Journal of Cleaner Production 382 (2023) 135334

Contents lists available at ScienceDirect

Journal of Cleaner Production

journal homepage: www.elsevier.com/locate/jclepro

Do We Need Exotic Models? Engineering Metrics to Enable Green Machine Learning from Tackling Accuracy-Energy Trade-offs

M.Z. Naser [a,b,1,*]

[a] School of Civil and Environmental Engineering & Earth Sciences, Clemson University, Clemson, SC, 29634, USA
[b] AI Research Institute for Science and Engineering (AIRISE), Clemson University, Clemson, SC, 29634, USA

ARTICLE INFO

Handling Editor: Prof. Jiri Jaromir Klemeš

Keywords:
Machine learning (ML)
Structural engineering
Energy
Carbon emissions

ABSTRACT

Machine learning (ML) has been shown to bypass key limitations of traditional methods (i.e., physical testing and numerical simulations) and hence presents itself as an attractive technology in engineering. While the integration of ML brings exciting opportunities, it also introduces unique challenges. One such challenge is related to the heavy reliance of ML on large datasets and computing facilities – for algorithm development, training, deployment, and storage. To realize Green ML (GML), this paper argues that ML users are to be cognizant of the hidden costs of energy consumption and subsequent carbon emissions arising from ML modeling, and hence they are ethically bound to apply ML responsibly. In this pursuit, a series of simple and exotic ML algorithms are examined, and their performance on a relatively large dataset (~8000 observations) is documented on five fronts; predictive performance, model size, training time, energy consumption, and generatedcarbon emissions. In addition, this work also examines the influence of algorithmic architecture, processing language, number of features, as well as dataset size on model predictivity and energy consumption. Findings from this investigation infer that a 23–99% reduction in energy consumption and carbon emissions can be attained (while maintaining a comparable level of accuracy) by adopting simple as opposed to exotic ML models. The same findings have also led to the development of two new metrics that can tie the predictivity (i.e., level of accuracy) to the amount of energy consumed per algorithm. These metrics can be used to compare model performance in a similar manner to that traditionally used to assess the accuracy of ML predictions, thereby integrating energy-based awareness as a key dimension for model comparison.

1. Introduction

Ongoing advancements in machine learning (ML) have made it possible to extend this technology to various industries. One such industry is structural engineering which also happens to rely on numerical simulations for multi-scale problems (Mahendran, 2015). While traditional simulation techniques (such as the finite element (FE) method) continue to be favorable given the familiarity of structural engineers with the fundamentals of such techniques, recent works have noted a rising interest in ML (Farrar and Worden, 2012; D'Amico et al., 2019). The outcome of such recent works, together with others (Rafiei and Adeli, 2017; Naser, 2020a, 2022a; Babanajad et al., 2017), notes the potential of ML as a complementary technology that can offer novel solutions to structural engineers.

More specifically, a collection of structural engineering-based ML approaches have been reviewed in (Wang et al., 2021; Xie et al., 2020; Tapeh and Naser, 2022) with a common convergence of how ML can bypass the limitations of traditional methods and hence advocate for its adoption as we near the era of *Construction 4.0*. In addition, the application of ML is seen to be fruitful across a number of sub-disciplines belonging to structural engineering. For example, ML models are reported to be able to accurately predict the properties of construction materials (Ashrafian et al., 2020; Gola et al., 2019), detecting damage in structures (Diez et al., 2016; Hasni et al., 2017), evaluating structural performance of elements (Ashteyat et al., 2020; Panev et al., 2021), components (Tarawneh et al., 2020; Alwanas et al., 2019), and structures (Fu, 2020). In addition, ML has also been used as an aid tool for laying out blueprints and structural plans (Liao et al., 2022; Freischlad and Schnellenbach-Held, 2005; Cheng et al., 2020). Recent trends in existing literature show that ML is being integrated into complex and

* School of Civil and Environmental Engineering & Earth Sciences, Clemson University, Clemson, SC, 29634, USA.
E-mail address: mznaser@clemson.edu.
[1] www.mznaser.com

https://doi.org/10.1016/j.jclepro.2022.135334
Received 21 January 2022; Received in revised form 17 October 2022; Accepted 24 November 2022
Available online 25 November 2022

unique problems as well (Avci et al., 2021; Naser, 2021).

Given the current interest in adopting ML, a look toward current practices (Rafiei and Adeli, 2017; Naser, 2020a, 2020b; Babanajad et al., 2017; Xie et al., 2020; Adeli, 2001; Sun et al., 2021; Çevik et al., 2015; Golafshani and Behnood, 2019; Degtyarev, 2021; Mangalathu and Burton, 2019; Pan and Zhang, 2021; Mohammadi and Al-Fuqaha, 2018) reveals a few interesting observations; 1) we continue to lack a unified procedure to apply ML to our problems,[2] 2) the application of ML is a user-derived operation wherein individual expertise influence and drive model development and deployment, and 3) ML is primarily applied to well-defined problems of a smaller search space as that commonly faced in other domains (e.g., medicine, or finance, etc.). These observations imply that we are yet to witness an upcoming boom in ML within structural engineering – which is likely to take place in the coming few years.

Therefore, it is of merit to proactively set the stage to mold the integration of ML in this domain. While it is virtually impractical to deliver one solution that fits all early into adopting ML, a few essential items can be emphasized and discussed. Such items may include the need for transparency, interpretability, fairness, and energy efficiency. The latter is the focus and motivation of this work.

