Journal of Science and Technology in Civil Engineering, HUCE, 2023, 17 (4): 95–108

DEVELOPMENT OF A PROCEDURE FOR MODEL SELECTION IN MACHINE LEARNING-BASED METHODS FOR STRUCTURAL DAMAGE DETECTION

Ha Manh Hung[a], Nguyen Long[b], Dang Viet Hung[a,*], Dinh Van Thuat[a]

[a] *Faculty of Building and Industrial Construction, Hanoi University of Civil Engineering, 55 Giai Phong road, Hai Ba Trung district, Hanoi, Vietnam*
[b] *Institute of Planning and Transportation Engineering, Hanoi University of Civil Engineering, 55 Giai Phong road, Hai Ba Trung district, Hanoi, Vietnam*

Article history:
Received 05/8/2023, Revised 05/9/2023, Accepted 18/9/2023

Abstract

Output-only methods based on machine/deep-learning algorithms are highly practical approaches for timely detecting potential damages in civil structures as they directly employ measured vibration signals but do not require exact knowledge of input loading nor the service interruption for manual inspection. However, there is no one-size-fits-all model that is optimal for all problems in different perspectives; hence, it is necessary to discover the advantages as well as drawbacks of different models, then leverage these understandings to select the most appropriate model for specific structures in reality. Therefore, this study develops a framework that facilitate the model selection by extensively comparing various machine learning-based methods ranging from relatively simple ones such as Naïve Bayes to complex ones such as the extreme boosting tree-based ensemble model. The framework can provide comparison results include various aspects such as model complexity, training time, detection accuracy, and inference time.

Keywords: structural health monitoring; machine learning; output-only; sensor signal.

https://doi.org/10.31814/stce.huce2023-17(4)-08

1. Introduction

Structures' vibration signal is a rich information source that can be measured directly in a straightforward fashion when the structures are under normal service, being subjected to random ordinary loads. There have been abundant methods developed for vibration-structural health monitoring, ranging from relatively simple statistical methods to sophisticated optimization algorithms and powerful artificial intelligence models.

The output-only structural damage detection (SDD) problem could be reformulated as an optimization problem in which one minimizes a fitness function measuring the deviation between measured data and computation data. The solutions of the optimization contain information about damage locations and damage levels. After that, one can resort to various heuristic optimization algorithms to solve the optimization problem, such as Colliding Bodies Optimization [1], Charged System Search [2], Particle Swarm Strategy [3], etc. However, these optimization methods usually require an additional pre-processing step to extract modal characteristics from vibration data such as eigenfrequency, mode shape, and modal assurance criteria [4].

Instead of using modal features [5], it is possible to use features in other domains, such as statistical, time, and frequency domains, to perform structural damage detection. It is noted that extracting these features is easier than modal features. After that, a number of machine learning (ML) algorithms

[*] Corresponding author. *E-mail address:* hungdv@huce.edu.vn (Hung, D. V.)

could be leveraged to perform damage detection tasks. In the early period of artificial intelligence development, Naïve-Bayes was a simple, practical, and interpretable method that could quickly provide SDD results, as demonstrated in [6]. However, a major limitation of this method is the assumption of feature independence, i.e., the features extracted from the same vibration signals are completely uncorrelated. Later, another ML algorithm, namely the Support vector machine, was proved to provide better performance than other simple methods such as Naïve-Bayes and regression classifier. Zhou et al. [7] used a support vector machine algorithm to perform building SDD and achieved a high accuracy of more than 90%. Furthermore, the authors proposed to employ multiple support vector machine (SVM) models to work with multisensory signals, helping increase further the model's robustness against sensor failures. SVM was also successfully utilized by other authors, such as Diao et al. [8], even with noisy signals. For many researchers, it is of great importance to understand the contribution of every feature to the final model performance. That is why the interpretable Decision Tree algorithm is widely favorited by many researchers; for example, Karbassi et al. [9] used DT for detecting damage in regular reinforced concrete buildings, Imad et al. [10] employed DT in monitoring the working conditions of the complex wind turbine structures under stochastic wind loads. In order to detect potential post-earthquake damage in braced-frame structures, Salkhordeh et al. [11] proposed a two-stage SDD method that first extracts dynamic features such as drift, correlation, and energy ratio, then utilizes the DT algorithm as a classifier to identify the structure's health status. Another practical machine learning algorithm favored in SDD is the k-nearest neighbor algorithm, for example, the work of Ghiasi et al. [12], for detecting damage due to corrosion in steel railway bridges. In order to improve further the SDD performance of tree-based methods, Zhou et al. [13] suggested using an ensemble of trees, i.e., the random forest model, along with a data fusion strategy. The authors postulated that the proposed method outperforms other counterparts, including conventional random forest (RF) and SVM methods. Other tree-boosting models that could provide competing performance with RF are Adaboost and LightGBM [14]. Recently, a widely considered one of the state-of-the-art machine learning algorithms, i.e., XGBoost, has been favored by various authors when performing structural health monitoring as the works of Dong et al. [15] for concrete structures, Wang et al. [16] for timber buildings, Zhang et al. [17] for bridge structures. Another appealing strategy is to directly use raw high dimensional vibration data and put them into a data-driven model to perform SDD tasks as done by Hung [18, 19].

Given a large number of machine learning-based SHM models, this study aims to development a procedure that could perform an extensive comparison to highlight the advantages and inconveniences of each model, thus, providing a useful basis for structural engineers in selecting an appropriate model for specific structures with available computational resources.

