance of these models is summarized in Table 4. Alternatively, we trained the three models on the original training set $(X, \boldsymbol{y})$ to

	with feature engineering			without feature engineering		
Size of training set	10 000	100 000	1 000 000	10 000	100 000	1 000 000
In-sample prediction						
MAE	7.9116e-04	2.2404e-03	5.9594e-03	9.1007e-04	2.4648e-03	6.4242e-03
AAE	1.4065e-04	3.7152e-04	5.4051e-04	1.5280e-04	4.0847e-04	5.7396e-04
MRPE	6.9812e-04	1.9924e-03	5.0191e-03	7.2565e-04	2.2115e-03	5.7268e-03
ARPE	1.2778e-04	3.3433e-04	4.8044e-04	1.3897e-04	3.6789e-04	5.1210e-04
Out-of-sample prediction						
MAE	1.4618e-02	5.2133e-03	4.3289e-03	1.3652e-02	5.5312e-03	3.9807e-03
AAE	1.0610e-03	5.8812e-04	5.3294e-04	1.0706e-03	6.3131e-04	5.7261e-04
MRPE	1.1065e-02	4.1159e-03	3.2906e-03	1.1489e-02	4.4840e-03	3.5128e-03
ARPE	9.5880e-04	5.3283e-04	4.8322e-04	9.7101e-04	5.7395e-04	5.2097e-04
Computation time						
Prediction time (s)	42.14	42.18	42.09	43.38	42.58	42.38
Speed-up	× 5858	× 5852	× 5864	× 5690	× 5797	× 5825

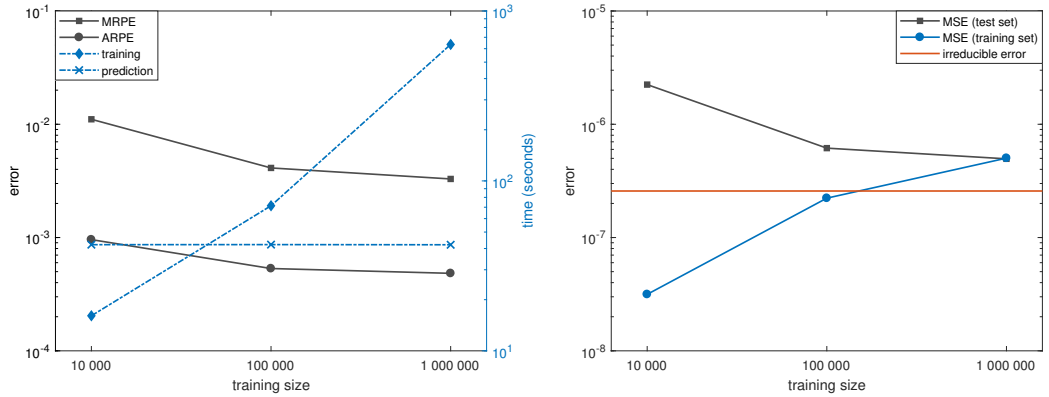
Table 4: Performance of GBM models for pricing bonus certificates in the Heston model. Out-of-sample predictions and their computation time[3] correspond to a test set of 100 000 instances.

illustrate the effect of feature engineering. In this example, feature engineering seems to have only little impact on the predictive performance of the model. 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

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

trained on 100 000 instances, the worst-case out-of-sample relative error has 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



(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.

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 sharply 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 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.

$$Var(\epsilon) \approx \frac{1}{100\ 000} \sum_{i=1}^{100\ 000} \frac{\sigma_i^2}{500\ 000} \tag{14}$$

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. 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.