With the growing need for larger yet accurate models, structural engineers will be expected to develop complex models. Such models are often tied to intricate architectures/topologies and are likely to require a tremendous amount of energy for training, development, storage, and deployment. To put this into perspective, two recent works have estimated the cost of training various deep learning models in terms of carbon emissions and monetary costs and reported estimates varying between 6.0 and 150,000 Kg and between $41.0 to $3.2 million per model (Strubell et al., 2020; Jackson, 2019). The reader is to note that a human being, on average, contributes to about 5000 Kg of carbon emissions on an annual basis, and 150,000 Kg of carbon emissions is equivalent to that emitted through the lifetime of five fuel-based vehicles (Strubell et al., 2020).

On a larger scale, 2.3% of global carbon emissions are attributed to the information and communication technology sector, as estimated in a recent report by the Global e-Sustainability Initiative (SMARTer, 2020). For comparison, the cement and aviation industry generates carbon emissions of about 5% and 1.7%, respectively (Farfan et al., 2019; Ritchie and Roser, 2021). Given the rise in recent calls to cut carbon emissions by 50% within the next ten years to negate the escalating rates of climate change, one means to align with such calls is to ensure considerate ML energy expenditure (Jackson, 2019).

On a different front, exotic or highly advanced versions of existing ML models provide improvements in accuracy, among others; however, such improvements may or may not positively reflect upon the consumed resources to attain such improvements. One particular example that fits the former is the well-known computer vision model released in 2015 known as *ResNet*. The improved version of this model, *ResNeXt*, not only required 35% more computational resources to train than *ResNet* but only achieved a 0.5% improvement in accuracy (Lu et al., 2018). A second example of the latter is that related to the deep learning network, *SqueezeNet*, which attained the same accuracy as *AlexNet* (the winner of The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012, with fifty times fewer parameters and a squeezed size of 0.5 MB as compared to 240 MB for *AlexNet* (Iandola et al., 2016).

It is true that improved accuracy comes in handy; however, for *some applications,* minimal improvement in accuracy may not warrant the additional energy-based resources needed to realize such accuracy. For instance, predicting if a concrete mixture can develop strength of 38 or 40 MPa may not *significantly* alter the practical application of such a mixture since both strengths remain within the range of normal strength concrete. In this example, pushing a typical ML model to chase accuracy metrics of unity may not be warranted. Similarly, training an exotic model (e.g., an ensemble or a deep learning model) to predict the above when a simple model (i.e., a decision tree) can be trained to yield a comparable prediction accuracy may also be unnecessary. On a parallel front, developing a highly complex neural network to classify if structural members will undergo damage or not may lead to high energy consumption as compared to developing a leaner classifier – one that is based on logistic regression (assuming that both models can deliver a comparable performance).

Given the rise in ML adoption, which is also expected to initiate a ML-based race within the construction industry, one can appreciate the environmental and societal benefits of arriving at energy efficient ML models. Realizing energy efficient ML models, or strategies that enable cognizant and reduced-order modeling, can be of merit since these are the main tasks that ML users can control in our domain – as opposed to tackling other dimensions to this problem such as developing new algorithms or hardware, or storage, or tuning of cloud computing capabilities etc. While advancements in other dimensions are indeed necessary to realize a more energy efficient ML; practically speaking, investments in new hardware-like facilities by structural engineering firms are often expected to last for a few years, and hence embracing a user-driven approach is seen of equal importance, if not of more importance, to accelerate adopting Green ML (GML). This work argues that adopting cognizant and reduced-order modeling strategies can be seen as a sustainable approach to ML adoption. The same exercise will educate structural engineers to be inherently efficient and is expected to have long-lasting benefits.

This paper examines cognizant and reduced-order strategies that can be practiced by structural engineers to minimize energy consumption arising as a byproduct of developing ML models. In addition, this paper carries out a series of comparisons between simple and exotic ML algorithms to report on their predictive performance, model size, energy consumption, and storage need. More specifically, this work explores the influence of algorithm architecture, processing language, number of features, as well as dataset size.[3] The selected algorithms comprise; Gradient Boosted Trees (GBT, including those of Extreme and Light Gradient nature), Keras Deep Residual Neural Network (KDNN)), TensorFlow Deep Learning (TDL), Vowpal Wabbit (VW), and Random Forest (RF). The performance of these algorithms was compared in classifying if a given concrete mixture can attain specified design strength at in-situ conditions by assessing a large tabular dataset.

2. Materials and methods

2.1. Selected machine learning algorithms

The selected algorithms in this work are briefly described herein, and their full description can be found in their respective references, as well as in (Ziegel, 2003; Ketkar and Ketkar, 2017; Chen and Guestrin, 2016;

[2] It is worth noting that the same is also true in other domains as well.

[3] Given the wide possibilities and approaches of tuning algorithms (especially those of different topologies), it is deemed necessary to establish the following rationale early into this work. Thus, the reader is to realize that in order to enable reproduction of this analysis and to allow reporting true benchmarks on models' performance, all of the examined algorithms were applied in their default settings. As such, the goal of this work is *not* to arrive at a "solution" to the examined phenomenon, *nor* to identify the most "suited" algorithm to solve such a phenomenon, but rather to report on findings from each of the examined algorithms (as obtained from their default settings) from an energy point of view. A full diagnostic test to explore the effect of hardware, explicit tuning of algorithms, as well as other algorithms that were not showcased herein, is deemed cumbersome to fit into one work and hence interested readers are invited to extend this work beyond its original message. The author hopes that the collective effort within this community and in the coming years will generate in depth insights to expedite the adoption of GML.

