Comparison of different machine learning algorithms for prediction of wind evolution

Yiyin Chen and Po Wen Cheng
Stuttgart Wind Energy at University of Stuttgart, Allmandring 5B, 70569 Stuttgart, Germany.
E-mail: chen@ifb.uni-stuttgart.de

Abstract. Wind evolution, i.e. the evolution of turbulence structures over time, has become an increasingly interesting topic in recent years. Part of this increasing interest is resulted from the development of lidar-assisted wind turbine control, which requires accurate prediction of wind evolution to avoid unnecessary or even harmful control actions. Moreover, study of wind evolution can deepen our understanding of turbulence characteristics and, for example, could make 4D stochastic wind field simulations possible by integrating wind evolution into 3D simulations. This study compares different machine learning algorithms for the prediction of wind evolution, including three variants of Gaussian process regression, regression tree, support vector machine regression, and shallow neural network. The comparison is done using lidar data measured by a nacelle-mounted lidar, and it focuses on the prediction accuracy and the computational time of models, to provide some insights of the trade-off between these factors. In terms of computational time, both the training time of the final model and the time for hyperparameter optimization are taken into account. Moreover, up-scaling of the computational time with data size is also investigated. It is found that Gaussian process regression provides high prediction accuracy but needs long computational time, increasing significantly with data size. For cases where long computational time might be a critical problem, shallow neural network could be a good alternative thanks to its high training efficiency, although its prediction accuracy is lower than Gaussian process regression. The other methods have no advantages in wind evolution prediction compared to these two methods.

1. Introduction
Taylor’s hypothesis [1], which assumes turbulence structures (eddies) remain unchanged, while the eddies are advected by the mean flow, is commonly used in turbulent wind field simulations and wind data processing. However, this hypothesis is no longer appropriate for the new concept of lidar assisted wind turbine control, i.e. a feed-forward control concept of wind turbines by using a nacelle-mounted lidar to measure the approaching wind field at some distances upwind (see e.g. [2–5]). The reason for that is the control system should only react to the changes in the wind field which can be accurately predicted by the lidar signal, to avoid harmful and unnecessary control actions. However, the wind field measured by the lidar in the upwind direction is not exactly the same as the wind field reaching the wind turbine because the turbulence structure will evolve during this time, which is referred as wind evolution. A possible solution for this problem is to apply an adaptive filter to filter the uncorrelated part of the lidar signal. This requires an accurate prediction of wind evolution. Moreover, the common 3D wind field simulation method [6] is not suitable for modelling the lidar-assisted control system because it assumes...
perfect correlation of turbulence in the longitudinal direction. Hence, different approaches [7,8] are proposed to include the modelling of wind evolution, to enable 4D wind field simulation.

Wind evolution refers to the change of turbulence structures over time. In practice, wind evolution is measured by the longitudinal coherence, i.e. coherence of turbulent velocity at locations separated in the mean direction of the flow (see e.g. [3,4]). And when estimating the longitudinal coherence, the data measured in downstream should be shifted by the travel time of the wind field, corresponding to the evolution time, to match the data measured in upstream.

To predict wind evolution, it is necessary first to model wind evolution using a function with a limited number of parameters which are predictable. Several different wind evolution models have been proposed in literature (see e.g. [4,9,10]). If wind evolution model parameters can be predicted according to variables related to wind field, the coherence can be reproduced by the corresponding wind evolution model. For this purpose, it is attempted to achieve a parameterization model to predict the wind evolution model parameters.

In a previous study [11], a prediction concept and the corresponding workflow were presented for parameterization of a two-parameter wind evolution model adapted from the model suggested by [12]. The parameterization model is a surrogate model trained with on-site measured data. Data from two measurement campaigns, carried on an onshore and an offshore location, was involved in that work. The coherence was estimated with lidar data measured by a nacelle-mounted lidar and the wind evolution model parameters were determined by fitting the estimated coherence to the wind evolution model. The predictors (input variables) for the parameterization model, i.e. the wind statistics and other relevant variables, were derived from data measured with the lidar and an ultrasonic anemometer installed at a similar height. Gaussian process regression (GPR) [13] was used to obtain the parameterization model, with application of the automatic relevance determination squared exponential (ARD-SE) kernel [14] to select suitable predictors. The results have demonstrated the potential of applying GPR for wind evolution prediction because GPR models can provide high prediction accuracy even though the measured data is very noisy.

