KU LEUVEN

Machine Learning for Derivative Pricing

Gaussian processes vs. Gradient boosting

Sofie Reyners

Joint work with Jesse Davis, Laurens Devos, Jan De Spiegeleer, Dilip Madan and Wim Schoutens

Derivative pricing is time-consuming...

- Sophisticated models
- Exotic products

  → we need to rely on numerical methods, e.g. Monte Carlo simulations

- Portfolio valuation
  - Sensitivity analysis
  - VaR/ES calculations

… and markets are moving!

time-consuming algorithms

$\updownarrow$

continuously moving markets

$\rightarrow$ prices are outdated when available, risk calculations cannot be completed in time, …

Let a machine learn the pricing function

machine learning

product, market and model parameters

model price

time-consuming method

Expensive pricing function is summarized with machine learning.

Let a machine learn the pricing function

machine learning

product, market and model parameters

model price

time-consuming method

When training is completed, prediction is extremely fast!

Literature

- Neural networks:
  - Ferguson & Green (2018), Deeply learning derivatives.
  - Liu, Oosterlee & Bohte (2019), Pricing options and computing implied volatilities using neural networks.
  - Horvath, Muguruza & Tomas (2019), Deep learning volatility.
  - ...

Literature

- Gaussian process regression:
  - De Spiegeleer, Madan, Reyners & Schoutens (2018), Machine learning for quantitative finance: fast derivative pricing, hedging and fitting.
  - Crépey & Dixon (2019), Gaussian process regression for derivative portfolio modeling and application to CVA computations.
  - Sousa, Esquível & Gaspar (2012), Machine Learning Vasicek Model Calibration with Gaussian Processes.
- Gradient boosting machines:
  - Davis, Devos, Reyners & Schoutens, Gradient Boosting for Quantitative Finance. Working paper.

Gaussian process regression

Gaussian process regression (GPR)

Consider a training set $(X, \boldsymbol{y}) = \{(\boldsymbol{x_i}, y_i) \mid i = 1, \dots, n\}$ and assume that

$$y_i = f(\boldsymbol{x_i}) + \varepsilon_i$$

- $f(\boldsymbol{x})$ is a Gaussian process, characterized by two functions:
  - mean function: $m(\boldsymbol{x}) = \mathbb{E}\big[f(\boldsymbol{x})\big]$
  - kernel function: $k(\boldsymbol{x}, \boldsymbol{x}') = \mathrm{Cov}(f(\boldsymbol{x}), f(\boldsymbol{x}'))$
- $\varepsilon_i \sim \mathcal{N}(0, \sigma_n^2)$ are i.i.d. random variables representing the noise in the data.

Gaussian process

- $f(\boldsymbol{x}) \sim GP(m(\boldsymbol{x}), k(\boldsymbol{x}, \boldsymbol{x}'))$
- If $(X, \boldsymbol{f}) = \{(\boldsymbol{x_i}, f_i) \mid i = 1, \ldots, n\}$ is a sample from $f(\boldsymbol{x})$, then

$$\boldsymbol{f} \sim \mathcal{N}\left(M(X), K(X, X)\right)$$

with

$$M(X) = \begin{bmatrix} m(\boldsymbol{x_1}) \\ \vdots \\ m(\boldsymbol{x_n}) \end{bmatrix}, \qquad K(X, X) = \begin{bmatrix} k(\boldsymbol{x_1}, \boldsymbol{x_1}) & \ldots & k(\boldsymbol{x_1}, \boldsymbol{x_n}) \\ \vdots & \ddots & \vdots \\ k(\boldsymbol{x_n}, \boldsymbol{x_1}) & \ldots & k(\boldsymbol{x_n}, \boldsymbol{x_n}) \end{bmatrix}.$$

GPR: a Bayesian method

- Don't model the relation as one function, but as a distribution over functions.
- Procedure:
  1. Start from a prior GP (with zero mean)
     - $\rightarrow$ prior knowledge: smooth function, periodic function, ...
     - $\rightarrow$ prior distribution over functions
  2. Include observed data points
  3. Compute a posterior GP

Posterior distribution

Only consider functions that agree with the data.