Scikit, 2020a; XGBoost Python Package, 2020; Scikit, 2021a; Natekin and Knoll, 2013; Keras, 2020; TensorFlow, 2020). These algorithms are among the most widely used algorithms in this research area, as noted in the following recent review papers (Tapeh and Naser, 2022; Thai, 2022).

2.1.1. Variants of Gradient Boosted Trees (GBT)

A GBT trains a simple tree-like model and then uses the first model's error as a feature to build successive models. Focusing on errors obtained from a feature when building successive models reduces the overall error in the model. Two variants of GBT are the XGBoost and Light Gradient Boosting Machine (LGBM). The XGboost improves the performance of a GBT via a weighted quantile sketch (an approximation algorithm that determines how to split candidates in a tree) and the sparsity-aware split finding (which works on sparse data, as well as data with missing values). The XGboost uses a pre-sorted algorithm and a histogram-based algorithm for computing the best split. This algorithm was first published by Chen and Guestrin (2016), and more details on this algorithm can be found in such work.

On the other hand, the LGBM algorithm by Microsoft (Ke et al., 2017) introduces two techniques to improve the performance of a GBT. These techniques are gradient-based one-side sampling (which identifies the most informative observations and skips those less informative) and exclusive feature bundling (which groups features in a near-lossless way). The LGBM is an improvement over both the XGBoost and traditional GBTs.

The code of the used XGBoost can be found online at (Scikit, 2020a; XGBoost Python Package, 2020). This algorithm incorporates the following pre-tuned settings; learning rate = 0.05, maximum tree depth = 3.0, subsample feature = 0.5, number of boosting stages = 6,250, and minimum interval for early stopping = 200. The LGBT algorithm can also be found at (LightGBM, 2020) with the following default settings: learning rate = 0.05, maximum depth = "none", number of boosting stages = 6,250, and minimum interval for early stopping = 200. Finally, two GBTs were used (one of Python origin and another of R origin), which can also be found herein in their default settings (Scikit, 2021a; Natekin and Knoll, 2013) .[4]

2.1.2. Keras Deep Residual Neural Network (KDNN)

Keras is a high-level library for developing neural networks (Li et al., 2018). In a residual network, a direct connection exists, linking data points to the outputs. Such a connection smoothens the loss function and enables better network optimization. In the used KDNN, default settings of a learning rate of 0.03 was used, along with a *Sigmoid* activation function, *Adam* optimizer, and different layer architectures (1 layer of 64 units, 1 layer of 1536 units, 2 layers of 64, and 64, and 3 layers of 256, 128, and 64 units). KDNN can be readily found at (Keras, 2020).

2.1.3. TensorFlow deep learning (TFDL)

A TFDL is an open-source and free neural network-based model that uses Deep Learning and is hosted by Google (Abadi, 2016). The used algorithm in its default settings (neurons in each layer = 100, number of training examples = 128, optimizer = *Adam*, adaptive learning rate, early stopping window = 5, and activation function of *ReLu*) can be found at (TensorFlow, 2020).

[4] This analysis acknowledges that each algorithm was developed with certain default settings as per the developers of each algorithm. Hence, algorithm settings were not set to be similar in GBT variants since: 1) other algorithms such as KDNN, TFDL, etc. also have unique settings, and 2) it is not possible to ensure that all algorithms will have similar settings (given that some settings which may exist for a particular algorithm, yet may not exist for another/all algorithms). As mentioned earlier, this analysis does *not* seek to identify the most "suited" algorithm, but rather to report on findings from each of the examined algorithms from an energy perspective.

2.1.4. Vowpal Wabbit (VW)

Vowpal Wabbit is a fast and out-of-core algorithm learner capable of streaming data (which comes in handy in large databases that cannot be supported by existing hardware). VW initially started at Yahoo and then moved to Microsoft. VW has a learning rate of 0.1, *logistic* loss function, power on the learning rate decay = 0.5, and can be found herein (VowpalWabbit, 2021).

2.1.5. Support vector machines (SVM)

The Support Vector Machine (SVM) algorithm aims to identify a line or a boundary (i.e., hyperplane) in an n-dimensional space to classify data into separate classes (Boser et al., 1992). This plane is found by maximizing the distance between data points of each class to allow for confident classification (Çevik et al., 2015). SVM uses a special form of mathematical function defined as kernels (k). A kernel function transforms inputs into the required form. The used SVM utilizes penalty parameter = 12,915, Gamma parameter = 0.208, and can be found at (Scikit, 2021b).

2.1.6. Logistic regression (LR) and Elastic Net (EN)

The logistic regression model is a generalized linear model that applies a binomial distribution to fit regression models to binary response variables. The applied LR model herein had the following settings; Sigma = 10^{-6}, fit intercept = *True*, and tolerance = 0.0001. The Elastic Net (EN) classifier is an extension of LR that applies L1 (estimate the median of the data) and L2 (estimate the mean of the data) prior as regularizers. Both models were taken from (Scikit, 2021c).