However, the drawback of standard GPR is that the training time scales cubically with the size of training data [15]. For example, it took more than eight hours to train a GPR model with the ARD-SE kernel using 5-fold cross-validation (explanation see Sect.2.2) without any hyperparameter optimization (explanation see Sect.2.4) for a training data set of 11 dimensions × approx. 10000 observations (Computer specifications: Intel(R) Core(TM) i7-8700 CPU @ 3.20 GHz, usable RAM 15.8 GB).

The importance of computational time depends on the considered application. Long model training time could become problematic if a model must be trained frequently in the application, especially when real-time training is desired. This motivates the present work to investigate the applicability of other common machine learning algorithms for wind evolution prediction on the one hand. On the other hand, some approaches are suggested to increase the training efficiency of GPR, e.g. sparse model approximation with representative subset of training data [15,16]. Therefore, a comparison is carried out among standard GPR and alternative models including: two different variants of sparse Gaussian Process Regression (GPRsd and GPRfic), Regression Tree (RT), Support Vector Machine Regression (SVMR), and Shallow Neural Network (SNN). The comparison focuses on prediction accuracy of the models taking into account their computational time, to provide some insights of the trade-off between these factors. In terms of computational time, not only the training time of the final model is considered in the comparison, but also the time for hyperparameter optimization because it plays an important role in the training of machine learning models. In addition, the increasing trend of computational time with the increase of data size is investigated.
2. Methodology

2.1. Wind evolution

This work is a follow-up research of [11] and is therefore based on its theoretical framework. Section 2.1 and 2.2 briefly introduce the most essential theoretical basic of wind evolution and its prediction concept so that readers can basically understand the main idea of this work without too much effort. More details and discussions can be found in [11].

As mentioned in the introduction, wind evolution is mathematically defined as magnitude-squared coherence $\gamma^2_{ij}$ (hereafter referred to as coherence) between two signals of turbulent velocity $i$ and $j$ measured at two locations separated in the longitudinal direction, with $i$ for the signal measured at upstream point and $j$ for downstream point:

$$\gamma^2_{ij}(f) = \frac{|S_{ij}(f)|^2}{S_{ii}(f)S_{jj}(f)},$$  \hspace{1cm} (1)

where $S_{ii}(f)$ and $S_{jj}(f)$ represent the power-spectral densities (PSDs) of signals $i$ and $j$, respectively, and $S_{ij}(f)$ represents the cross-spectral density between $i$ and $j$. Coherence $\gamma^2_{ij}$ takes values between zero, for no correlation, to unity, for perfect correlation. It must be emphasized that the coherence corresponds to a lagged correlation, which means for calculation of the coherence, the signal $j$ should be shifted by the travel time after which the signal $i$ is expected to arrive at the downstream point.

2.2. Prediction of wind evolution

To predict wind evolution, a function — with as few parameters as possible — is needed to model wind evolution, i.e. wind evolution model. As long as the wind evolution parameters can be predicted, the wind evolution (i.e. the coherence) can be reproduced by the corresponding wind evolution model.

The wind evolution model used in this work (as well as in [11]) is adapted from the two-parameter exponential model suggested in [12]:

$$\gamma^2_{\text{model}} = \exp \left( -\sqrt{a^2 \cdot f_{\text{dless}}^2 + b^2} \right) \quad \text{and} \quad f_{\text{dless}} = f \cdot \Delta t,$$  \hspace{1cm} (2)

where $\gamma^2_{\text{model}}$ is the modeled coherence, $f_{\text{dless}}$ is the dimensionless frequency defined as the multiplication of frequency $f$ and the time shift $\Delta t$ between the two signals $i$ and $j$, corresponding to the travel time. $\Delta t$ is determined by the time lag of the peak of the cross-correlation between the two signals. Both wind evolution model parameters are dimensionless. $a$ is the decay parameter representing decay effect of coherence with increasing frequency, and $b$ is the offset parameter to allow the coherence at zero frequency to be lower than unity.

The wind evolution model parameters $a$ and $b$ are the target variables which are supposed to be predicted by two respective parameterization models. The approach of using machine learning to build a parameterization model is, in principle, to train a selected regression model using observed values (i.e. measured data) of a target variable and some suitable predictors. The observed values of $a$ and $b$ are determined by fitting coherence estimated with measured data to the wind evolution model. In general, predictors could be any relevant variables derived from measured data, such as wind statistics, stability parameters, and relative positions of measurement points. Thus, feature selection needs to be done to select the predictors with sufficient predictive power (accounting for most of the variation of the target values). These works have been accomplished in [11], where more details and discussions can be found.