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.

$$\frac{BC_{pred}(i) - BC_{true}(i)}{BC_{true}(i)}, \qquad i = 1, \ldots, 100\ 000, \tag{15}$$

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

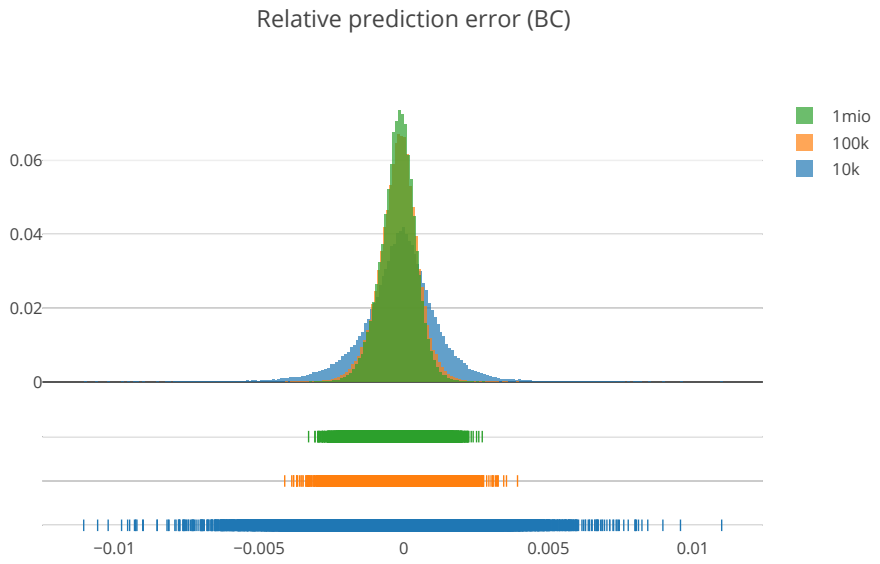
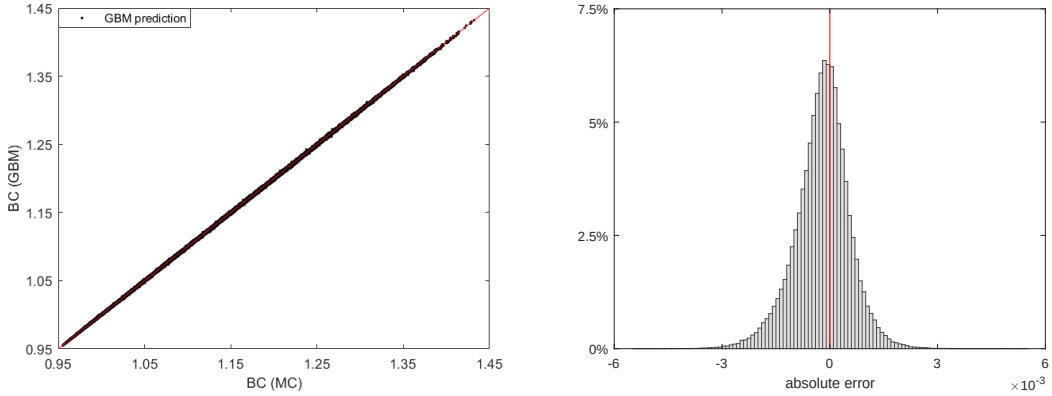


Figure 2: Out-of-sample relative errors of bonus certificates predicted by GBM models trained on respectively 10 000, 100 000 and 1 000 000 instances.

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 depend on the price level. Figure 3b displays the absolute prediction errors as defined by the numerator in Equation (15). Note that this histogram is not exactly centered around zero. In fact, 60.39% of the predicted prices are lower than their benchmark value.

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 usually 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 [3], where we discretize the lifetime of the options in time steps of 1/750.



(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.

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 (S_0) to be equal to 1.

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.

For each input observation, the price of the American put option is calculated with a binomial tree and stored in $\boldsymbol{y}$. In order to avoid too many 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. Next, we augment the input matrix X with the following transformations of the original input variables:

$$\sigma\sqrt{T}, \quad KT, \quad K\sigma, \quad e^{-rT}, \quad r-q, \quad \text{and} \quad \frac{e^{(r-q)T}}{K}. \tag{16}$$

This feature engineering procedure results in a final input matrix $\tilde{X}$ with 11 feature columns. Table 3 states the optimal hyperparameter configuration for the GBM model pricing American put options. In this case, the optimal parameter setting changes when feature engineering is incorporated.