- Take new inputs X_*, with corresponding (unknown) function values $\boldsymbol{f}_*$.
- Joint distribution of training outputs and function values:

$$\begin{bmatrix} \boldsymbol{y} \\ \boldsymbol{f}_* \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}\ ,\ \begin{bmatrix} K(X,X) + \sigma_n^2 I & K(X,X_*) \\ K(X_*,X) & K(X_*,X_*) \end{bmatrix}\right)$$

Posterior distribution

- Condition on the observations:

$$\boldsymbol{f}_*|X_*, X, \boldsymbol{y} \sim \mathcal{N}\Big(\mu, \Sigma\Big)$$

with

$$\mu = K(X_*, X)\big[K(X, X) + \sigma_n^2 I\big]^{-1}\boldsymbol{y}$$

$$\Sigma = K(X_*, X_*) - K(X_*, X)\big[K(X, X) + \sigma_n^2 I\big]^{-1}K(X, X_*)$$

- Point prediction: μ

Kernel function

Squared exponential kernel function (with ARD):

$$k(\boldsymbol{x}, \boldsymbol{x}') = \sigma_f^2 \exp\left(-\frac{1}{2}\sum_{j=1}^{d}\frac{|x_j - x_j'|^2}{\ell_j^2}\right)$$

with hyperparameters σ_f and $\boldsymbol{\ell}$:

- σ_f^2 : signal variance
- $\ell_1, \ldots, \ell_d$: characteristic length-scale parameters

$\rightarrow$ Hyperparameters (including σ_n) are estimated on the training set, usually with MLE.

Machine learning for derivative pricing

- Construct a training set:

$$\text{product, market and model parameters} \xrightarrow{\text{time-consuming method}} \text{model price}$$

Machine learning for derivative pricing

- Construct a training set:

product, market and model parameters $\xrightarrow{\text{time-consuming method}}$ model price

$\downarrow$

$$\boxed{\text{sample } n \text{ random combinations } \boldsymbol{x_i} \longrightarrow \text{compute } n \text{ corresponding prices } y_i}$$

Machine learning for derivative pricing

- Construct a training set:

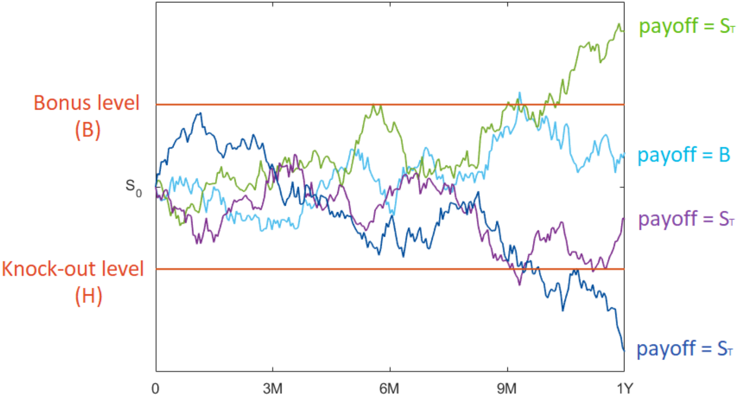
product, market and model parameters $\xrightarrow{\text{time-consuming method}}$ model price

- Fit a GPR model.

- Fast prediction of new model prices.

Case study:
pricing a structured product

Bonus certificate's payoffs



→ Price according to Heston model, using Monte Carlo simulation.