2.1.7. Random Forest (RF)

The Random Forest (RF) algorithm is an ensemble learner that forms a series of decision trees (Liaw and Wiener, 2002). In a classification problem, the majority of predictions, as compared against all trees, are used to consolidate the final outcome. Two RFs were used (one of Python origin and another of R origin), which can also be found herein in their default settings herein (Scikit, 2020b) and (Breiman, 2001), respectively.

2.2. Description of dataset

This section describes the examined dataset to be used in this work. This dataset was selected given its tabular nature, which is also likely to be present in most structural engineering problems (Tapeh and Naser, 2022; Thai, 2022). As such, all observations were numeric, and no missing data points were present. This allows for somewhat of a fair field for comparison for the selected ML algorithms, given each's tendency to handle missing and/or categorical data differently (which may skew the primary focus of this investigation).

This dataset comprises compressive strength and mixture proportion of about 8000 concrete mixtures as measured from in-situ conditions and reported by Young et al. (Young et al., 2019; Yu, 2021). The compressive strength of each concrete mixture at 28 days was measured following the ASTM C39 standard. In addition, mixture proportions including water, cement, and fly ash contents (in kg/m^3 of concrete), water-reducing admixture (WRA), and air-entraining admixture contents (AEA in 0.01kg/kg of cementitious material), coarse and fine aggregate contents (in kg/m^3 of concrete), and fresh air content (in volume %) were reported. The focus of this dataset was to predict if a given concrete mixture will be able to develop its design compressive strength of concrete at in-situ conditions. As such, each measured mixture was compared against that of the design strength. If the attained strength is equal to or exceeds that of the design strength, then the mixture is deemed satisfactory. If not, then the mixture is labeled "*under-designed*".

After cleansing the dataset of outliers and measurements with features of missing values, 6647 measurements were labeled as *satisfactory*, and 1099 measurements were labeled as *under-designed*. This cleansing

process was carried out using a new approach that has been recently published by the author, wherein unsupervised and supervised learning methods are used to identify and remove outliers (see (Naser, 2022b) for full details on this approach).[5] Fig. 1 shows additional insights into the statistical distributions of each feature used in this dataset.

2.3. Model development

Given the large range of features noted in the previous section, it is thought to apply data processing methods such as feature normalization (Marsland, 2014). In this process, the collected data is transformed such that the standard deviation is set equal to unity and the mean equal to zero (Luor, 2015). Then, additional steps were taken to ensure the proper performance of the model. For a start, the dataset was randomly shuffled to eliminate the influence of neighboring measurements and data points (especially those taken within one in-situ job). Then, the shuffled dataset is split into two sets – a training and a testing set. The split of choice was 70:30 as per recommendations of recent works (Abubakar and Tabra, 2019; Biswas et al., 2020).

The training set was further processed via 10-fold cross-validation technique which became handy in ensuring the proper distribution of samples and hence allowed the algorithms to be examined at different subsets. In this technique, the training set is split into 10 equal-sized subsets such that 9 subsets are used for training, and the remainder of these is kept for validation (Pedregosa et al., 2011). All subsets were the same across all algorithms.

In addition to adopting a 10-fold cross-validation technique, model predictions were also checked against a series of performance metrics that measure the closeness of a predicted outcome to that predicted by a given ML model (Schmidt and Lipson, 2010; Laszczyk and Myszkowski, 2019). In this work, metrics that are commonly used by the structural engineering domain are selected (Degtyarev, 2021; Alavi et al., 2010; Naser and Seitllari, 1007; Taffese and Sistonen, 2017). Three classification metrics are used in this classification-based problem. These metrics are accuracy (ACC), Area under the ROC curve (AUC), Log Loss Error (LLE). Additional details on each metric are shown herein.

$$ACC = \frac{TP + TN}{P + N} \tag{1}$$

where, P: predictions, N: number of real negatives, TP: number of true positives, TN: number of true negatives.

- Evaluates the ratio of the number of correct predictions to the total number of samples.
- Presents performance at a single class threshold only.
- Assumes equal cost for errors (Huang and Burton, 2019).

$$AUC = \sum_{i=1}^{N-1} \frac{1}{2}(FP_{i+1} - FP_i)(TP_{i+1} - TP_i) \tag{2}$$

where, FP number of false positives, FN number of false negatives.

- A value of unity indicates an accurate prediction.

$$LLE = -\sum_{c=1}^{M} A_i log P \tag{3}$$

where, M: number of classes, c: class label, y: binary indicator (0 or 1) if c is the correct classification for a given observation.

- Penalizes for being confident in the wrong prediction.
- Has a probability between zero and unity.
- A lower value for log loss is favorable.

3. Results and discussion

This section evaluates the performance of all of the selected ML models upon the presented dataset from two main perspectives: predictivity and estimated energy consumption. In addition, this section highlights the influence of the dataset size, the number of features used in model development, ML model architecture, and programming language in more detail and from each of the aforenoted two perspectives. Finally, a discussion on the estimated carbon emissions emitted per algorithm is also presented. It is worth noting that a private cloud computing service was obtained for this analysis, and hence all algorithms were granted access to the same computation and hardware resources. As such, the influence of such resources on model performance is not discussed as it is normalized.