2. Procedure for Model Selection in Machine Learning-Based Methods for Structural Damage Detection

2.1. Overall procedure for model selection

The proposed procedure for selecting an appropriate ML model for a specific SDD problem is schematically presented in Fig. 1, consisting of six sequential steps. The first step, namely data management, aims to properly manage SDD by indexing data by user-defined rules, for example, by collection time, by size, by group, etc. In addition, it is critical to label data with corresponding structures' operational states. After that, the data is stored in an environment where the ML model can easily be accessed later. The second step involves some sub-steps necessary for an effective training process, such as data standardization and data splitting. A widely recognized splitting method is K-fold cross-validation, which can be used to reduce the overfitting problems of ML models. In the third step, various ML models are defined and configured by either using open-source libraries such as scikit-learn, XGBoost, or even some user-defined models. Following that, in the fourth step, key parameters of each ML model that have a large impact on the model performance and computation time are defined. A hyper parameter optimization algorithm, such as Bayesian optimization, grid search, and random search, is then leveraged to determine the optimal set of parameters. Subsequently, the fine-tuned model will be trained and then serve to assess the structures' states. Besides, the metrics reflecting the time complexity, like the training time and inference time, are also recorded in this step. Finally, the performance and efficiency of different ML models are placed side by side to compare and discuss, providing useful insights and helping the users determine the most suitable ML problem for the investigated SDD problem.

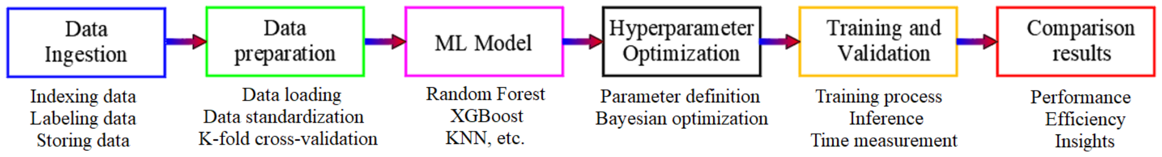


Figure 1. Schematic representation of the model selection procedure for SDD problems

2.2. *Machine Learning-Based Methods for Structural Damage Detection*

As machine learning models is the heart of the model selection procedure, the following section will briefly describe the theoretical foundation and main intuition of the most popular and advanced Machine Learning models, providing necessary understanding for users before applying these models.

a. Logistic regression

The main idea of Logistic Regression (LR) is to calculate the probability of every class by applying a logistic function to a latent variable z obtained from a linear combination of input variables. Mathematically, the latent variable z is calculated by:

$$\begin{aligned} z(y=1) &= w_{10} + w_{11}x_1 + \ldots + w_{1n}x_n = W_1^T.X \\ z(y=2) &= w_{20} + w_{21}x_1 + \ldots + w_{2n}x_n = W_2^T.X \end{aligned} \tag{1}$$

Then the probability output is:

$$\begin{aligned} P(y=1) &= \frac{e^{z(y=1)}}{e^{z(y=1)} + e^{z(y=2)}} \\ P(y=2) &= \frac{e^{z(y=2)}}{e^{z(y=1)} + e^{z(y=2)}} \end{aligned} \tag{2}$$

In which $w_{i,j}$ are the model parameters, $i = 1, 2$ and $j = 1, \ldots, n$. Then, the predicted class is the one with the highest probability. For multi-label classification problems such as detecting the damage element among multiple structural elements, the logistic regression can be extended into the softmax regression with the probability of a class k is:

$$P(y=k) = \frac{e^{z(y=k)}}{\sum_{i=1}^{C} e^{z(y=i)}} \tag{3}$$

with $z(y=k) = w_{k0} + w_{k1}x_1 + \ldots + w_{kn}x_n = W_k^T.X$.

The parameters W can be determined by maximizing the Maximum Likelihood Estimation loss function using one of some popular optimization algorithms lbfgs, bilinear, newton-sage, etc.

b. Naïve Bayes

The Naïve Bayes method is a fast and simple method based on Bayes' theorem and under a strong (naïve) assumption that features are completely independent. Let's denote $P(y)$ and $P(x_i|y)$ being class prior probability and features' conditional probability which can be determined from the training dataset. After that, given a feature vector $X = (x_1, \ldots, x_n)$, the posterior probability of each class is calculated via the Bayes formula as follows:

$$P(y = k|X) = \frac{p(k)\,p(X|k)}{p(X)} = \frac{p(k)\,p(x_1|k)\ldots p(x_n|k)}{p(x_1)\ldots p(x_n)} \tag{4}$$

Then the class having the highest posterior probability is selected as the predicted result.

$$y_{pred} = \underset{k\in\{1,..,C\}}{\operatorname{argmax}} P(k|X) \tag{5}$$

c. Support Vector Machine

The key idea of the SVM classifier is to first use a kernel function to project data samples to equal or higher dimensional spaces that are potentially easier to classify data samples than in the original space; second, to seek a hyperplane in the new space to clearly separate data samples. The desired hyperplane is that the distance, a.k.a margin, from the nearest data points (support vectors) of all classes to this hyperplane is the largest. Mathematically, the SVM classifier can be described as follows. The transformation operator is $\Omega(x)$: $R^D \to R^M$, then the function of the linear hyperplane in the transformed space is:

$$W^T\Omega(x) + b = 0 \tag{6}$$

The margin between any data point x to the hyperplane is:

$$d(x) = \frac{|W^T\Omega(x) + b|}{||W||_2} \tag{7}$$

where $||W||_2$ is the Euclidean norm, a.k.a L2 norm, which is calculated by

$$||W||_2 = \sqrt{w_1^2 + \ldots + w_M^2} \tag{8}$$

After that, the SVM classifier can be reformulated as a conventional constrained optimization problem that seeks W and b to maximize the margin between data points to the hyperplane under the constraint that the classes of all training data points are correctly determined. Next, various optimization algorithms, such as quadratic programming, gradient-based methods, etc., can be employed to solve this problem. For some problems, a non-linear version of SVM can be assorted to achieve better performance. On the other hand, SVM uses a kernel function to compute the distance between two data points in higher dimensional space without explicitly performing the transformation, helping significantly boost the computational time.