Build a training set

Parameter ranges:

Product/market	**Heston model** [(*)]
$B \in [105\%, 155\%]$	$\kappa \in [0.2, 1.6]$
$H \in [55\%, 95\%]$	$\rho \in [-0.95, -0.25]$
$T \in [11M, 1Y]$	$\theta \in [0.15, 0.65]$
$r \in [2\%, 3\%]$	$\eta \in [0.01, 0.25]$
$q \in [0\%, 5\%]$	$v_0 \in [0.01, 0.25]$

$\rightarrow$ sample parameter combinations + calculate corresponding prices.

(∗) κ = rate of mean reversion, ρ = correlation stock - vol, θ = vol-of-vol,
η = long run variance, v_0 = initial variance.

Pricing a bonus certificate

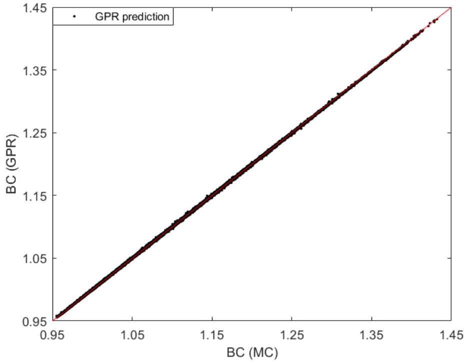
- Train a GPR model

$$\begin{matrix} B,\, H,\, T,\, r,\, q, \\ \kappa,\, \rho,\, \theta,\, \eta,\, v_0 \end{matrix} \xrightarrow{\text{learn this relation}} \text{BC price}$$

$\rightarrow$ matrix inversion + hyperparameter optimization

- Construct a test set (100 000 instances):
  - similarly to training set construction
  - slightly smaller parameter intervals

GPR's predictions



Maximal error

absolute:	0.0058
relative:	0.51%

Prediction:	84 seconds
Speed-up	$\times$ 2900

$\rightarrow$ GPR model trained on 10 000 samples, tested on 100 000 samples.

KU LEUVEN

Gradient Boosting Machines

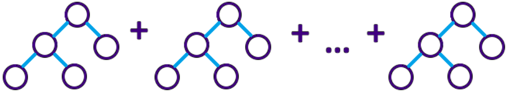
Gradient boosting machine (GBM)

= ensemble of regression trees h

$$\hat{y}_i \;=\; F_K(\boldsymbol{x_i}) \;=\; \sum_{k=0}^{K} f_k(\boldsymbol{x_i}) \;=\; c + \sum_{k=1}^{K} \nu h_k(\boldsymbol{x_i})$$

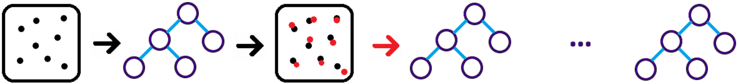
with $0 < \nu \leq 1$ a shrinkage parameter.

→ Many simple models f_k are combined into one strong model.



Gradient BOOSTING machine

- GBM is trained **stage-wise**:

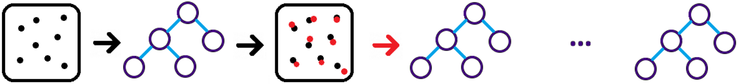


$\rightarrow$ Trees are added sequentially:

f_k tries to correct for the errors made by model F_{k-1}.

$\rightarrow$ Previously fitted trees are not readjusted.

GRADIENT boosting machine



How to measure the errors?

- Use pseudo-residuals

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

where $\mathcal{L}$ is a loss function that measures how well the data is captured by the model.

Case study:
pricing a structured product

Boosting derivative pricing

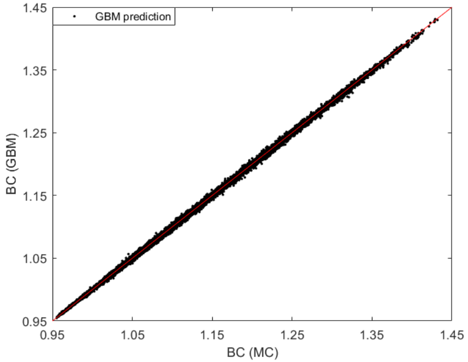
- Construct a training set.
- Train a GBM model:

$$\begin{matrix} B,\ H,\ T,\ r,\ q, \\ \kappa,\ \rho,\ \theta,\ \eta,\ v_0 \end{matrix} \xrightarrow{\text{learn this relation}} \text{BC price}$$

$\rightarrow$ LightGBM implementation.