3.1. Insights from a predictivity perspective

The selected ML models were first evaluated from a predictivity point of view (as in how well the predictions of each model match with the measured observations). Thus, predictions from all models were compared utilizing the ACC, AUC, and LLE performance metrics. Fig. 2 and Table 1 articulate the outcome of this comparison. Table 1 shows that most, if not all, models seem to capture the phenomenon on hand adequately. The same table also shows that all models achieved comparable performance across all metrics and against training, validation, and testing splits. While some variations exist between all models in the reported metrics, a look into Fig. 2 infers that these variations can be considered minor (as per the scale of the horizontal axis).

For example, the lowest ACC value obtained against the full dataset was 0.946 by the KDNN with one layer of 64 units during its training procedure (which also happens to be the only value outside of 0.983 (or 98.3%) in ACC or AUC metrics. Beyond that, all models performed comfortably within the 98.6–100.0% range. In the case of the AUC metric, the lowest value was noted by the EN model with 99.1%, with all other models reaching the range of 99.2–100.0%. All models also performed well when their performance was compared using the LLE metric. Except for the KDNN with one layer of 64 units, all models scored within 0.05, with the majority scoring in the range of 0.01-0.004 (note that a score of 0.0 implies a perfect model).

As expected, and as Fig. 2 and Table 1 show, not a particular model scored the highest in all metrics. As such, a new composite metric was developed to enable a unified and more accessible comparison between all models developed in this study (from a predictivity perspective). This composite metric was created by combining ACC, AUC, and LLE. The synthetic metric acknowledges that higher values of ACC and AUC and lower values of LLE infer good predictivity. This metric is listed below, and Fig. 2d shows that the best performing model based on this metric is the XGboost model, followed by RF and LGBM.[6]

$$Predictivity\ (P) = \sum_{i}^{n} \frac{ACC \times AUC}{LLE} \tag{4}$$

where, i = training, validation, and testing. Higher values of this metric are favorable; for example, for a hypothetical case of ACC and AUC of unity, and LLE = 0.001, this metric yields 1000.

[5] From a practical view, and for simplicity, the reader can assume that this dataset has two classes (Class 1 contains 6647 satisfactory measurements and Class 2 with 1099 under designed measurements).

[6] Despite this comparison, the reader is reminded that the goal of this work is not to identify the best performing ML model but rather to showcase that in many instances, a variety of ML models can inherently be good candidates for solving a particular problem.