Model performance is evaluated with root-mean-square error (RMSE):

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{i} (y_i - y_{\text{pred},i})^2},$$  \hspace{1cm} (3)
and coefficient of determination $R^2$ between observed target values and predicted target values:

$$R^2 = 1 - \frac{\sum_i^N (y_i - y_{\text{pred},i})^2}{\sum_i (y_i - \bar{y})^2}.$$  (4)

In both equations, $y$ and $y_{\text{pred}}$ denote observed target values and predicted target values, $\bar{y}$ denotes the average of the observed target values, $N$ denotes the number of observations.

Indeed, RMSE and $R^2$ are equivalent when applied to the same data set. The smaller the RMSE is, the closer the $R^2$ is to unity. RMSE is rather an indicator of the absolute error between the observed and the predicted data, while $R^2$ is a normalized indicator. Moreover, it must be emphasized that according to the definition of Eq.(4), $R^2$ ranges from negative infinity to unity, not literally meaning the square of a value. $R^2$ yields zero if the prediction is simply made with the mean value of the observations. The higher the $R^2$ is, the better the model performs, and a $R^2$ of unity means perfect prediction. A negative $R^2$ indicates that the prediction with the evaluated model is even worse than the prediction directly using the mean value of the observations. In other words, $R^2$ can be understood as taking the prediction using the mean value of the observations as a reference to evaluate the model performance.

The trained model is evaluated with the 5-fold cross-validation: The data is divided into five equally sized subsets; The model validation is done with one subset (also called in-fold observations), while the training is done with the rest four subsets (also called out-of-fold observations): This procedure is repeated five times, each time with a different subset for validation; The predicted target values as well as the RMSE and the $R^2$ of the regression models are computed for in-fold observations using a model trained on out-of-fold observations. Using cross-validation can avoid model over-fitting and reduce the model variance.

2.3. Workflow

This work first attempts to compare the model performance of standard GPR with the five alternative machine learning algorithms. Due to the limitation of the article length, their principles cannot be introduced here in details but can be found in e.g. [13, 17–19]. Some important information must be mentioned: The two variants of sparse GPR are GPR with the subset of data approximation (GPRsd) and GPR with the fully independent conditional approximation (GPRfic), respectively. As its name implies, GPRsd reduces computational complexity by using subsets of data instead of the whole data set to train the model, while GPRfic makes approximation of kernel functions. Shallow neural network specifically refers to a neural network model containing only one to two hidden layers in this work.

New parameterization models are acquired by training these models using the predictors and the observed target values from [11]. The model training is done in two steps: The first step is hyperparameter optimization for the model; The second step is the training of the final model with the optimal hyperparameters. The hyperparameter optimization is done using Bayesian optimization (see Sect.2.4).

In [11], data from two measurement campaigns was investigated. However, only the lidar data from the measurement campaign — LidarComplex [3, 11] are used in this study as an example because the results are generalizable. In LidarComplex, a pulsed lidar with five measurement range gates (at 54.5 m, 81.75 m, 109 m, 136.25 m, and 163.5 m) was installed on the nacelle of a wind turbine at about 95 m, measuring the inflow of the turbine. The selected lidar data set contains 3570 observations. The corresponding predictors for the targets $a$ are the mean value, the standard deviation, the skewness and the kurtosis of wind speed as well as the integral length scale and the travel time approximated by $d/U$ ($d$ is distance between two measurement points and $U$ is the mean wind speed). And the predictors for the target $b$ are the above mentioned four wind statistics, the integral time scale, and the distance between the measurement points.
In other words, the sizes of training data for the parameterization models of the two targets are both seven dimensions (six predictors + the target itself) \times 3570 observations.

The second objective of this work is to illustrate how the computational time of the involved machine learning algorithms increases with the size of training data. To simulate various data amounts, nine data subsets are created by randomly selecting the observations from the original data set without replacement (i.e. each observation can be selected only once). These subsets contain 10\%, 20\%, ..., 90\% of the original data set, respectively. The same training process as above is conducted with these subsets of data to document the computational time. To guarantee reproducible results, the random seed for selecting data is fixed.

The specifications of the computer used in this study are Intel(R) Xeon(R) CPU E5-2630 v3 @ 2.40GHz (2 processors) with an installed memory (RAM) of 128 GB.