- Fast prediction of new model prices.

GBM's predictions



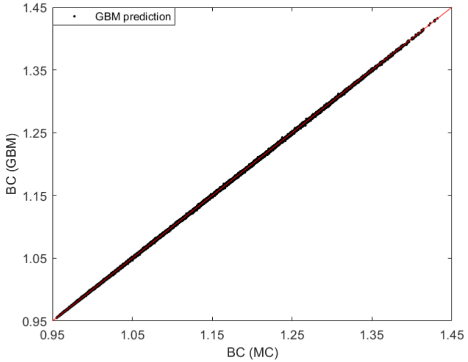
Maximal error

absolute:	0.0146
relative:	1.11%

Prediction:	42 seconds
Speed-up	$\times$ 5800

$\rightarrow$ GBM model trained on 10 000 samples, tested on 100 000 samples.

GBM's predictions



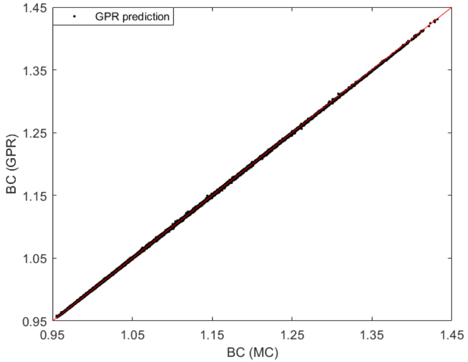
Maximal error

absolute: 0.0052
relative: 0.41%

Prediction: 42 seconds
Speed-up $\times$ 5800

$\rightarrow$ GBM model trained on 100 000 samples, tested on 100 000 samples.

GPR's predictions



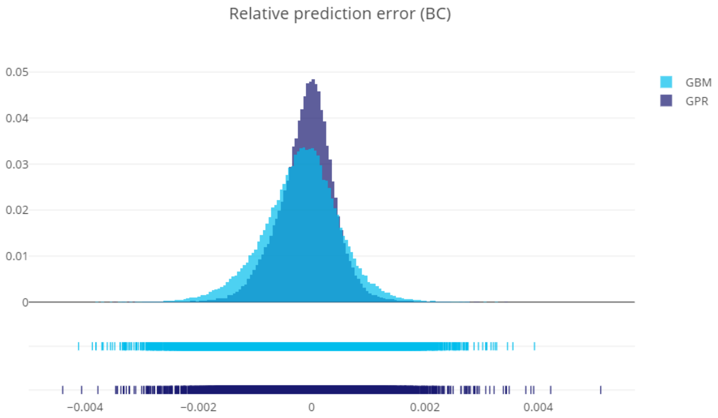
Maximal error

absolute:	0.0058
relative:	0.51%

Prediction:	84 seconds
Speed-up	$\times$ 2900

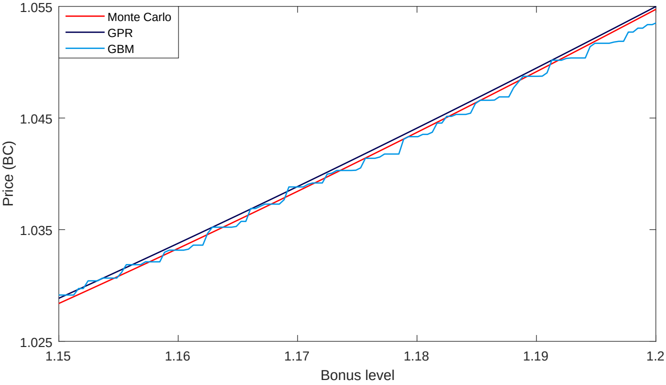
$\rightarrow$ GPR model trained on 10 000 samples, tested on 100 000 samples.