3.2.2 GBM's price predictions

We build GBM models on both the original training set $(X, \boldsymbol{y})$ and the augmented set $(\tilde{X}, \boldsymbol{y})$, where the corresponding optimal hyperparameters are used. In both cases, three

models are trained on respectively 10 000, 100 000 and 1 000 000 training instances. Table 6 summarizes their performance[4] in terms of predictive power and computation time. Although we deleted and replaced prices below 0.0001 in all data sets, Figure 4a shows that the test set still contains a lot of small prices. More specifically, 9.52% of the test prices are below 0.01. Therefore, relative pricing errors are large and do not give a reliable summary of the predictive performance. We focus on the absolute error measures instead.

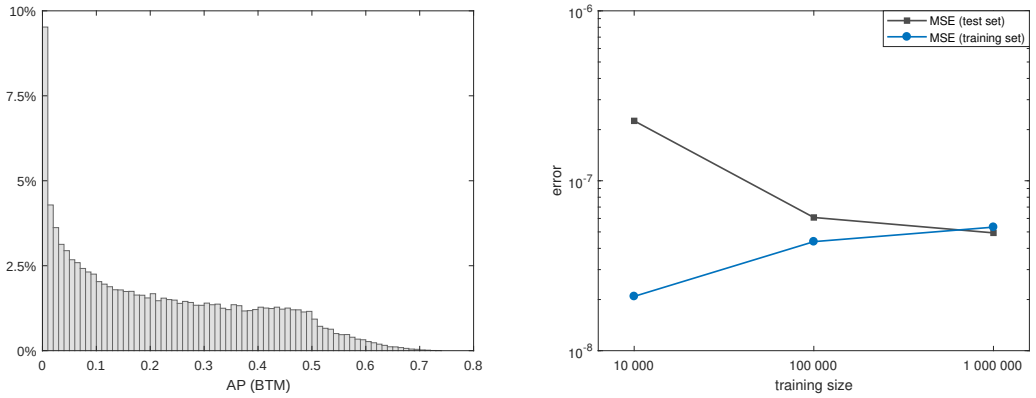
	with feature engineering			**without feature engineering**		
Size of training set	10 000	100 000	1 000 000	10 000	100 000	1 000 000
In-sample prediction						
MAE	1.1706e-03	1.6190e-03	2.9238e-03	1.3412e-03	2.4801e-03	3.6053e-03
AAE	1.1164e-04	1.5962e-04	1.7225e-04	2.2757e-04	3.0878e-04	3.3450e-04
MRPE	1.0000	1.4138	3.6790	1.5810	3.6678	4.9752
ARPE	1.2581e-02	1.3868e-02	1.3587e-02	25506e-02	3.0356e-02	3.2217e-02
Out-of-sample prediction						
MAE	4.5707e-03	1.8891e-03	1.8933e-03	4.6224e-03	2.5484e-03	2.3492e-03
AAE	3.4636e-04	1.8653e-04	1.6838e-04	5.4407e-04	3.4930e-04	3.2802e-04
MRPE	4.7584	1.5033	2.2447	7.5866	5.4077	2.3651
ARPE	1.7340e-02	1.1891e-02	1.0765e-02	3.5553e-02	2.7044e-02	2.6156e-02
Computation time						
Prediction time (s)	45.07	45.06	44.57	31.52	31.07	31.34
Speed-up	× 88	× 88	× 89	× 126	× 128	× 127

Table 6: 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 model performance. We observe that the absolute prediction errors are nearly halved due to the feature engineering procedure. On the other hand, the prediction time of these models is significantly higher. Feature engineering itself has no direct impact on the prediction time, but the resulting hyperparameter setting has. The main driver is in this case the number of leaves, which is only 8 when the model is trained on $(X, \boldsymbol{y})$, leading to reduced prediction times. Since we attach more importance to the improved accuracy, the Figures in the remainder of the discussion are based on the models built on $(\tilde{X}, \boldsymbol{y})$.

The absolute errors in Table 6 show that the worst-case prediction error is of order 10^{-3} for all models, while the average prediction error becomes smaller than two basis points when the training set is large enough. Again, we observe that the in-sample errors increase when the training set expands, while the out-of-sample predictive performance improves. The same evolution can be seen in the mean squared prediction errors in Figure 4b. Since 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 (trained on $(\tilde{X}, \boldsymbol{y})$) needs approximately 45 seconds to predict the 100 000 prices in the test set. This is roughly 90 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

[4]Some of the raw predictions made by the GBM model were negative. In all graphs and accuracy results reported, these negative prices are set to zero.