2.4. Hyperparameter optimization using Bayesian optimization

In the context of machine learning, a hyperparameter refers to a preset model parameter which modifies the model behaviour in the fitting process. Using proper hyperparameters can improve the performance of a machine learning model. Thus, an optimization process is applied to search for the optimal hyperparameters by minimizing an objective function which can represent the model performance. One of the common used algorithms for hyperparameter optimization is the Bayesian optimization.

The Bayesian optimization attempts to minimize a scalar objective function with bounded independent variables. For the hyperparameter optimization, the objective function is e.g the mean squared error (MSE) of the cross-validation of the trained model and its independent variables are the model hyperparameters to be optimized. A Gaussian process model is used to model the objective function and will be steadily updated at each function evaluation. One of the advantages of the Bayesian optimization is that the combination of the hyperparameters to be evaluated at the next iteration is selected by maximizing an acquisition function rather than random search [20], which can reduce the number of iterations to converge to the minimum of the objective function. One of the common used acquisition functions is expected improvement. It can be understood as the algorithm inferring the hyperparameters which would maximally improve the result of the objective function, i.e. reducing the objective function value in this case. Figure 1 illustrates an example of the Bayesian optimization process.

In this work, the MSE of the 5-fold cross-validation is chosen as the objective function. The applied acquisition function is expected improvement with a mechanism to escape a local objective function minimum and to avoid overexploiting an area. The number of function evaluations for the GPR based models and the SNN models is set to 30, whereas that for the RT models is set to 50 because the objective function minimum cannot be reached within 30 iterations. For the SVMR models, the observed and estimated objective function values do not converge to each other even though the number of iterations is increased to 70. This is no longer increased because the computational time for the optimization is already very long in comparison to the other methods (see Table 1).

3. Results and Discussions

3.1. Comparison of prediction accuracy and computational time

Table 1 summarizes the main outcomes of this study, including the optimal hyperparameters for each model determined using Bayesian optimization, the prediction RMSE and $R^2$, and the computational time of hyperparameter optimization and training of the final model for the complete data set.

Figure 2 shows the comparison of the prediction $R^2$ (curve corresponding to the right y-axis) of the six models and their computational time for hyperparameter optimization and training of the final model (bars corresponding to the left y-axis). As mentioned above, the computational
Table 1. Comparison of the model performance among the six machine learning algorithms.

| Target | Model     | Optimal Hyperparameters                      | RMSE | $R^2$ | Time [s] |
|--------|-----------|----------------------------------------------|------|------|---------|
|        | GPR       | basis function linear                        | 0.3933 | 0.70 | 17270   |
|        |           | kernel function ARD exp.                     | 1.4553 $\times 10^{-4}$ |      | 261     |
|        | GPRsd     | basis function constant                      | 0.4431 | 0.62 | 8825    |
|        |           | kernel function ARD exp.                     | 8.9739 $\times 10^{-4}$ |      | 53.5    |
|        | GPRfic    | basis function none                          | 0.4940 | 0.53 | 18970   |
|        |           | kernel function ARD exp.                     | 0.2467 |      | 150     |
|        | RT        | min. leaf size 19                           | 6.428  | 0.20 | 42.5    |
|        |           | max. splits 2619                            |       |      | 0.0241  |
|        | SVMR*     | box constraint 996.93                        | 5.891  | 0.33 | 11399   |
|        |           | kernel function Gaussian                     |       |      | 6.62    |
|        |           | kernel scale 0.41568 $\times 10^{-3}$        |       |      |         |
|        | SNN       | training function CGB [26, 24]               | 0.5605 | 0.39 | 2016    |
|        |           | hidden layer size                            |       |      | 13.6    |
|        | GPR       | basis function linear                        | 0.0487 | 0.68 | 16909   |
|        |           | kernel function ARD exp.                     | 1.4196 $\times 10^{-4}$ |      | 289     |
|        | GPRsd     | basis function linear                        | 0.0529 | 0.62 | 7124    |
|        |           | kernel function ARD rat. quad.              | 4.3822 $\times 10^{-3}$ |      | 76.4    |
|        | GPRfic    | basis function constant                      | 0.0579 | 0.55 | 13315   |
|        |           | kernel function ARD rat. quad.              | 2.9344 $\times 10^{-2}$ |      | 177     |
|        | RT        | min. leaf size 30                           | 0.0692 | 0.35 | 36.2    |
|        |           | max. splits 1746                            |       |      | 0.0182  |
|        | SVMR*     | box constraint 51.573                       | 0.0695 | 0.34 | 5938    |
|        |           | kernel function Gaussian                     |       |      | 59.9    |
|        |           | kernel scale 6.6491 $\times 10^{-3}$         |       |      |         |
|        | SNN       | training function CGP [7, 5]                 | 0.0686 | 0.36 | 2223    |
|        |           | hidden layer size                            |       |      | 6.75    |