GBM vs. GPR



KU LEUVEN

GPR predictions are smoother



Fixed parameters: $H = 80\%$, $T = 1Y$, $r = 2.5\%$, $q = 3\%$, $\kappa = 1$, $\eta = 0.05$, $\theta = 0.5$, $\rho = -0.7$, $v_0 = 0.05$.

GPR has built-in analytic derivatives

- GPR's point prediction for $\boldsymbol{x}_*$:

$$f^*(\boldsymbol{x}_*) = K(\boldsymbol{x}_*, X)\underbrace{\left[K(X,X) + \sigma_n^2 I\right]^{-1}\boldsymbol{y}}_{\alpha}$$

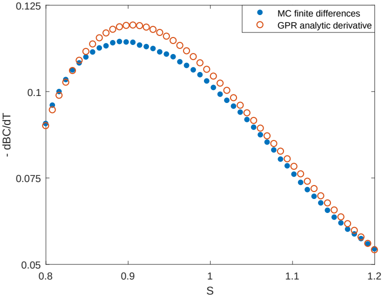
- Derivatives w.r.t. $\boldsymbol{x}_*$:

$$\frac{\partial f^*(\boldsymbol{x}_*)}{\partial \boldsymbol{x}_*} = \frac{\partial K(\boldsymbol{x}_*, X)}{\partial \boldsymbol{x}_*}\boldsymbol{\alpha} = -\Lambda^{-1}\tilde{X}_*{}^T\left[K(\boldsymbol{x}_*, X)^T \circ \boldsymbol{\alpha}\right]$$

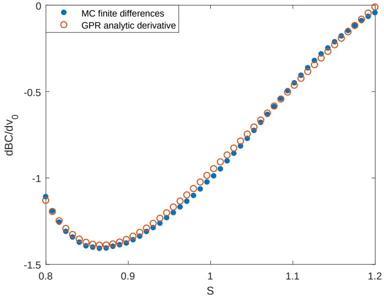
with

$$\tilde{X}_* = [\boldsymbol{x}_* - \boldsymbol{x}_1, \ldots, \boldsymbol{x}_* - \boldsymbol{x}_n]^T, \qquad \Lambda = \begin{bmatrix} \ell_1^2 & & 0 \\ & \ddots & \\ 0 & & \ell_d^2 \end{bmatrix}$$

Smooth Greek profiles for free!



(a) GPR's Theta



(b) GPR's Vega

$\rightarrow$ Monte Carlo simulations based on 100 million price paths.

Conclusion

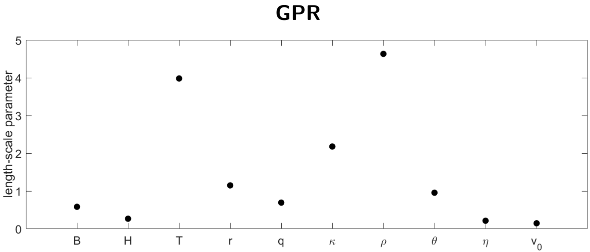
- Time-consuming pricing methods
- Machine learning methods
  - Gaussian process regression (GPR)
  - Gradient boosting machines (GBM)
- Pricing a structured product:
  - Speed-ups of several orders of magnitude.
  - GBM prediction is faster, training scales more easily.
  - GPR provides smoother price predictions and Greeks.

Thank you!

Contact: sofie.reyners@kuleuven.be

Hyperparameters

ntrees = 10 000	bagging_freq = 1
learning_rate = 0.2	bagging_fraction = 0.5
num_leaves = 16	L2_regularization = 0.8
min_obs_in_leaf = 24	booster = DART
max_bin = 255	drop_rate = 0.1



$$\sigma_f = 0.4580$$

$$\sigma_n = 0.0011$$