(a) Test set prices.

(b) Mean squared errors.

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

higher than the values reported in Table 6. 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

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

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

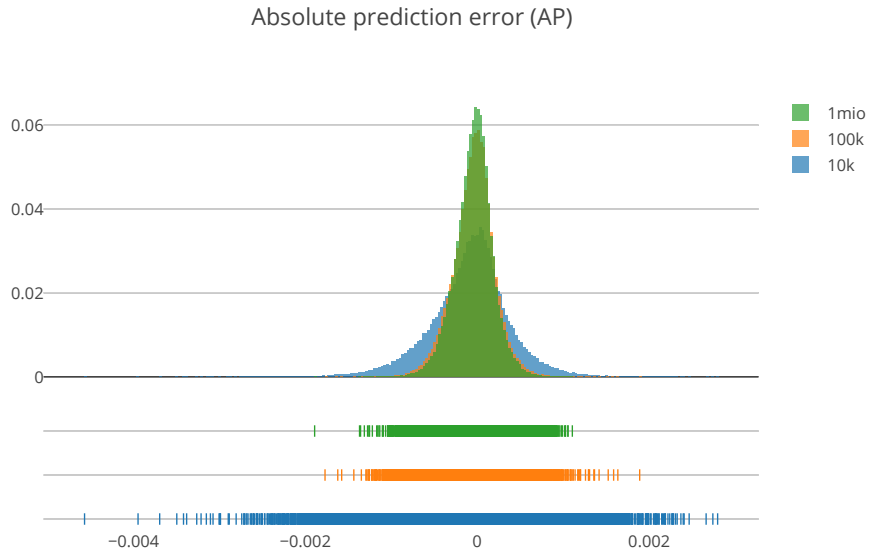
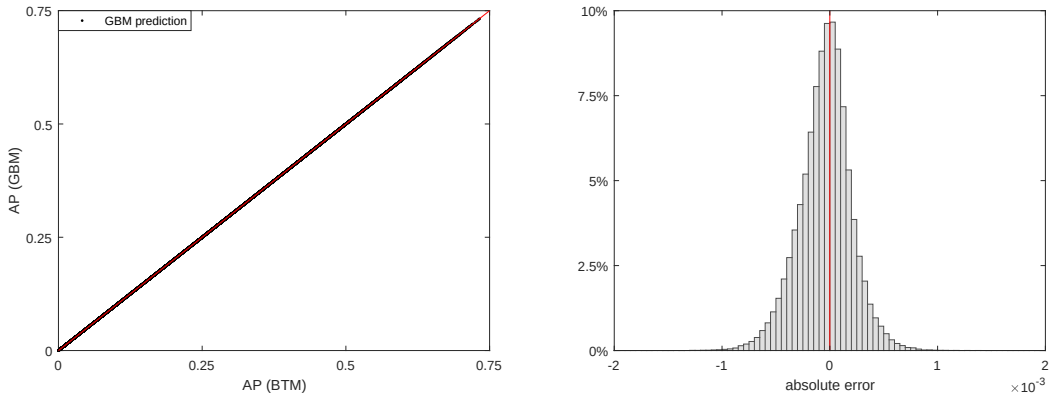


Figure 5: Out-of-sample prediction errors for GBM models trained on sets of 10 000, 100 000 and 1 000 000 instances.

in predictive performance when extending the training set from 10 000 to 100 000 observations. Training on 1 million observations improves the model only slightly. Figure

6 focuses on the predictive performance of the model trained on 100 000 observations. Figure 6a plots the predicted prices against the benchmark prices calculated with the binomial tree model. The absolute errors do not seem to depend on the price level. The prediction errors in Figure 6b, defined by Equation (17), are roughly symmetrically distributed around zero, with 55.80% of the prices being lower than their benchmark value.



(a) Out-of-sample predictions.

(b) Absolute prediction errors.