d. K-Nearest Neighbor

K-Nearest Neighbor (KNN) is one of the most simple ML algorithms that are easy to understand and implement. Its underlying idea is based on the voting mechanism. Specifically, given a new unlabeled data point, KNN first identifies the K-closest labeled data points in the training dataset of that new data point. Next, the most popular class among these K-labeled data is assigned to the new data point under consideration. To measure the distance among data points, some popular distances could be used, such as Euclidean distance, Manhattan distance, etc. It can be seen that KNN does not require any training process; it simply stores the labeled training data. Thus, it is classified into the lazy-learner methods. However, when the size of the training data is big and/or the data is high-dimensional, the computational effort could be highly expensive because it is necessary to compute the distances between new data with all training data points.

e. Decision Tree

Decision tree algorithm is a recursive algorithm that partition data into two non-overlapping group based on values of a specific feature. The partition is carried out until a stopping condition such as the maximum number of splitting steps, the minimum number of samples in a leaf, etc. is met. The original data correspond to the Root node at the top of the tree, each feature used to split data is called decision node, and associated sub-dataset correspond to a divided branch is called sub-tree. And the final group, at the bottom of each tree branch is named leaf node. Give a new data sample, based on values of each decision nodes, it will be classified into one of leaf nodes and assigned to the corresponding output class. At each decision node, the popular way to perform the split is the information gain technique which attempt to minimize the randomness of samples within groups. The randomness of data is measuring by using the entropy metric which is mathematically expressed by:

$$Entropy\,(D) = -\sum_{k=1}^{C} p\,(k) \log_2 p\,(k) \tag{9}$$

where $p(k)$ is the percentage of data belonging to a class k, the entropy value ranges from 0 to 1 if the data is perfectly homogeneous, i.e., all samples belong to one group, and there is no randomness, i.e., Entropy equals 0. In contrast, if the data is uniformly distributed among groups, the randomness is at its highest, i.e., Entropy equals 1. Next, the information gain is derived from measuring the difference in Entropy before and after performing the splitting action with a selected feature.

$$IG\,(D, f) = Entropy\,(D) - \sum_{v \in f} \frac{|D_v|}{|D|} Entropy\,(D_v) \tag{10}$$

where v is a value of feature f, $|D|$ is the total number of data, $|D_v|$ is the number of data with feature f equal to v. The feature that yields the highest information gain will be selected as the decision node.

f. Random Forest

In order to improve further the classification performance and robustness of tree-based models, Breiman [10] developed the Random Forest algorithm which utilizes multiple decision tree models in a bagging fashion for predicting results, then the final output is obtained via a majority-voting mechanism. The underlying idea of Random Forest is that each individual has its own strength and weakness due to its specific configuration, different hyperparameters, selected features, etc. Thus, combining multiple DT models will increase the diversity, bring additional complementary strengths, reduce the bias and improve the generalization improvement. Specifically, the bagging technique involves dividing the entire data and set of features into different subsets with placement, i.e., data points could be reused multiple times. Each subset serves to train an uncorrelated decision tree. The realization steps of Random Forest could be summarized as follows:

- Define a forest comprised of N uncorrelated decision tree models;
- Sample Bagging: Randomly divide original data into N sub-datasets with replacement;
- Feature bagging; randomly select N subset of features;
- Training all N decision tree models;
- Aggregating classification outputs of DT models by majority-voting.

g. Adaboost

Apart from the bagging technique, another popular technique to combine multiple ML models is boosting. Boosting refers to a family of algorithms which converts multiple weak learners to a single strong learner. Yoav Freund and Robert Schapire proposed to combine DT models via a weighted sum whose weights are adaptively updated in a sequential manner (Fig. 2). This method is called the adaptive boosting technique (Adaboost). Note that, unlike the bagging method, in the boosting method, each DT mode is trained with the entire training data. Formally, let's denote $\boldsymbol{DT_k(X_i)}$ as base models, with $\boldsymbol{k = 1, \ldots, N_{model}}$, $\boldsymbol{N_{model}}$ is the total number of models, $\boldsymbol{\alpha_k}$ is the weight assigned to $\boldsymbol{DT_k(X_i)}$, $\boldsymbol{X_i}$ is a data sample with $\boldsymbol{i = 1, \ldots, N_{data}}$, $\boldsymbol{N_{data}}$ is the total number of samples. Initially, the weights are set uniformly to $\boldsymbol{\alpha_i = 1/N_{model}}$. Considering model $\boldsymbol{k}$, an error made by $\boldsymbol{DT_k}$ on $\boldsymbol{X_i}$ is denoted by $\boldsymbol{e_{i,k}}$. Thus, the total error by $\boldsymbol{DT_k}$ is:

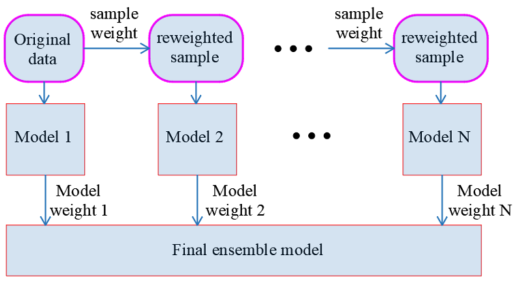


Figure 2. Representation of Adaboost

$$IGe_k = \sum_{i=1}^{N_{data}} e_{ki} \tag{11}$$

Next, the weight of DT_k is computed as follows:

$$\alpha_k = \frac{e_k}{1 - e_k} \tag{12}$$

On the other hand, the Adaboost algorithm, also assigns weights to data samples and pay more attention to those that are misclassified by previous models. The weight of each data sample i is updated for the next model $k+1$, as below:

$$w_{k+1,i} = \frac{w_{k,i}\alpha_k}{\sum_{i=1}^{N_{data}} w_{k,i}\alpha_k} \tag{13}$$