Abbr.: optim. = optimization, ARD exp. = ARD exponential, ARD rat. quad. = ARD rational quadratic, CGB = Conjugate Gradient with Powell/Beale Restarts, CGP = Polak-Ribiére Conjugate Gradient.

*For the SVMR models, the optimal hyperparameters are selected based on the best observed points instead of the best estimated points of Bayesian optimization as for the other models.
Figure 1. Illustration of the Bayesian optimization process. (a)–(e) the Gaussian process model of the objective function at the 4\textsuperscript{th}, 5\textsuperscript{th}, 10\textsuperscript{th}, 20\textsuperscript{th}, and 50\textsuperscript{th} function evaluation. (f) The plot of the minimum objective value vs. the function evaluations.

time of the model very much depends on the hyperparameters even for the same model, and thus the specific value of the time is not very important but the order of magnitude. The left y-axis is set to be logarithmic to show the difference of the orders of magnitude of the computational time.

Figure 2. Comparison of the prediction $R^2$ of the six models and their computational time. (a) models for the target $a$. (b) models for the target $b$.

Overall, these six models can be categorized into three groups: 1) models having high prediction accuracy but long computational time, i.e. the three GPR based models; 2) models having medium prediction accuracy and medium computational time, i.e. the SVMR model and SNN model; 3) model having low prediction accuracy but extremely less computational time, i.e. the RT model.

First, compare the three GPR based models. As expected, the models of GPRsd and GPRfic show lower values of $R^2$ in comparison to the standard GPR model since both algorithms make approximation when estimating the hyperparameters of the GPR model. However, in this case, GPRsd outperforms GPRfic with a higher value of $R^2$ and less computational times for the optimization and the final model training. Therefore, it is not necessary to consider GPRfic in
the further analysis.

Then, compare the models of SVMR and SNN. It can be observed that for the prediction of the target variable $a$, the SNN model shows a higher value of $R^2$ and less optimization time but a slightly longer final model training time than that of SVMR. For the prediction of the target variable $b$, the SVMR model and the SNN model show similar prediction accuracy, but the computational time of the SNN model is obviously less. In total, the performance of SNN is slightly better than that of SVMR. However, the observed and estimated objective function values in the hyperparameter optimization of the SVMR model do not converge to each other even after 70 iterations, which makes the SVMR models less plausible. Thus, SVMR is not involved in further discussions.

Finally, it can be noted that the RT model shows a low value of $R^2$ for the prediction of $a$, but surprisingly, its prediction for $b$ is as good as that of SVMR and SNN. Considering that the computational time of a RT model is generally two to three orders of magnitude less than that of the other algorithms, RT is still kept as an alternative model for further analysis.

Based on these consideration, only four models are further analyzed in the following part: GPR, GPRsd, SNN, and RT. Figure 3 illustrates the scatter plots of the target variables $a$ and $b$ determined with the measured data versus that predicted with these four models, respectively. Overall, the prediction of $b$ is generally more difficult and the data points in Fig.3 (e)–(h) are more scattered. An interesting phenomenon can be observed in Fig.3 (c) and (g) that the target values predicted by the RT models are restricted within certain value ranges. This probably results from the principle of the RT. In the algorithm of RT, the data is split into two parts according to some criteria, and both parts are further split into two parts, respectively. This kind of data split is continued until the number of samples has reached the minimum leaf size. Therefore, a RT model is generally less sensitive to the extreme values in the data when the minimum leaf size is not set to be too small. Moreover, the prediction of $b$ by the RT and SNN models is probably not satisfactory enough since the data points in Fig.3 (g) and (h) do not show a clear trend following the one-to-one curve (black dashed line).

Figure 3. Comparison of wind evolution model parameters determined with measured data and that predicted with different models (with subscript "pred"). (a) and (e): GPR. (b) and (f): GPRsd. (c) and (g): RT. (d) and (h): SNN.