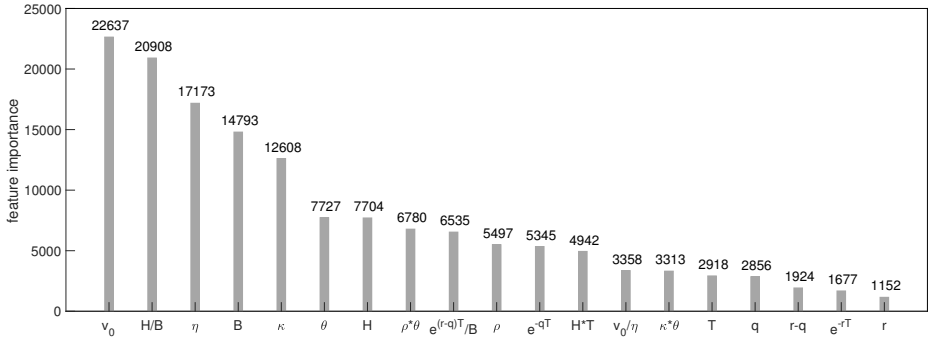
Figure 6: Out-of-sample predictive performance of the GBM model for American put options, trained on 100 000 instances.

4 Boosted predictions explained

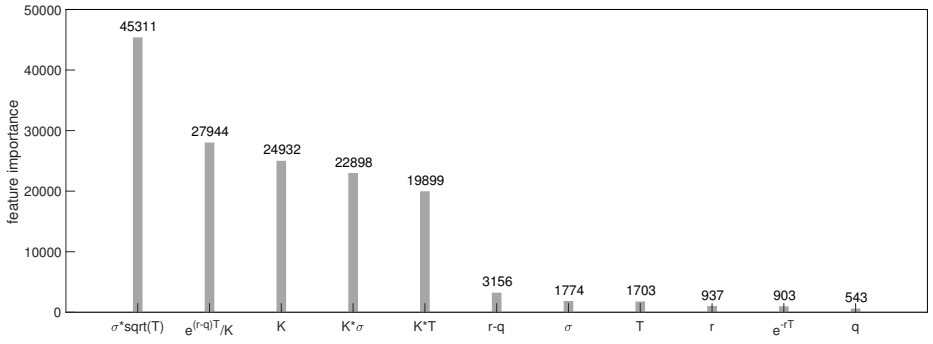
In this section we shift our attention from predictive performance to interpretability of the pricing models. For both applications, we look under the hood of the gradient boosting model and try to reveal how the price is constructed and interpreted.

4.1 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 7 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, both models consist of 10 000 regression trees with maximum 16 leaves. The aggregate feature importance is therefore bounded by $(16-1)\times 10\,000 = 150\,000$ splits. Figure 7a 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 7b 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 140 984 of the 150 000 splits in the



(a) Bonus certificates.



(b) American put options.

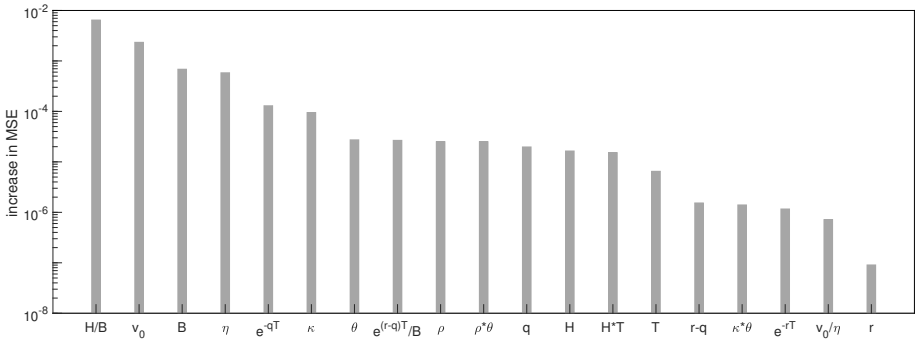
Figure 7: Feature importance plots for the models trained on 100 000 instances, measured by the number of times each feature is used in the model.

model. Note that four of these five variables are constructed by feature engineering, again indicating the importance of this procedure.