h. LightGBM

Another boosting strategy is gradient boosting, in which a sequence of DT models is successively designed and trained. At each iteration, the prediction mistakes, a.k.a residual errors made by previous models, are calculated, then a new DT model is derived from previous ones to predict these residual errors. Hence, by summing the output predictions of previous DT models with the residual error prediction by the new DT model, obtained results will approach the actual output. With gradient boosting, the parameters of the new DT model, such as split points, split variables, and leaf values, are determined by using the gradient, a.k.a; the rate of change of the loss function with respect to the model's predicted values. Based on the principle of gradient boosting, a highly efficient algorithm, namely, LightGBM is developed, which grows DT models in a leaf-wise fashion, i.e., the new tree is obtained by including a new leaf to its predecessor. A graphical explanation of LightGBM is illustrated in Fig. 3. It can be seen that starting with model 1, one will create a model 2 based on a critical feature (in red), and the results at iteration 2 are obtained by aggregating outputs of both models 1 and 2. Next, at iteration 3, model 3 is devised to improve the performance of model 2, and the results at iteration 3 are calculated based on the outputs of all three models. The above procedure is repeated until one of the convergence criteria is reached. Furthermore, LightGBM also leverages the gradient information to identify samples difficult to predict, i.e., those with high-gradient samples. By doing so, LightGBM could reduce the training size by focusing more on samples with high gradients while only utilizing a small portion of samples with low-gradient values, thus shortening convergence speed and potentially improving accuracy.

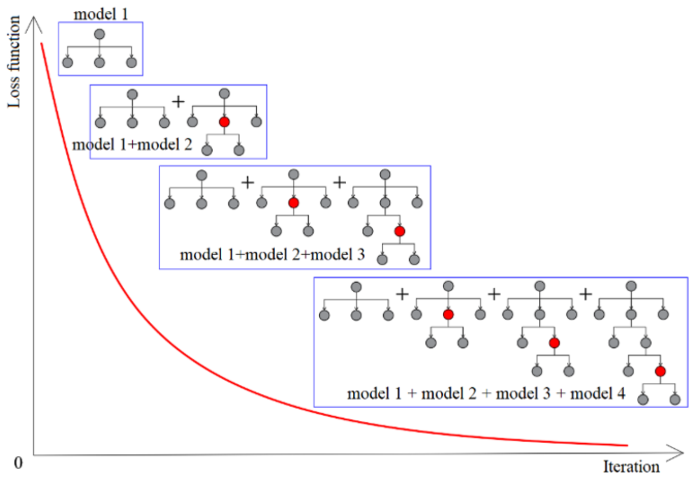


Figure 3. Representation of the LightGBM algorithm

i. XGBoost

The last ML algorithm of interest in this study is Extreme Gradient Boosting (XGBoost) which is a favorite of various authors thanks to its high performance and efficiency; however, the theoretical background of the method is fairly more complicated than those of others, and it has a number of user-defined hyperparameters. Therefore, it is necessary to understand the algorithm mechanism to better apply this algorithm to specific problems. Different from LightGBM, XGBoost uses a level-wise tree growth approach to engineer new trees in the ensemble. This strategy expands all nodes at a considered level before going down to a deeper level. The growth of each branch is stopped when one of the predefined criteria is met, such as the maximum depth of the tree, a minimum number of samples per leaf, etc. To measure the improvement of the model, XGBoost elaborates an objective function including two components: the loss function evaluating the prediction error and a regularization term for controlling the model complexity and mitigating the overfitting risk, as follows:

$$obj_t = \sum_{i=1}^{n} L\left(y_i, y_i^{pred}\right) + \Omega\left(f_t\right) = \sum_{i=1}^{n} \left(y_i - \left(y_i^{t-1} + f_t\left(x_i\right)\right)^2\right) + \Omega\left(f_t\right) \tag{14}$$

In which n is the number of samples, T is the number of trees, y_i, y_i^{pred} are the actual and predicted outputs of data sample i, y_i^{t-1} and $y_i^t = f_t\left(x_i\right)$ are prediction values at previous tree level and current level $t-1$ and t, respectively. $L\left(y_i, y_i^{pred}\right)$ is the loss function, and $\Omega\left(f_t\right)$ is a regularization term of tree t. On the other hand, in addition to gradient information, XGBoost employs the hessian of loss function w.r.t predicted values to update the new DT model.

$$\begin{aligned} g_t(x_i) &= \frac{\partial L\left(y_i, y_i^{pred}\right)}{\partial f_t(x_i)} \\ h_t(x_i) &= \frac{\partial^2 L\left(y_i, y_i^{pred}\right)}{\partial f_t(x_i)^2} \end{aligned} \tag{15}$$

By using the second order Taylor expression, the objective function is rewritten in function of the gradient and hessian information as below:

$$\begin{aligned} obj_t &= \sum_{i=1}^{n} L\left(y_i, y_i^{pred}\right) + \Omega(f_t) = \sum_{i=1}^{n} \left(y_i - \left(y_i^{t-1} + f_t(x_i)\right)^2\right) + \Omega(f_t) \\ obj_t &= \sum_{t=1}^{T} \left(G_t \alpha_t + \frac{1}{2}(H_t + \lambda)\alpha_t^2\right) + \gamma T \end{aligned} \tag{16}$$

with

$$G_t = \sum_{i \in 1,\ldots,n} g_t(x_i), \quad H_t = \sum_{i \in 1,\ldots,n} h_t(x_i), \quad \alpha_t = -\frac{G_t}{H_t + \lambda} \tag{17}$$

where λ and γ are hyperparameters, T is the number of leaves.