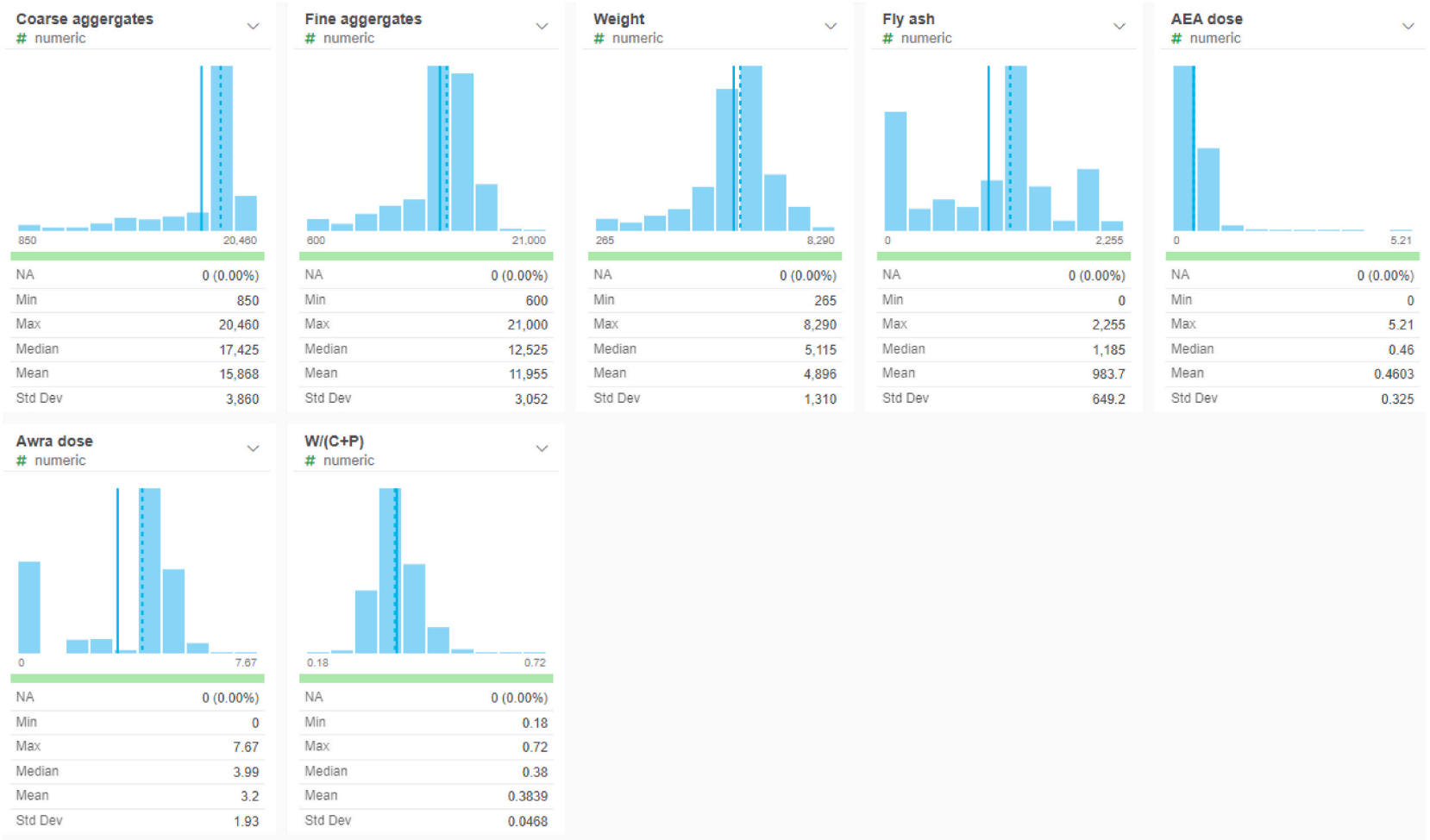


Fig. 1. Details on the dataset.

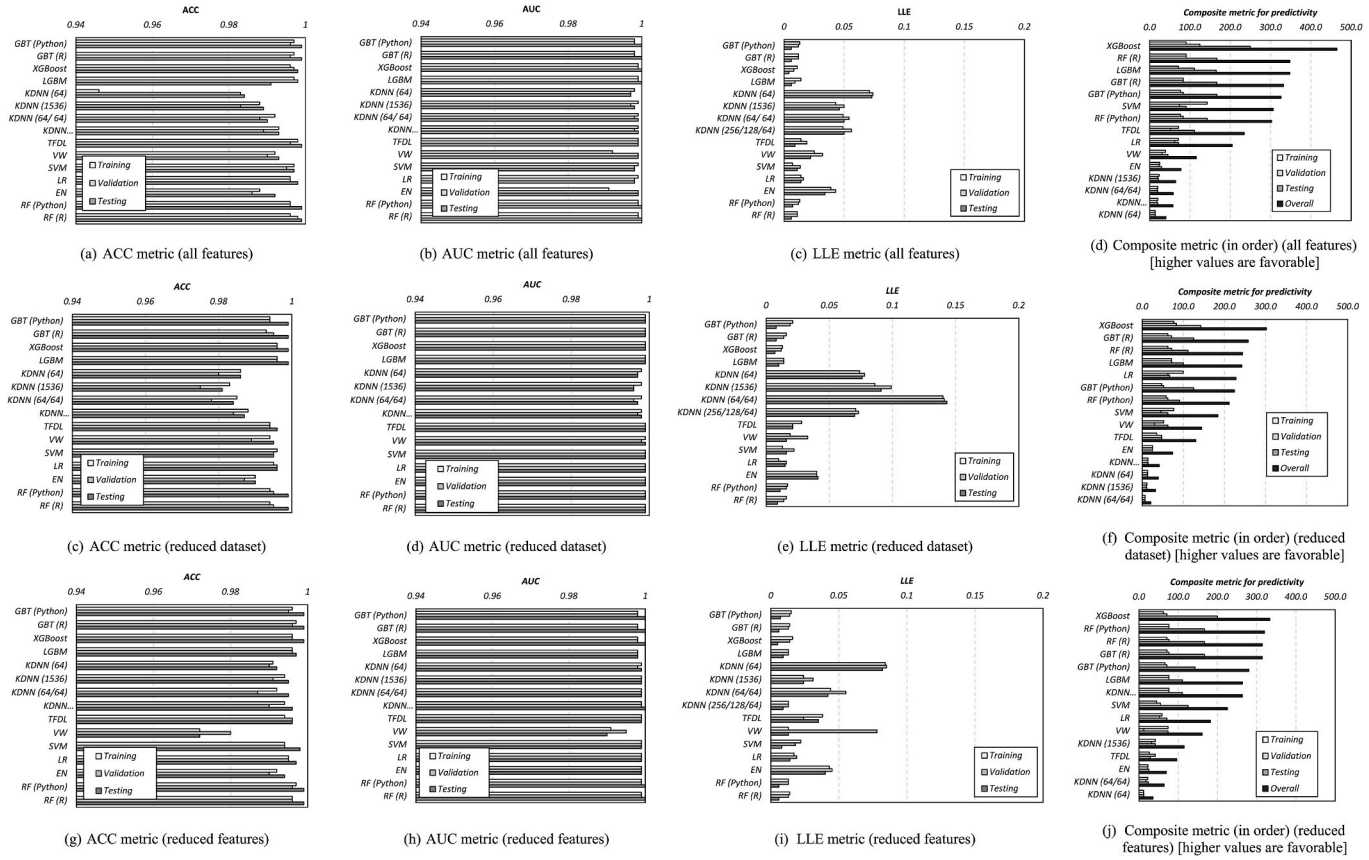


Fig. 2. Comparison of metrics.

3.1.1. *Influence of dataset size*

Given the large size of our dataset (~8000 observations), it is thought to explore the influence of reducing the size of the dataset upon the predictivity of all selected models (on training, validation, and testing split by examining the ACC, AUC, LLE and P metrics). Thus, only the training split is changed from 70% to 25% while controlling all other aspects of model development (i.e., the analysis is re-run by reducing the number of used observations from 70% of the whole dataset to 25% with the remainder (45%) of data used in the initial training omitted from the analysis). The outcome of this analysis is shown in Fig. 2 and Table 1.