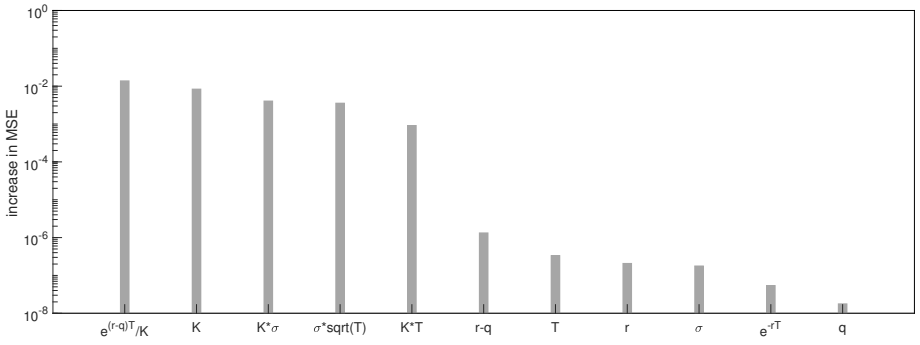
Alternatively, the important features can be identified by directly looking at the feature's impact on the predictions made by the model, i.e by computing the increase in prediction error when this feature is not available. Permutation feature importance, introduced by [1], makes a feature irrelevant 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 8 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 7a and Figure 8a 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 8b agree with those in Figure 7b. Moreover, they again have a significantly higher importance value than the remaining input variables.

4.2 Feature impact and prediction smoothness

Partial dependence plots [8] and individual conditional expectation curves [10] 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



(a) Bonus certificates.



(b) American put options.

Figure 8: Out-of-sample permutation feature importance plots for the GBM models trained on 100 000 instances.

value changes when a subset of features $\boldsymbol{x_S}$ varies, i.e.

$$f_S^{PD}(\boldsymbol{x_S}) = \frac{1}{n}\sum_{i=1}^{n} F_K(\boldsymbol{x_S}, \boldsymbol{x_i^*}) \tag{18}$$

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 (18). For the ith observation in the training set, the ICE function is given by

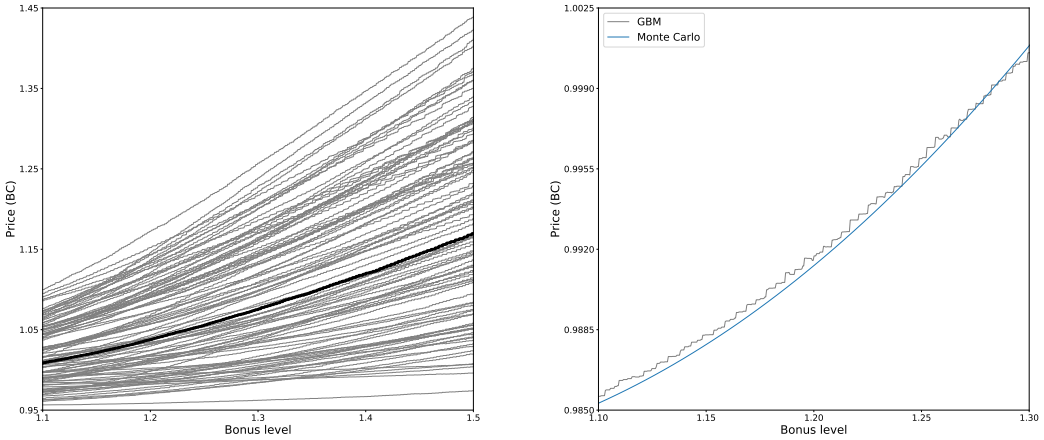
$$f_{i,S}^{ICE}(\boldsymbol{x_S}) = F_K(\boldsymbol{x_S}, \boldsymbol{x_i^*}). \tag{19}$$

Averaging all n ICE curves produces again the partial dependence plot. Figure 9a 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

$$\boldsymbol{x_S} = \left(B,\ \frac{H}{B},\ \frac{e^{(r-q)T}}{B}\right), \tag{20}$$

as the predictions 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_i,

r_i, q_i and T_i in the training set. Figure 9a 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. Figure



(a) ICE and partial dependence curves.