3. Application example

3.1. Problem description

In this subsection, the experimental database from an experiment with a full-size reinforced concrete column at Missouri University of Science and Technology [20] is used to validate the ML models under investigation. The column has a circular section with a diameter of 610 mm and a height of around 3660 mm as shown in Fig. 4. The columns were subjected to reversed-cyclic loads with various amplitudes, leading to 12 different damage scenarios, as listed in Table 1. To monitor the structural health of the column, smart aggregate sensors were used as transducers to collect and send waves to a computer. The sensors were preinstalled at the column top and bottom locations before casting concrete. It is expected that damaged areas (crack, spalling, etc.) will reduce the transmission energy of the signal waves when the column is subjected to external load; thus, the transmission energy will scale linearly with the damage severity of the columns.

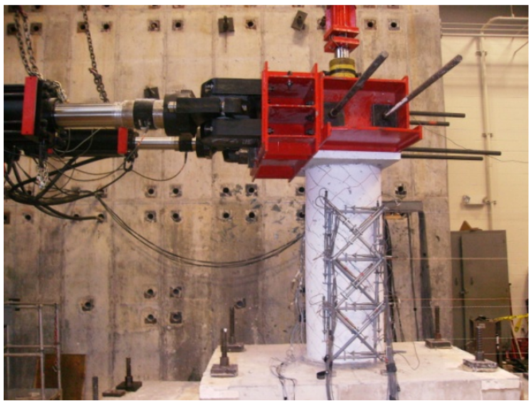


Figure 4. Experiment with a full-size reinforced concrete circular column at Missouri University of Science and Technology [20]

The measured signals, which are the input for the ML models, are divided and reshaped in a 3D tensor with a shape of [700, 7, 500] in which 700 is the number of samples, i.e., the size of the database, 7 is the number of sensors, and 500 is the signal length. The output of interest labeled by twelve working status of the columns, corresponding to 12 ductility levels shown in Table 1. Furthermore, examples of various raw vibrations are depicted in Fig. 5.

Table 1. Structural states of reinforced concrete columns

Ductility level	1	2	3	4	5	6	7	8	9	10	11	12
Value (%)	20	40	60	80	90	100	150	300	450	600	800	1000

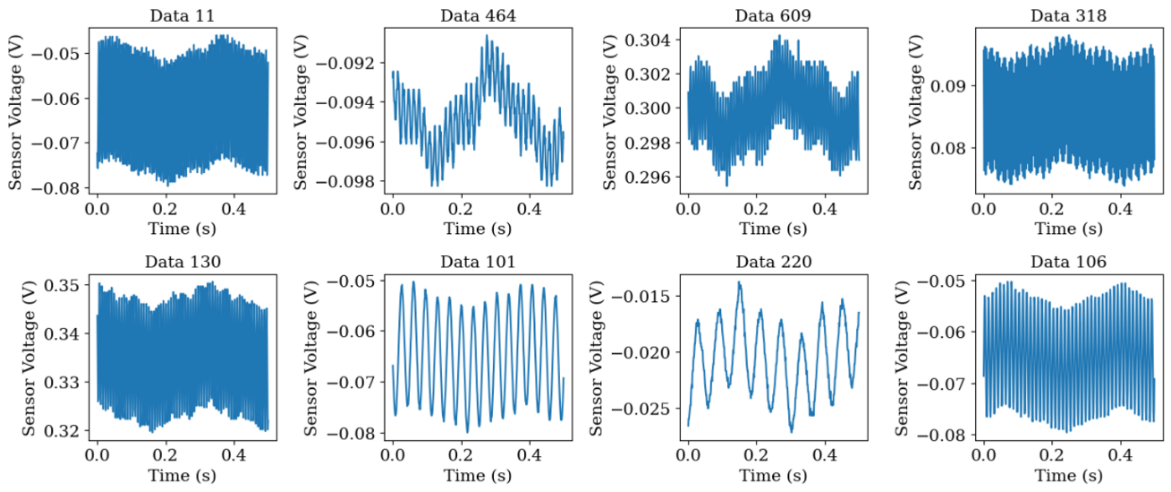


Figure 5. Representative examples of measured signals for different data samples in the database

3.2. Comparison results

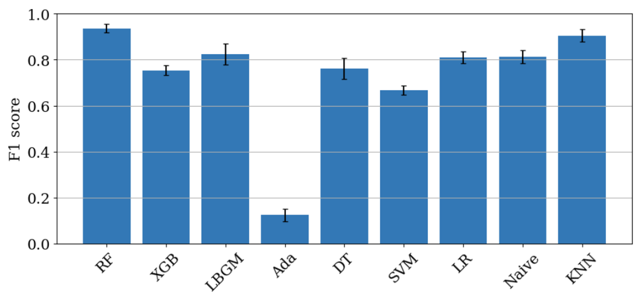
Once the database is in place, it is put through these ML-based methods described above. Initially, the authors adopt the default parameters of each method. By doing so, one can have relatively subject baseline detection results. Some key parameters of every model are enumerated in Table 2. On other hand, it is widely acknowledged that hyperparameters may have important impacts on both the model performance and efficiency; hence, hyperparameter optimization (HO) is carried out to determine the appropriate set values of the hyperparameters. Actually, this is an active and challenging topic that has received significant attention from scientists. Further details and the most recent HO techniques can be found in [21]. Herein, the practical GPyOpt [22] library is leveraged to perform HO. The set of parameters for fine-tuned models are enumerated in Table 2.

After that, the training and inference stages are carried out using the k-fold cross-validation technique, with $k = 10$ as widely used in the literature. The comparison results are graphically illustrated via a bar chart in Fig. 6, in which the y-axis denotes the f1-score of detection results, and the x-axis corresponds to the method. The standard deviation of the f1-score is represented by a vertical bar at the top of each bar. When using the default values of hyperparameters, the bagging ensemble method, Random Forest, achieves the highest f1 score of around 0.92. Unexpectedly, the second best method is the quasi-non-parameter KNN method, which yields a f1-score of 0.91 with a significantly small variance. In contrast, the Adaboost method provides poor results with a f1-score of around 0.12. Furthermore, after fine-tuning, all ML models achieve higher f1-scores; for example, KNN, FR, XGB, and LGBM can all provide SDD results with f1 values near 0.95. These results underline the importance of hyperparameter optimization in the model selection procedure because the best-performance ML model may not be the same before and after hyperparameter optimization.