To intuitively display the different prediction results of the four models, the corresponding
predicted coherence curves are visualized in the frequency domain by combining both predicted wind evolution model parameters using Eq.(2). Figure 4 shows a case study example on 08 Dec. 2013 at 13:00–13:30. Figure 4 (a) and (b) illustrate the coherence curves estimated with the measured data, the fitted curves of the wind evolution model, and the predicted curves as well as the corresponding 95% confidence interval by the models for two different measurement separations, i.e. between the first range gate $R_1$ and the second one $R_2$, and the first one $R_1$ and the fifth one $R_5$, respectively. In this case, only the GPR based models have confidence interval because they are probabilistic models, whereas the RT models and SNN models are not. Figure 4 (c) shows the corresponding time series plot of the lidar wind speed measured at the relevant three range gates. It can be observed that the GPR models and the GPRsd models have a very good prediction for both wind evolution model parameters. Their predicted curves completely match the fitted one and the corresponding 95% confidence interval curves are too narrow to be distinguished. Nevertheless, the RT models and SNN models can predict the decay parameter $a$ quite well, but their predictions of the offset parameter $b$ are not good enough, as observed from Fig.3. The predicted curves by the RT models and SNN models still show a good match with the fitted one in Fig.4 (a) because there is no offset of the coherence between a small separation, whereas a clear discrepancy can be observed in the low frequency range in Fig.4 (b), which results from the deviation of the prediction of $b$.

![Figure 4. Example predicted coherence curves by the four models for two different measurement separations: (a) $R_1$ vs. $R_2$, $d = 27.25$ m and (b) $R_1$ vs. $R_5$, $d = 109$ m. (c) Lidar wind speed of the example data block. Mean reference wind speed: 10.2 m s$^{-1}$. Data source: LidarComplex. Date and time: 08 Dec. 2013 13:00-13:30.](image-url)
dramatically increases with the number of observations as expected. Compared to that, the computational time of the GPRsd models (applied only to the data subsets with more than 2000 observations) confirms that the application of the subset of data approximation for GPR can significantly reduce the computational time for a large data set. The computational time of the SNN models seems to increase linearly with data size. The computational time of the RT models is completely negligible compared to that of the other models.

Moreover, a special issue is observed in Fig. 5 (a). The fourth data point on the blue curve, i.e., the prediction of $a$ using GPR trained with 40% of the data, is found to be a special case having much longer computational time than it is supposed to. The reason for that is that one of the evaluations in the Bayesian optimization took almost 6000 s, which is quite abnormal, whereas the other 29 evaluations took only about 1500 s in total. The corresponding optimization process was repeated with the same random seed to avoid accidental factors (like special issues of the computer). The new result turned out to be similar, which means the abnormally long computational time of that evaluation is not by accident. The extreme long training time might be related to the specific combination of hyperparameters in that evaluation.

![Graph showing computational time vs number of observations for different models](image_url)

**Figure 5.** Comparison of the up-scaling of the computational time of (a) hyperparameter optimization and (b) training of the final model with increasing data size.

4. Conclusions

This study aims to compare different machine learning algorithms for prediction of the wind evolution model parameters. The comparison not only focuses on the prediction accuracy of the model but also the computational time to provide some insights of the trade-off between them. The machine learning algorithms involved in the comparison are standard Gaussian Process Regression (GPR), two different variants of sparse Gaussian Process Regression (GPRsd and GPRfic), Regression Tree (RT), Support Vector Machine Regression (SVMR), and Shallow Neural Network (SNN).

The model training is done in two steps: The first step is hyperparameter optimization using Bayesian optimization; The second step is the training of the final model with the optimal hyperparameters. Both computational time are taken into account in the comparison. Moreover, the up-scaling of computational time with data size is also investigated. The training data has been processed in a previous study [11].
To intuitively display the results, Table 2 shows a simple ranking of the six machine learning algorithms according to their prediction accuracy and computational time.

The results show that these machine learning algorithms can be categorized into three groups. The first group is the models having high prediction accuracy but long computational time, i.e. the three GPR based models. Among them, the standard GPR model performs the best but takes the longest computational time. Moreover, its computational time increases dramatically with the increase of the training data size. Thus, standard GPR is worth considering only for a small data size. For cases where the data size is relatively large, but high prediction accuracy is still desired, GPRsd is preferable rather than GPRfic because the former can significantly reduce the training time of the final model while maintaining good prediction accuracy. However, GPR based models generally need long time for hyperparameter optimization.

