Ensemble Kernel Methods for Dynamic Portfolio Valuation

Francis Clément, Lips Thomas, Novaković Miloš
CS-433 Machine Learning - Project 2

***Abstract*—In this paper, the potential of ensemble learning methods for dynamic portfolio valuation is assessed. First a portfolio is modelled and a baseline estimator is created, both strongly guided by the results of Boudabsa and Filipovic [1]. Then a number of aggregating and boosting techniques are evaluated and compared to the baseline performance. The reults show that although some ensemble methods perform well in lower dimensional cases, they seem to be unable to reduce the error significantly in the high-dimensional case.**

I. Introduction

Estimating the value of a financial portfolio at some time T in the future is an essential step in risk management for banks, insurance companies and other financial institutions. In their paper [1], Boudabsa and Filipović provide a first Machine Learning based approach to this problem. Their method is sample efficient as only a modest amount of training samples is required. Furthermore it allows to assess the portfolio in all points in time, up until T (dynamic portfolio valuation) using a closed-form approximator. The results from [1] (which serve as a baseline) are recreated and a number of ensemble methods are evaluated to assess if they are able to improve this baseline, while making sure that closed-form evaluation of the value process is still possible.

After validating that the ensemble methods indeed improve on the result in lower dimensions, they are evaluated in the higher dimensions using SCITAS[1], the HPC infrastructure at the EPFL.

II. Background

A. Portfolio Modelling and Valuation

To model a portfolio, a driver process

$$X = (X_0, ..., X_T) \tag{1}$$

is introduced, which generates all randomness in the values of the portfolio assets (stocks, bonds, options and other financial instruments) and hence makes these values deterministic when given a sample trajectory x of X. The components X_t are independent random variables, distributed according to the distribution $\mathbb{Q}_t$, the risk-neutral pricing measure.

The goal of portfolio valuation is to compute its dynamic value process given by the martingale

$$V_t = \mathbb{E}[f(X)|\mathcal{F}_t], t = 0...T \tag{2}$$

where $f(X)$ models the cumulative cash flow of the portfolio given its stochastic drivers X. $\mathcal{F}_t$ is the filtration of the stochastic process and models the information available at time t.

Computing V_t analytically is usually impossible, as the portfolios contain financial products with very complex cash flow functions such as path-dependent options. Therefore one has to fall back on simulations to make an estimation of V_t.

This estimation is usually performed by introducing the measure $\mathbb{Q}$ of the stochastic drivers of the economic model up until time T, and performing nested Monte Carlo Simulations to estimate the value of the portfolio at time T for each Monte Carlo Simulation of the $t+1$ first components of the driver X.

The closed-form solution as presented in [1], allows to compute V without falling back to these computationally expensive simulations, which is a significant improvement.

[1]https://www.epfl.ch/research/facilities/scitas/

B. Machine Learning Approach to Portfolio Valuation [2]

In their paper [1], Boudabsa and Filipović show how to obtain the closed-form estimation of the value process. Firstly, the cumulative cash flow function $f(\cdot)$ is approximated with a function $f_\lambda(\cdot)$, which is the λ-regularized projection of the function $f(\cdot)$ on a well chosen reproducing kernel Hilbert Space (RKHS). Secondly, a number of samples are drawn from the driver process $\boldsymbol{X} = (X^{(1)}, ..., X^{(N)})$ according to the predefined measure $\mathbb{Q}$. Thirdly, the values $f(X^{(i)})$ are computed to obtain the dataset $\mathcal{D}$, which is then used to train an estimator $f_X(\cdot)$ for the function $f_\lambda(\cdot)$.

The authors also show that with a well chosen RKHS, the estimator of the value process

$$V_{\mathbf{X},t} = \mathbb{E}_{\mathbb{Q}}[f_{\mathbf{X}}(X)|\mathcal{F}_t], t = 0, ..., T \tag{3}$$

is given in closed form and has a bounded error, which increases with the time t. Lastly the authors prove that Gaussian Process Regression (GPR) gives this expression for the estimator $f_{\mathbf{X}}(\cdot)$ [1].

C. Gaussian Process Regression

Supervised Machine Learning methods train an estimator on the dataset $\mathcal{D}$ containing the data samples $X^{(i)}$ and their labels $f(X^{(i)})$. To be able to do this, assumptions need to be made about the unknown function $f(\cdot)$ [2]. The most common approach is to restrict the class of functions that is considered, according to some belief or knowledge about this function $f(\cdot)$ (e.g. Regression, SVM etc.). A second approach is to not restrict the class of functions directly, but to attach a prior probability (again according to some beliefs about the function $f(\cdot)$) to each function and to update these probabilities based on the dataset $\mathcal{D}$.