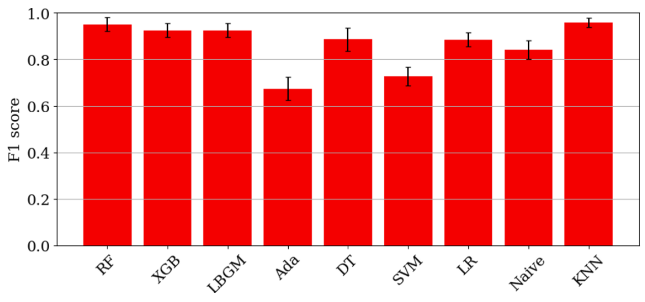
Fig. 7 presents an example of SDD results obtained by RF for the RC column problem in a more detailed way compared to only using a statistic metric such as F1-score or accuracy. In the figure, the X- and Y-axis shows the actual and predicted operations of the column; hence, diagonal cells indicate

Table 2. Hyperparameters of machine learning algorithms

Method	Hyperparameter	Fine-tuned
RF	n_estimators = 100, criterion = 'entropy', min_samples_leaf = 1, max_depth = 3	n_estimators = 60, criterion = 'gini', min_samples_leaf = 4, max_depth = 9
LR	penalty = 'l2', solver = 'lbfgs', max_iter = 100, C = 1.0	penalty = 'l2', solver = 'lbfgs', max_iter = 1000, C = 7.849
SVM	kernel = 'rbf', alpha = 0.0001, epsilon = 0.1, eta0 = 0.001	kernel = 'rbf', alpha = 0.002, epsilon = 0.1, eta0 = 0.3
LightGBM	learning_rate = 0.1, n_estimator = 100, max_depth: 2	learning_rate = 0.1, n_estimator = 250, max_depth: 10
XGBoost	learning_rate = 0.1, n_estimators = 100, max_depth = 2	learning_rate = 0.1, n_estimators = 150, max_depth = 3
AdaBoost	learning_rate = 1.0, n_estimator = 50, algorithm = 'SAMME.R'	learning_rate = 0.1, n_estimator = 130, algorithm = 'SAMME.R'
Naïve Bayes	No parameter	No parameter
DT	criterion = 'Entropy', splitter = 'best', min_samples_leaf = 1	The same as default
KNN	n_neighbors = 5, weights = 'uniform', metric = 'euclidean'	n_neighbors = 7, weights = 'uniform', metric = 'manhattan'



(a) Default models



(b) Fine-tuned models

Figure 6. Comparison of SHM results obtained by different ML-based algorithms

the number of data points correctly predicted by RF, i.e., the actual label on the X-axis matches the predicted one on the Y-axis. On the other hand, the non-zero off-diagonal cells represent the number of misclassified by RF.

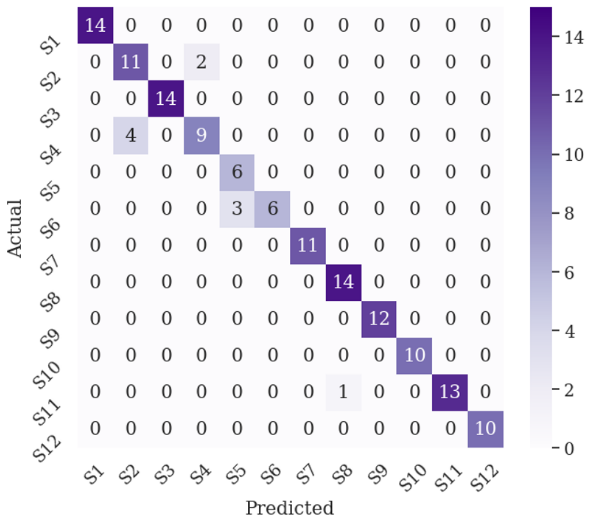


Figure 7. Confusion matrix of SHM results obtained by Random Forest

On the other hand, the efficiency of these methods is also compared in terms of training time, inference time, and model sizes, as listed in Table 3. The inference time is the computational time required to assess the corresponding structural states of validation data. In the Appendix, we have provided a code snippet showing how to calculate the inference time, i.e., the execution time of the prediction command. The model size is the memory required to store the model definition, its parameters, and the wrapper, if necessary. Herein, each ML model, after training, is saved into a separate file, and the size of this file is reported as the model size. It can be seen that KNN does not require a training process, as discussed in the previous section; thus, the training time is near zero. In contrast, its inference time is higher than other methods because it needs to repeat the distance calculation every time with all training data. Overall, it can be seen that ML-based SDD methods have small model sizes, thus requiring only a small amount of memory, while its inference times are less than a second. Thus, they can be installed in edge devices attached to real structures to provide real-time monitoring results.

Table 3. Computational efficiency of Machine learning models

Method	RF	LR	SVM	LBGM	XGB	Ada	Naïve	DT	KNN
Training time (s)	0.4	1.0	0.09	3.6	2.6	1.96	0.005	0.17	0.001
Inference times (s)	0.01	0.005	0.18	0.01	0.004	0.06	0.04	0.001	0.04
Model size (kB)	66	68	43	50	48	71	49	63	73

4. Conclusions

In this study, a comparison study on SDD methods based on ML algorithms using raw vibration data has been carried out. Compared to usually considered black box DL models, the ML algorithms have significantly fewer trainable parameters and a is more understandable; meanwhile, compared to classical statistical-based methods, which require in advances some subjective assumptions, ML algorithms are more objective as they learn directly from the database. There are, in total, nine models, including simple and understandable algorithms such as logistic regression to complex and powerful ones such as XGBoost. For each model, the main idea and theoretical background, and graphical representation HAVE been introduced.