The second group is the models having medium prediction accuracy and medium computational time, i.e. the SVMR model and the SNN model. The SNN model significantly outperforms the SVMR model with higher prediction accuracy and much less computational time, especially the time for hyperparameter optimization. Therefore, SNN will be very advantageous when processing large amounts of data, although its prediction accuracy is not as high as the GPR based methods.

The third group is the model having low prediction accuracy but extremely less computational time, i.e. the RT model. Although a RT model can be trained extremely fast, its prediction accuracy is quite low. Hence, it is not suggested to use a RT model for the prediction of wind evolution. But with extremely high training efficiency, RT could be used as a tool for a preliminary data analysis to acquire an overview of some basic correlations among data.

To summarize, three models are recommended for prediction of wind evolution, i.e the standard Gaussian process regression, the Gaussian process regression with the subset of data approximation, and shallow neural network. Standard Gaussian process regression is recommended only for cases where very high prediction accuracy is desired given a small data size. For cases where the data size is relatively large, but high prediction accuracy is still required, Gaussian process regression with the subset of data approximation could be a good choice. For cases where model training efficiency is more critical than prediction accuracy, especially given a very large amount of data, shallow neural network might be a good alternative. The other models have no advantages compared to these three.

Acknowledgments
This research is carried out within the framework of Joint Graduate Research Training Group Windy Cities supported by Ministry of Baden-Wuerttemberg for Sciences, Research and Arts. Some data used in this work is generated in the project LidarComplex (code number: 0325519A) funded by the German Federal Ministry for Economic Affairs and Energy (BMWi).
References

[1] Taylor G I 1938 Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences 164 476–490 ISSN 1364-5021

[2] Bossanyi E A, Kumar A and Hugues-Salas O 2014 Journal of Physics: Conference Series 555 012011 ISSN 1742-6588

[3] Schlipf D 2015 Lidar-assisted control concepts for wind turbines Dissertation

[4] Simley E J and Pao L Y 2015 Wind Speed Preview Measurement and Estimation for Feedforward Control of Wind Turbines (Ann Arbor: ProQuest Dissertations & Theses) ISBN 9781339036748

[5] Simley E, Fürst H, Haizmann F and Schlipf D 2018 Remote Sensing 10 863 ISSN 2072-4292

[6] Paul S Veers 1988 Three-dimensional wind simulation Tech. rep. Sandia National Laboratories Albuquerque, New Mexico and Livermore, California

[7] Bossanyi E 2013 IET Renewable Power Generation 7 321–329 ISSN 1752-1416

[8] Laks J, Simley E and Pao L 2013 2013 American Control Conference (IEEE) pp 3673–3679 ISBN 978-1-4799-0178-4

[9] Panofsky H A and Mizuno T 1975 Boundary-Layer Meteorology 9 247–256 ISSN 0006-8314

[10] Kristensen L 1979 Boundary-Layer Meteorology 16 145–153 ISSN 0006-8314

[11] Chen Y, Schlipf D and Cheng P W 2020 Wind Energ. Sci. Discuss., Preprint

[12] Simley E and Pao L Y 2015 2015 American Control Conference (ACC) (IEEE) pp 3708–3714 ISBN 978-1-4799-8684-2

[13] Rasmussen C E and Williams C K I 2006 Gaussian processes for machine learning Adaptive computation and machine learning (Cambridge, Mass. and London: MIT) ISBN 026218253X

[14] Duvenaud D 2014 Automatic model construction with Gaussian processes Apollo - university of cambridge repository

[15] Schreiter J, Nguyen-Tuong D and Toussaint M 2016 Neurocomputing 192 29–37 ISSN 09252312

[16] Das S, Roy S and Sambasivan R 2018 Big Data Research 14 12–26 ISSN 22145796

[17] Breiman L, Friedman J, Stone C J and Olshen R 1984 Classification and regression trees (Boca Raton, FL: CRC Press) ISBN 9780412048418

[18] Caudill M and Butler C 1992 Understanding neural networks: Computer explorations (Cambridge, MA: The MIT Press) ISBN 978-0262530996

[19] Vapnik V N 1995 The Nature of Statistical Learning Theory (New York, NY: Springer New York) ISBN 978-1-4757-2440-0

[20] Snoek J, Larochelle H and Adams R P 2012 Practical bayesian optimization of machine learning algorithms URL http://arxiv.org/pdf/1206.2944v2