Gaussian process regression (GPR) takes this latter approach. A Gaussian process is a collection (possibly of infinite size) of random variables, any finite number of which have a Gaussian distribution [3]. As each variable is Gaussian, the process can be fully specified as follows:

$$f_X(\mathbf{x}) \sim \mathcal{GP}(m(\mathbf{x}), K(\mathbf{x}, \mathbf{x}')). \tag{4}$$

Usually the outputs are assumed to have a zero mean in which case $m(\mathbf{x}) = 0$ and one only needs to specify the covariance function, usually referred to as the kernel. This kernel contains beliefs about the function $f(\cdot)$ (e.g. smoothness, noise types etc.) and represents the prior probability over the function space. The GPR is using these prior probabilities to create a posterior probability, based on the dataset. If the kernel contains hyperparameters, the GPR can use a maximum likelihood estimation on the dataset to automatically find the optimal parameters within a given range, making an extensive grid search unnecessary [2].

The big downside of GRPs is their complexity. To determine the posterior, one needs to compute and invert the covariance matrix over all points in the dataset. This matrix inversion has a time-complexity of $\mathcal{O}(N^3)$ and a memory-complexity of $\mathcal{O}(N^2)$ [4].

[2]Quite a bit of mathematical details were omitted in this part because they are not directly relevant to this project. See [1] for a detailed derivation.

D. *Ensemble Learning*

With ensemble learning, the idea is to use multiple estimators to improve on the performance of a single model (the base estimator). Broadly there exist 3 categories of ensemble methods: Aggregating, Boosting and Stacking. Stacking is not used in this paper, the other two approaches are briefly explained below.

1) Aggregating: Multiple estimators are averaged as:

$$f(X) = \sum_{i=1}^{N} w_i(X) f_i(X) \tag{5}$$

where w_i is the normalized relative weight of the i^{th} estimator $f_i(X)$. Often these weights are set to $1/N$ (mostly due to a lack of information about the estimations). This is referred to as hard voting. The more general case is called soft voting.

The main goal of aggregating methods is to create estimators that are as uncorrelated as possible so that the effects of averaging the estimations are as large as possible [5]. One way of doing this is by combining several learning methods to make their errors as independent as possible. However, this is not possible in case of the value process estimation, because it requires a method that yields closed form solution.

Another approach is to use subsets of the dataset to train multiple instances of estimators. There are two main methods to sample these subsets from the dataset: bagging (with replacement) and pasting (without replacement) [5]. Reducing the size of the subset will in general increase the bias of the different estimators individually, but by combining them one can usually decrease the bias again to at least the bias of the base estimator while having a smaller variance.

2) Boosting: Refers to any Ensemble method that can combine several weak learners into a strong learner. The general idea of most boosting methods is to train predictors sequentially, each trying to correct its predecessor[5]. In this paper, two version of Gradient Boosting are used: Batch Gradient Boosting (BGB) and Stochastic Gradient Boosting (SGB). The key difference between both is that BGB takes all training data, whereas the SGB takes only a random subset of the training data to train each model. SGB trades a higher bias for a lower variance. It also speeds up training considerably [5].

BGB training of the sequential chain of GPRs models is shown in Algorithm 1. Where at the end of the computation, the estimation $\hat{\mathbf{y}}^*$ of the unknown label value $\mathbf{y}^*$ (of a new datapoint $\mathbf{X}^*$) is performed by summing up the predictions of all models in the ensemble

$$\mathbf{y}^* \approx \hat{\mathbf{y}}^* = \sum_{i=1}^{\text{numModels}} models[i].\text{predict}(\mathbf{X}^*) \tag{6}$$

Algorithm 1 Gradient Boosting training

```
1:  procedure GRADIENT BOOSTING TRAIN(NUM_MODELS, KERNEL, DATA, LABELS)
2:      $models \leftarrow$ Array$[num\_models]$.
3:      $residuals \leftarrow$ init residuals$(data)$
4:      $i \leftarrow 1$.
5:      while $i \leq num\_models$ do
6:          $models[i] \leftarrow$ Model_GaussianProcessRegressor$(kernel)$.
7:          $models[i]$.train(X := $data$, Y := $residuals$).
8:          $residuals \leftarrow residuals - models[i-1]$.predict$(data)$.
9:          $i \leftarrow i + 1$.
10:     return $models$.
```

III. Methods and Implementation

A. *Defining the Portfolio Model and Baseline Estimator*

Guided by [1], a portfolio model is defined, from which the train and test sets are generated. Additionally a Gaussian process estimator is created and used both to create a baseline and as a basemodel for the ensemble methods.