These algorithms are applied to an experimental database collected from experiments of full-size reinforced concrete columns in the literature. Interestingly, the authors found that the KNN method works extremely well since it can provide second-best detection accuracy without requiring a training process; however, at the inference stage, its inference time is longer than those of other methods. The author also found that the bagging ensemble model, i.e., Random Forest outperforms other ensemble models, such as adaptive and gradient boosting methods.

In the future step, based on the procedure presented in this study, we will carry out an extensive comparison study including more ML/DL models [23, 24] and a large number of benchmark examples, and expect to gain more conclusive insights about the applicability of ML/DL model-based SDD methods. It is also interesting to explore the robustness of these algorithms against different unwanted effects on vibration signals, such as noise, anomaly, missing data, etc.

References

[1] Kaveh, A., Mahdavi, V. R. (2016). Damage identification of truss structures using CBO and ECBO algorithms. *Asian Journal of Civil Engineering*, 17(1):75–89.

[2] Kaveh, A., Zolghadr, A. (2015). An improved CSS for damage detection of truss structures using changes in natural frequencies and mode shapes. *Advances in Engineering Software*, 80:93–100.

[3] Kaveh, A., Javadi, S. M., Maniat, M. (2014). Damage assessment via modal data with a mixed particle swarm strategy, ray optimizer, and harmony search. *Asian Journal of Civil Engineering*, 15(1):95–106.

[4] Deraemaeker, A., Reynders, E., De Roeck, G., Kullaa, J. (2008). Vibration-based structural health monitoring using output-only measurements under changing environment. *Mechanical Systems and Signal Processing*, 22(1):34–56.

[5] Qui, L. X. (2023). Damage identification of trusses using limited modal features and ensemble learning. *Journal of Science and Technology in Civil Engineering (STCE) - HUCE*, 17(2):9–20.

[6] Addin, O., Sapuan, S. M., Mahdi, E., Othman, M. (2007). A Naïve-Bayes classifier for damage detection in engineering materials. *Materials & Design*, 28(8):2379–2386.

[7] Zhou, Q., Zhou, H., Zhou, Q., Yang, F., Luo, L., Li, T. (2015). Structural damage detection based on posteriori probability support vector machine and Dempster–Shafer evidence theory. *Applied Soft Computing*, 36:368–374.

[8] Diao, Y., Jia, D., Liu, G., Sun, Z., Xu, J. (2021). Structural damage identification using modified Hilbert–Huang transform and support vector machine. *Journal of Civil Structural Health Monitoring*, 11(4):1155–1174.

[9] Karbassi, A., Mohebi, B., Rezaee, S., Lestuzzi, P. (2014). Damage prediction for regular reinforced concrete buildings using the decision tree algorithm. *Computers & Structures*, 130:46–56.

[10] Abdallah, I., Dertimanis, V., Mylonas, H., Tatsis, K., Chatzi, E., Dervili, N., Worden, K., Maguire, E. (2018). Fault diagnosis of wind turbine structures using decision tree learning algorithms with big data. In *Safety and Reliability – Safe Societies in a Changing World*, CRC Press, 3053–3061.

[11] Salkhordeh, M., Mirtaheri, M., Soroushian, S. (2021). A decision-tree-based algorithm for identifying the extent of structural damage in braced-frame buildings. *Structural Control and Health Monitoring*, 28 (11).

[12] Ghiasi, A., Ng, C.-T., Sheikh, A. H. (2022). Damage detection of in-service steel railway bridges using a fine k-nearest neighbor machine learning classifier. *Structures*, 45:1920–1935.

[13] Zhou, Q., Ning, Y., Zhou, Q., Luo, L., Lei, J. (2012). Structural damage detection method based on random forests and data fusion. *Structural Health Monitoring*, 12(1):48–58.

[14] Mangalathu, S., Jang, H., Hwang, S.-H., Jeon, J.-S. (2020). Data-driven machine-learning-based seismic failure mode identification of reinforced concrete shear walls. *Engineering Structures*, 208:110331.

[15] Dong, W., Huang, Y., Lehane, B., Ma, G. (2020). XGBoost algorithm-based prediction of concrete electrical resistivity for structural health monitoring. *Automation in Construction*, 114:103155.

[16] Wang, J., Du, X., Qi, X. (2022). Strain prediction for historical timber buildings with a hybrid Prophet-XGBoost model. *Mechanical Systems and Signal Processing*, 179:109316.

[17] Zhang, H., Zhou, Y., Huang, Z., Shen, R., Wu, Y. (2023). Multiparameter Identification of Bridge Cables Using XGBoost Algorithm. *Journal of Bridge Engineering*, 28(5).

[18] Dang, V.-H., Le-Nguyen, K., Nguyen, T.-T. (2023). Semi-supervised vibration-based structural health monitoring via deep graph learning and contrastive learning. *Structures*, 51:158–170.

[19] Dang, V.-H., Vu, T.-C., Nguyen, B.-D., Nguyen, Q.-H., Nguyen, T.-D. (2022). Structural damage detection framework based on graph convolutional network directly using vibration data. *Structures*, 38:40–51.

[20] Mo, Y.-L., Song, G., Belarbi, A., Gu, H., Moslehy, Y. (2009). *loading*. DesignSafe-CI.

[21] Yang, L., Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, 415:295–316.

[22] authors, T. G. (2016). *GPyOpt: A Bayesian Optimization framework in python*.

[23] Hung, T. V., Viet, V. Q., Thuat, D. V. (2019). A deep learning-based procedure for estimation of ultimate load carrying of steel trusses using advanced analysis. *Journal of Science and Technology in Civil Engineering (STCE) - NUCE*, 13(3):113–123.

[24] Do, D. T. T., Thai, S., Bui, T. Q. (2022). Long short-term memory for nonlinear static analysis of functionally graded plates. *Journal of Science and Technology in Civil Engineering (STCE)-HUCE*, 16(3): 1–17.