When Fig. 2a, b, c, and d are compared against Fig. 2e, f, g, and h, one can see the following with regard to the performance of models in the full and reduced dataset; 1) the variation between models performance in terms of ACC and AUC is very comparable; with ML models performing slightly better in the reduced dataset, 2) LLE metric performance in the whole dataset is better than the reduced dataset (especially in the case of KDNN variants), 3) the performance of models in terms of the composite metric is higher in the case of the full dataset which reflects the larger degree of changes in LLE as to those observed in ACC and AUC, 4) despite XGBoost remaining the highest-ranked model, minor changes to overall models ranking occurred, and 5) from a practical engineering point of view, the performance of all models is adequate which may infer the suitability of all models from a predictivity perspective.

3.1.2. *Influence of number of features*

The influence of the number of features used in model development was also investigated. As such, the importance of features in all models (based on the full dataset analysis) was measured via the SHAP method (Lundberg and Lee, 2017) and presented in Fig. 3. This figure clearly shows that the most reoccurring features among all models are coarse aggregates, fine aggregates, and weight. Thus, the features in the full dataset were reduced to only these three features (while keeping the number of observations (~8000) and analysis procedure the same).

A look into the ACC, AUC and LLE metrics reveals that the performance of all models is comparable, if not improved, to that of the case of the analysis that incorporates all features. The VW model shows odd performance with a slight reduction in predictivity, which could be related to this model being heavily reliant upon the two aggregate features with little to nothing allocated for other features. Fig. 2j shows the ranking of all models as per the reduced number of features. As one can see, the XGBoost remains the highest ranking model, followed by the RF and GBT variants.

3.1.3. *Influence of programing language*

Results from the conducted analysis can also be used to investigate the influence of adopting similar models but from different programming languages (see Figs. 3 and 4). In this analysis, ML models were used of different languages RF (Python and R), as well as GBT (Python and R). For a start, the examined variants realized similar feature importance scores for most involved features, which implies consistency. Looking at the final outcome of the predictivity-based examination shows that the R version of both models performed well ahead of the Python version.

3.1.4. *Influence of model architecture*

Figs. 3 and 4 can also be used to explore the influence of model architecture. In this exercise, four architectures of the KDNN model were investigated by changing the number of units and layers in each model. The outcome of this investigation shows that these four models achieved consistent feature importance measurements (especially in the case of fine aggregates and weight). In addition, these models performed at the bottom of all other models in terms of predictivity (however, the reader must be reminded that all models performed adequately from a practical point of view).

Table 1
Evaluation of ML models.

Cases	Model		*ACC*[b]	*AUC*	*LLE*	*Model size (MB)*	*Training time (Sec)*	*Prediction time (sec)*[c]
Full features	GBT	Python	0.997/0.996/0.999	0.998/0.998/1.000	0.013/0.012/0.006	0.337	389	0.523
		R	0.997/0.996/0.999	0.998/0.998/1.000	0.012/0.012/0.006	0.259	391	2.567
	XGBoost		0.996/0.997/0.998	0.999/0.999/1.000	0.011/0.008/0.004	1.239	437	0.478
	LGBM		0.997/0.998/0.991	0.999/0.999/1.000	0.014/0.009/0.006	1.130	610	0.428
	KDNN	64[a]	0.946/0.983/0.984	0.998/0.997/0.997	0.071/0.074/0.073	0.271	677	0.851
		1536	0.988/0.983/0.989	0.999/0.997/0.998	0.043/0.051/0.046	4.757	197	0.895
		64/64	0.992/0.988/0.990	0.999/0.998/0.999	0.049/0.054/0.050	0.290	677	0.835
		256/128/64	0.993/0.989/0.993	0.999/0.998/0.999	0.049/0.056/0.050	0.446	677	1.300
	TFDL		0.998/0.996/0.999	0.999/0.999/0.999	0.014/0.019/0.009	1.818	318	0.778
	VW		0.992/0.990/0.993	0.992/0.999/0.999	0.025/0.032/0.022	0.223	696	0.404
	SVM		0.997/0.995/0.997	0.999/0.998/0.998	0.007/0.014/0.011	2.162	233	0.490
	LR		0.996/0.996/0.998	0.999/0.998/0.998	0.014/0.016/0.014	0.152	259	0.430
	EN		0.988/0.986/0.992	0.991/0.999/0.999	0.039/0.043/0.034	0.223	322	0.489
	RF	Python	0.996/0.996/0.999	0.999/0.999/1.000	0.013/0.012/0.007	0.626	366	0.427
		R	0.996/0.998/0.999	0.999/0.999/1.000	0.011/0.011/0.006	1.460	386	2.657
Reduced dataset	GBT	Python	0.994/0.994/0.999	0.999/0.999/0.999	0.021/0.019/0.008	0.201	402	0.212
		R	0.993/0.995/0.999	0.999/0.999/0.999	0.016/0.014/0.008	0.144	366	2.581
	XGBoost		0.996/0.996/0.999	0.999/0.999/0.999	0.013/0.012/0.007	0.573	446	0.259
	LGBM		0.996/0.996/0.999	0.999/0.999/0.999	0.014/0.014/0.010	1.452	246	0.255
	KDNN	64	0.986/0.980/0.986	0.998/0.997/0.997	0.074/0.078/0.076	0.184	785	0.745
		1536	0.983/0.975/0.981	0.998/0.996/0.996	0.086/0.099/0.091	1.816	857	0.580
		64/64	0.985/0.978/0.984	0.998/0.996/0.997	0.140/0.141/0.143	0.204	842	0.625
		256/128/64	0.988/0.984/0.987	0.998/0.997/0.998	0.071/0.073/0.070	0.359	750	0.556
	TFDL		0.994/0.994/0.996	0.999/0.999/0.999	0.028/0.021/0.021	3.881	310	0.586
	VW		0.994/0.989/0.995	0.999/0.998/0.999	0.019/0.033/0.016	0.136	410	0.224
	SVM		0.996/0.995/0.995	0.999/0.999/0.999	0.013/0.022/0.016	2.067	198	0.271
	LR		0.995/0.996/0.996	0.999/0.999/0.999	0.010/0.016/0.015	0.066	226	0.249
	EN		0.990/0.987/0.990	0.999/0.999/0.999	0.040/0.040/0.041	0.137	440	0.249
	RF	Python	0.994/0.995/0.999	0.999/0.999/0.999	0.017/0.016/0.011	0.336	333	0.334
		R	0.994/0.995/0.999	0.999/0.999/0.999	0.016/0.014/0.009	0.929	492	2.670
Reduced features	GBT	Python	0.996/0.995/0.999	0.998/0.998/1.000	0.015/0.014/0.007	0.333	269	0.367
		R	0.997/0.996/0.999	0.998/0.998/1.000	0.014/0.013/0.006	0.258	179	2.595
	XGBoost		0.996/0.996/0.999	0.998/0.998/1.000	0.016/0.014/0.005	0.933	458	0.595
	LGBM		0.996/0.996/0.997	0.998/0.998/0.998	0.013/0.013/0.009	5.886	337	0.482
	KDNN	64	0.991/0.990/0.992	0.999/0.998/0.999	0.084/0.085/0.082	0.221	417	0.835
		1536	0.994/0.991/0.995	0.999/0.999/0.999	0.024/0.031/0.024	2.880	217	0.773
		64/64	0.992/0.987/0.995	0.999/0.999/0.999	0.044/0.055/0.042	0.239	364	0.756