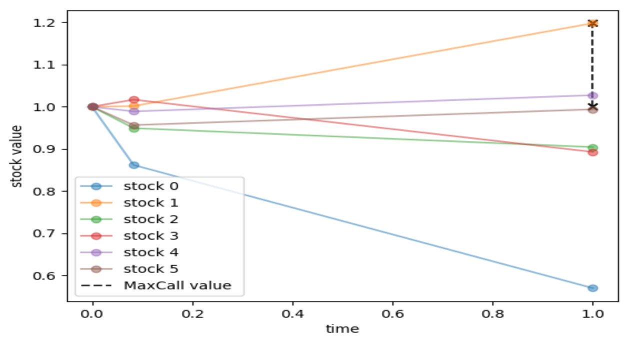

Figure 1: Illustration of stock prices using a driver sample and the resulting Maxcall value (dotted line)

The portfolio model used in this paper contains a single option on d stocks. These stock prices are modelled using the multivariate Black-Scholes model, where the driver variables are i.i.d. Gaussian random variables. The prices can be calculated for any time $t < T$ using the following recursive formulation:

$$S_{i,t} = S_{i,t-1} exp[\sigma_i X_t \sqrt{\Delta_t} + (r - \frac{1}{2}\|\sigma_i\|^2)\Delta_t], \tag{7}$$

where the stocks are assumed to be independent. Also, the initial stock values are set to 1.

Initially a low dimensional case with $d = 1$,T= $2, \Delta T = [1/12, 11/12]$ is defined. Later on we also consider a high dimensional case where $d = 6$.

The option is defined as a European max-call option, having a payoff function:

$$f(X) = e^{-rT}(\max_i S_{i,T} - K)^+, \tag{8}$$

where K is the strike price, i.e. the price payed for taking the option. The model is further simplified by taking the the risk-free rate r to be 0, which makes it possible to neglect the continuous discounting in both the payoff function (8) and the stock price calculations (7).

An example of the stock prices generated by the driver model and the resulting Max-call value of the portfolio can be seen in figure 1, for the high-dimensional case of $d = 6$.

The goal is to approximate $V_{X,t}$ for all t up to T. In this paper however, the focus in on approximating $V_{X,T}$ since this bounds the error on all other values of $V_{X,t}$, as shown by [1].

The metric for the error in the approximation of $V_{X,T}$ is the normalized $\| \cdot \|_2$ difference defined by

$$\frac{\|V_T - V_{X,T}\|_{2,\mathbb{Q}}}{V_0}. \tag{9}$$

which can be simplified to

$$\frac{\sqrt{\frac{1}{N}\sum_{i=1}^{N}(f(X^{(i)}) - f_X(X^{(i)}))^2}}{\frac{1}{N}\sum_{i=1}^{N} f(X^{(i)})}. \tag{10}$$

Since the samples $X^{(i)}$ are already drawn according to the distribution $\mathbb{Q}$ and since V_0 is the average value of $f(\cdot)$, as $\mathcal{F}_0$ contains no information about the trajectories.

The GPR uses two types of kernels: the Gaussian-exponentiated kernel

$$k_1(x, x'; \alpha, \beta) = e^{-\alpha\|x-x'\|^2 + \beta x^T x'} \tag{11}$$

and the White Noise kernel

$$k_2(x, x'; \lambda) = \lambda \mathbb{1}\{x = x'\}. \tag{12}$$

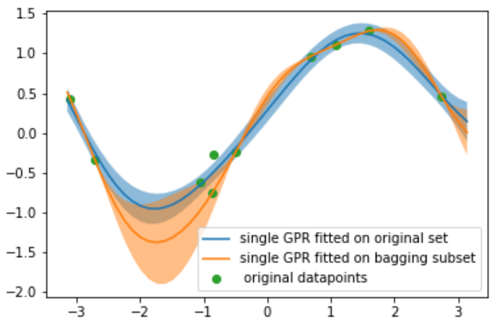

Figure 2: The effect of sampling with replacement on the mean and variance functions of the posterior of a single GPR

The first kernel is tractable and ensures that the approximation is given in closed form [1]. This kernel is also universal in $L^2_{\mathbb{Q}}$ meaning that the projection of $f(\cdot)$ onto the kernel space is lossless. The second kernel models noise on the data and serves as the regularization.