[25] Bernal, D., Dyke, S. J., Lam, H.-F., Beck, J. L. (2002). Phase II of the ASCE benchmark study on SHM. In *Proceedings of the 15th ASCE Engineering Mechanics Conference*, Columbia University, Columbia University.

Appendix A.

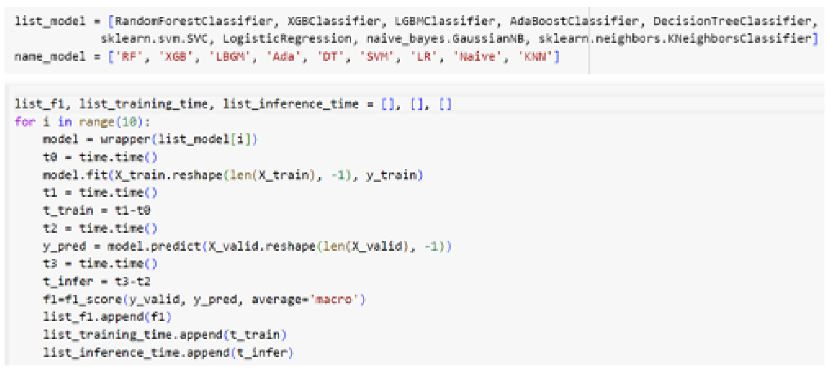
a. F1 score

The F1 score is calculated by:

$$F_1 = \frac{2}{1/P + 1/R}$$

where $P = \dfrac{TP}{TP + FP}$ and $R = \dfrac{TP}{TP + FN}$, with TP, FN, FP are true positive, false negative, and false positive samples. Further explanations could be found in [20]. In general, the f1-score reflects better the performance of a classification model, even when facing imbalanced data, as it accounts for both two types of errors in the senses of statistic, i.e., false negative and false positive, rather than only focusing on the accuracy.

b. Code snippet for determining the training and inference time



```
list_model = [RandomForestClassifier, XGBClassifier, LGBMClassifier, AdaBoostClassifier, DecisionTreeClassifier,
              sklearn.svm.SVC, LogisticRegression, naive_bayes.GaussianNB, sklearn.neighbors.KNeighborsClassifier]
name_model = ['RF', 'XGB', 'LBGM', 'Ada', 'DT', 'SVM', 'LR', 'Naive', 'KNN']
```

```
list_f1, list_training_time, list_inference_time = [], [], []
for i in range(10):
    model = wrapper(list_model[i])
    t0 = time.time()
    model.fit(X_train.reshape(len(X_train), -1), y_train)
    t1 = time.time()
    t_train = t1-t0
    t2 = time.time()
    y_pred = model.predict(X_valid.reshape(len(X_valid), -1))
    t3 = time.time()
    t_infer = t3-t2
    f1=f1_score(y_valid, y_pred, average='macro')
    list_f1.append(f1)
    list_training_time.append(t_train)
    list_inference_time.append(t_infer)
```

Figure A.1. Code snippet showing how to calculate the training and inference times

c. Additional benchmark problem

Another example utilized to demonstrate the applicability of the proposed model selection process is a steel frame with braces at the university of British Columbia [25], as illustrated in Fig. A.2. The structure was subjected to white noise excitations, and its responses were recorded via 16 sensors. For this example, the measured signals are divided and reshaped in a 3D tensor with a shape of [1735, 16, 300] in which 1735 is the number of samples, 16 is the number of sensors, and 300 is the signal length. The output of interest labeled by twelve working status of the columns, corresponding to 9 damage levels shown in Table A.1.

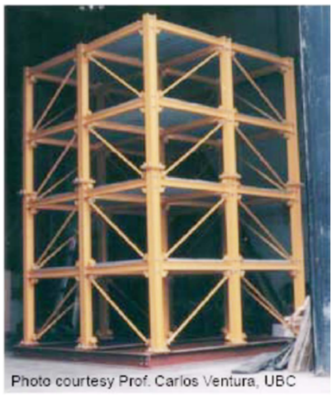


Figure A.2. Benchmark example with a steel frame structure at the university of British Columbia [25]

Table A.1. Structural states of the steel frame

States	Damage level
1	Healthy
2	Remove side braces
3	Remove one bay's braces
4	Remove braces on the 1st and 4th floors
5	Remove braces on the 1st floors
6	Remove east-face braces
7	Remove all braces
8	Remove all braces and loose east-face bolts
9	Remove all braces and loose bolts on the 1st and 2nd floors

The comparison results for the steel frame structure are visually shown in Fig. A.3 through a bar chart. For this structure, the boosting ensemble model such as LGBM, and XGB outperforms other algorithms, whereas the KNN model provides less accurate results. It is noted that each structure exhibits unique vibration characteristics, and the difficulty of the SDD detection tasks depends on the type of damage; thus, there is no one-size-fits-all model that can achieve optimal results for every SDD problem. Thus, it is necessary to carry out a model selection study to select the most appropriate model for a specific problem. It is possible that a traditional statistic model can outperform most of complex ML algorithms on a small dataset but for other, powerful ensemble ML models are resorted to.

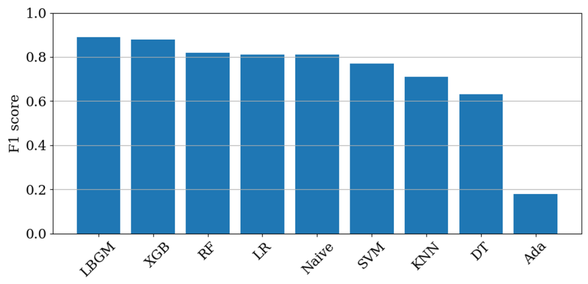


Figure A.3. Comparison of SHM results obtained by different ML-based algorithms for the steel frame structure