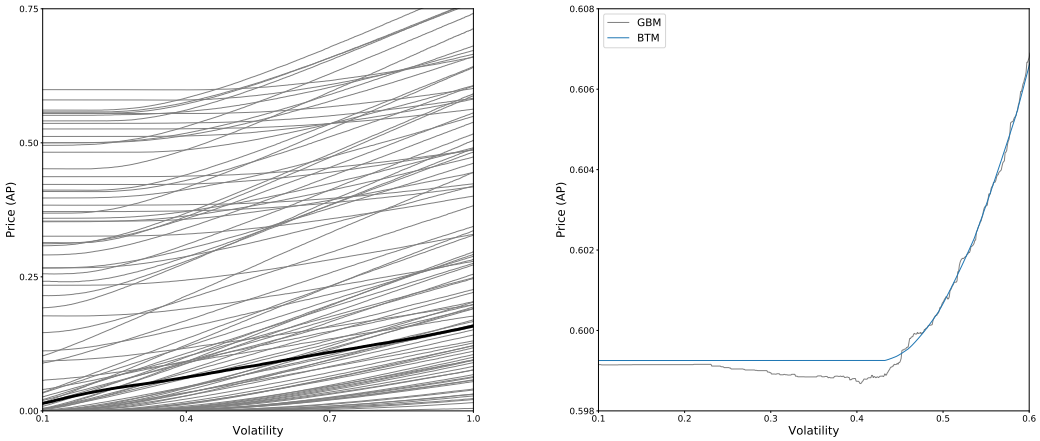
(b) An ICE curve in detail.

Figure 9: ICE curves (grey) and partial dependence plot (black) for the GBM model predicting prices of bonus certificates.

9b zooms in on one arbitrary ICE curve. It turns out that the expected monotonic relation is not consistently satisfied. This imperfection could be tackled by imposing a (marginal) positive monotonic constraint for B, which would enforce the price to increase with B, all other variables being constant. However, an increase in B leads to changes in $\frac{H}{B}$ and $\frac{e^{(r-q)T}}{B}$ and the constraint will not be satisfied in practice, unless the feature engineering procedure is dropped. For this reason, we did not impose any monotonic constraints in our models. The stepwise curve in Figure 9b moreover affirms 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 9 will never be smooth. As a consequence, one should be careful when computing price sensitivities using these models. Figure 10 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

$$\boldsymbol{x_S} = \left(\sigma,\ \sigma\sqrt{T},\ K\sigma\right), \tag{21}$$

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 10a. Figure 10b shows one arbitrary ICE curve in detail, which illustrates that the expected monotonic relation for σ is again violated. The true price curve, given by the prices computed by the benchmark binomial tree model, is added in blue to give an idea about the accuracy of the predictions. The predicted price decreases when σ goes from 20% to 40% and is moreover too low in this volatility range. However, one should take into account that the scale on the y-axis in Figure 10b is rather small.



(a) ICE and partial dependence curves.

(b) An ICE curve in detail.

Figure 10: ICE curves (grey) and partial dependence plot (black) for the GBM model predicting prices of American put options.

5 Conclusion

In this paper, we illustrated 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. On the other hand, it can be hard to find an optimal configuration and the resulting model highly depends on human choices. 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 the last part of the paper, we discussed how tree-based models can be understood better. 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. Finally, we want to point out that the same methodology can be applied to other (parametric) pricing models and products. However, we believe that an error analysis should be carried out carefully for each specific application.

References

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

[2] 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.

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

[4] Crépey, S. and Dixon, M. F. (2019) Gaussian process regression for derivative portfolio modeling and application to CVA computations. Available at SSRN: `http://dx.doi.org/10.2139/ssrn.3325991`.

[5] 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**.

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

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

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

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

[10] 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.

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

[12] 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.

[13] 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.

[14] Horvath, B., Muguruza, A., and Tomas, M. (2019) Deep learning volatility. ArXiv: 1901.09647.

[15] 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.

[16] 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. *Conference on Neural Information Processing Systems (NIPS)*, Long Beach, CA, USA.

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

[18] 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.

[19] 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.