Guided by the results in [1], we set β to zero for the Max-call value function to reduce the complexity. Hence, the resulting kernel is a sum of the Radial Basis Function (RBF) kernel and the White Noise kernel.The hyperparameters α and λ are tuned by the model, as discussed earlier. For the implementation of The resulting kernel and the Gaussian process, the Scikit-learn library [6] is used.

As the driver data are generated, the only restrictions on the size of the dataset are that they should reflect real world settings. Gathering training data in real-world requires evaluating the complex value function of true portfolios for each trajectory. This is rather expensive (since the portfolio often depends on numerous other portfolio's etc). Therefore the size of the training set (N_{train}) is chosen to be 5000 for the lower dimensional case and 5000 or 20000 for the higher dimensional case, depending on the method that is used. For the test set there are no restrictions and it should be large enough to capture the error accurately (hence 100'000 is used when memory allows it, if not the size is reduced to 50'000).

B. Aggregating Methods

GPR estimators provide both the prediction (the mean of the process) and the variance of that prediction (process). Hence, the following weighting factors can be constructed:

$$w_i(X) = \left((\sigma_i(X) + \epsilon) \sum_j \frac{1}{\sigma_j(X) + \epsilon}\right)^{-1}. \tag{13}$$

Figure 2 provides an intuition as to why this weighting in combination with the bagging technique makes sense, i. e. when sampling points from the original dataset with replacement. The estimators in the ensemble will have posterior probabilities with differing variance functions (orange region is smaller than the blue region on the right part of the plot), related to the frequency of each sample in the subsets. This means that predictions for an unseen X can be more certain (smaller deviation) for one estimator than for another. Hence it makes sense to use the constructed weighting (13) to give more weights to predictors that are more certain about a specific region. However, the inverse situation on Figure 13 is equally as important, where the blue lines are thinner than orange ones on the left part of the plot.

A big downside of this soft voting mechanism however, is that it can only be used to estimate the value process at time T, since it takes the covariance function of the predictions explicitly into account during the weighting and makes closed-form evaluation of the process at any time other than T impossible. The hyperparameters of these aggregating ensembles are the size of each train set (expressed as fraction of the train set of the system) α and the number of estimators M.

C. Boosting Methods

As discussed earlier, the idea of Boosting to improve the estimations recursively by estimating the residuals of the previous estimator — the sequential chain of models. The initialisation of the residual can have a big impact on the performance of this chain. In this project, two types of estimator initialisation are tried. Firstly, the base estimator (denoted $\hat{\mathbf{y}}_{base}$ in (14)) is initialized with zeros , and secondly the base estimator initialized with the mean value of the value vector (estimator is plain sample mean estimator). The original Gradient boosting algorithm starts with estimator equal to zero, but the big disadvantage for this assumption is that it takes the program too long to converge. On the other hand, if the initial estimator is set to the sample mean of training labels, then the initial value of the estimator is already close enough to the convergense value, hence making the rate of convergence faster.

Additionally, in the original BGB and SGB methods, the constant learning rate (γ) is assigned to the new i^{th} prediction (estimation) $\hat{y}^{(i)}$, which is a label prediction based on i^{th} GPRs model (with kernels from (11) and (12)) trained on the whole dataset ($\alpha = 1$) in the BGB case, or trained on the random subsample-fraction of the dataset ($\alpha < 1$) in the SGB case. Both, γ and α (in the SGB case) are hyperparameters. The training and estimation is given by:

$$\hat{\mathbf{y}}_{final} = \hat{\mathbf{y}}_{base} + \sum_{i=1}^{\text{numModels}} \gamma \cdot \hat{\mathbf{y}}_i, \tag{14}$$

where $\hat{\mathbf{y}}_i = models[i]$.predict($\mathbf{y} - (\hat{\mathbf{y}}_{base} + \sum_{j=1}^{\text{i-1}} \gamma \cdot \hat{\mathbf{y}}_j)$) for every i^{th} iteration in training, with the learning rate γ and the "numModels" denotes number of models in the boosting chain of models.

Furthermore, the early stopping technique was used as in [5]. The main idea is as follows: if during the i^{th} iteration of training, the result (sum of all previous predictions from 1 to i) does not improve by at least $\epsilon > 0$ in the $p \in \mathbb{N}$ consecutive iterations, then the early stopping algorithm is going to exit the training phase, and return exactly $(i - p)$ models. Both, ϵ and p are hyperparameters. This technique improves the speed of boosting algorithm (hence the name "early stopping").

D. Scaling up the experiments using SCITAS

After evaluating the methods in the lower dimensional case, their performance is assessed in the higher dimensional case (which makes the value function a lot more complex). At this point it becomes infeasible to do the training on a regular desktop (in terms of both processing power and memory requirements) and therefore the SCITAS HPC infrastructure is used. To leverage the processing power of the HPC, the MPI standard is used to parallellize the hyperparameter searches and the training/evaluation of the different estimators in the ensemble when possible. To this end, the *mpi4py* [7] library is used.

Converting the ensemble methods to MPI is rather straightforward. The convenient MPIPoolExecutor interface offered by *mpi4py* is used to manage the jobs, while asynchronous broadcasting is used to distribute the train and test sets to all worker tasks when they are spawned.

IV. Results and Discussion

A. Baseline errors

The baseline errors are obtained by training a single estimator on the full training set. For the case where $d = 2, N = 5000$, the baseline error is around 0.142. For the high dimensional case where $d = 6$, the baseline error is around 0.151 when $N = 5000$ and 0.124 when $N = 20000$ (this is obtained from [1]).

B. Aggregating Methods

First, the two basic methods (hard pasting and hard bagging), and the method that was constructed before (soft bagging), are compared in the low dimension case. The range of the hyperparameters of the

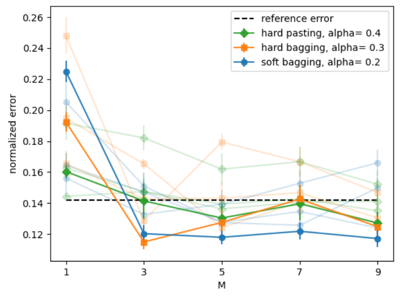

Figure 3: Comparison of the different aggregating methods in low dimensions. The best alpha values are highlighted for each method.

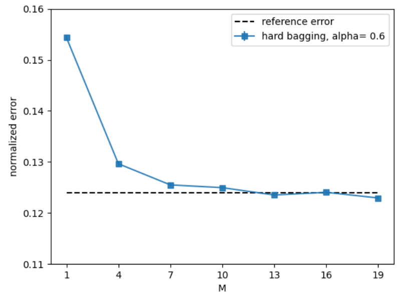

Figure 4: Hard bagging results for the high dimensional case

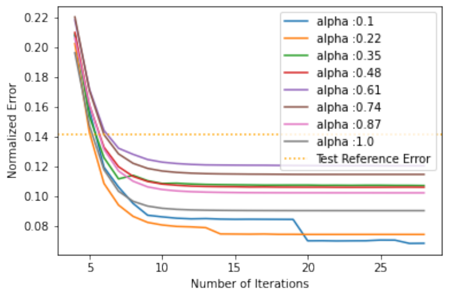

Figure 5: Boosting results for the low dimensional case. case

ensemble is set to $\alpha \in \{0.2, 0.3, 0.4, 0.5\}$ and $M \in \{1, 3, 5, 7, 9\}$. The reported metric is the average normalized loss over 3 different test sets. The results can be seen in figure 3, which compares the results of the different hyperparameter settings and methods with the baseline error and highlights the best value of α for each method.

As expected, the bagging approaches have better performance than the pasting approach, as they introduce more variety in the training sets of the different estimators in the ensemble. However, the improvements are rather small and the lowest error obtained improves only 0.02 on the baseline.

Based on these first results, SCITAS is used to evaluate the performance of bagging on the high dimensional case. The improvements over the baseline estimator are largelely reduced for this case with the improvement being smaller than 0.003 for hard bagging, as can be seen in figure 4. Even with soft bagging, the results are similar. It is hence clear that aggregating methods do not provide large improvements in terms of accuracy. However, the bagging methods require only α^3 the amount of time and α^2 the amount of memory compared to the baseline estimator, if all M estimators are trained in parallel. This is by itself already quite an improvement and is also related to the original motivation for aggregating methods [8].

C. Boosting Methods

Both boosting algorithms (BGB and SGB) are first evaluated on the low dimensional case. As expected both BGB ($\alpha = 1$) and SGB ($\alpha < 1$), yield better results than the base model. For the best runs, the error is almost halved as can be seen in figure 5. It is clear that in this low-dimensional case, SGB performs better. Furthermore it is also faster to train as the size of each training set is smaller.

Next, the HPC is used to evaluate BGB and SGB for higher dimensions (d=6, T=2). However due to the sequential chaining of boosting, training takes a lot more time and hence the number of training points was kept at 5000. As can be seen in figure 6, the improvements are almost negligible in this case, both for BGB and SGB. Some experiments with 20000 training points and about 40 iterations, did not even manage to improve on the baseline.

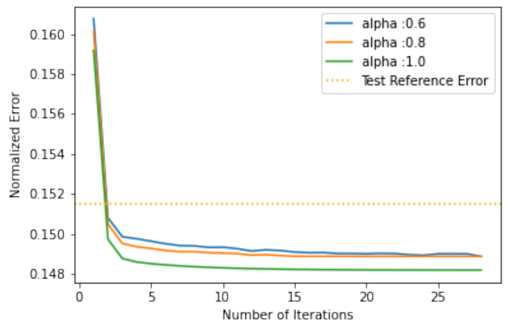

Figure 6: Boosting results for the high dimensional case

D. Reflection on the results

From the previous sections, two conclusions can be made.

First of all, the improvements that were made with the aggregating methods are very small. This is probably due to the fact that GPRs are already very strong learners on their own, which reduces the benefits of aggregating them in comparison to weaker learners (and the same goes for the boosting sequence).

The second conclusion is that the improvements in the lower dimensions, do not translate to higher dimensions. For the aggregating methods, this is most likely related to the fact that in higher dimensions, way more training data is required to cover the high-dimensional space which makes splitting up relatively small sets of training data not beneficial. For boosting on the other hand, kernel methods are known to have scalability issues in higher dimensions, which might explain why even boosting cannot improve the results beyond the baseline.

V. Further Work

In this paper, only the European Max-Call was considered. A logical extension would hence be to evaluate how the different methods perform on other option types such as the European Min-Put or the Barrier reverse convertible which are both used in [1]. Another extension would be to evaluate the performance of the ensemble methods on $V_{X,t}, t < T$. This performance is expected to be a lot better than and proven to be at least as good as $V_{X,T}$ [1].

VI. Conclusion

In this paper, a number of different ensemble methods are used to evaluate on the constructed portfolio containing d assets and one European Max-Call option. Most notably the boosting methods were able to reduce the error significantly below the baseline created in [1]in lower dimensions. However all ensemble methods struggle to improve on the baseline in higher dimensional cases. This leads to the conclusion that GPRs are probably not well suited for ensemble methods due to the fact that they are by themselves already strong learners.

Acknowledgments

The authors would like to thank Lotfi Boudabsa, Doctoral Assistant at the College of Management of Technology at EPFL and author of [1], who supervised this project and introduced us to the challenging topics of kernel methods and portfolio valuation.

References

[1] L. Boudabsa and D. Filipovic, "Machine learning with kernels for portfolio valuation and risk management," 2020.

[2] C. Rasmussen and C. Williams, *Gaussian Processes for Machine Learning*. the MIT Press, 2016. [Online]. Available: www.GaussianProcess.org/gpml

[3] R. Turner, "Gaussian processes: From the basics to the state-of-the-art," 2017. [Online]. Available: http://cbl.eng.cam.ac.uk/pub/Public/Turner/News/imperial-gp-tutorial.pdf

[4] M. Belyaev, E. Burnaev, and Y. Kapushev, "Exact inference for gaussian process regression in case of big data with the cartesian product structure," 2014.

[5] A. Géron and a. O. M. C. Safari, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition*. O'Reilly Media, Incorporated, 2019. [Online]. Available: https://books.google.ch/books?id=O2VJzQEACAAJ

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[7] L. Dalcin, R. Paz, and M. Storti, "Mpi for python," *Journal of Parallel and Distributed Computing*, no. 65, pp. 1108–1115, 2005. [Online]. Available: http://dx.doi.org/10.1016/j.jpdc.2005.03.010

[8] L. Breiman, "Pasting small votes for classification in large databases and on-line." *Mach. Learn.*, vol. 36, no. 1-2, pp. 85–103, 1999. [Online]. Available: http://dblp.uni-trier.de/db/journals/ml/ml36.html#Breiman99