(*continued on next page*)

Table 1 (*continued*)

Cases	Model		*ACC*[b]	*AUC*	*LLE*	*Model size (MB)*	*Training time (Sec)*	*Prediction time (sec)*[c]
		256/128/64	0.994/0.990/0.996	0.999/0.999/1.000	0.013/0.013/0.009	0.390	406	0.815
	TFDL		0.994/0.996/0.996	0.999/0.999/0.999	0.038/0.024/0.035	4.061	362	0.675
	VW		0.972/0.980/0.972	0.991/0.995/0.990	0.013/0.078/0.013	0.176	527	0.392
	SVM		0.994/0.994/0.998	0.999/0.999/0.999	0.022/0.018/0.008	2.115	184	0.450
	LR		0.995/0.995/0.997	0.999/0.999/0.999	0.017/0.019/0.014	0.151	134	0.480
	EN		0.992/0.990/0.994	0.999/0.999/0.999	0.043/0.045/0.040	0.176	290	0.400
	RF	Python	0.997/0.996/0.999	0.999/0.999/1.000	0.013/0.013/0.006	0.579	103	0.459
		R	0.996/0.996/0.999	0.999/0.999/1.000	0.014/0.013/0.006	1.403	171	2.661

[a] Training/validation/testing.
[b] Number of units in each layer.
[c] To score 1000 observations.

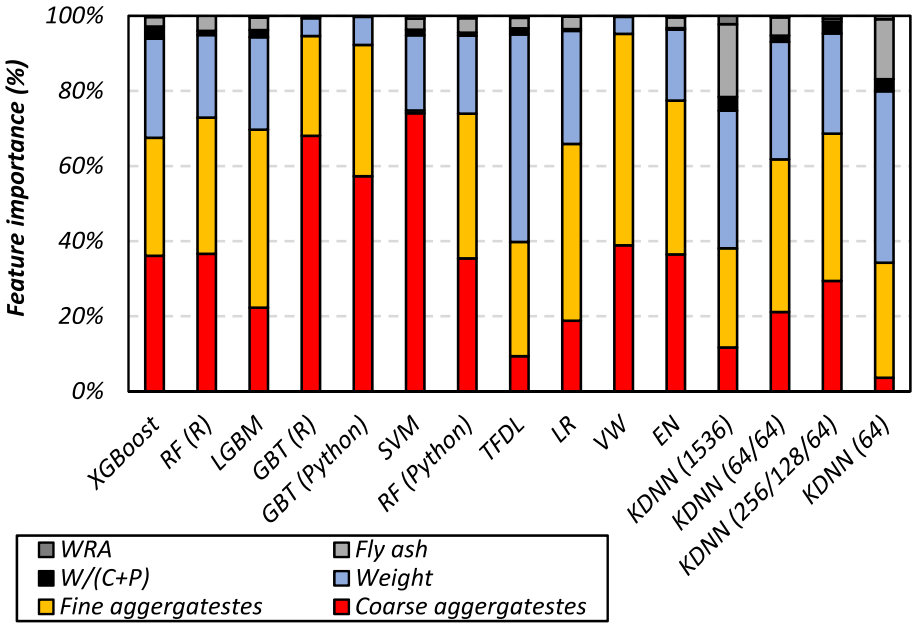


Fig. 3. Feature importance as measured in all models.

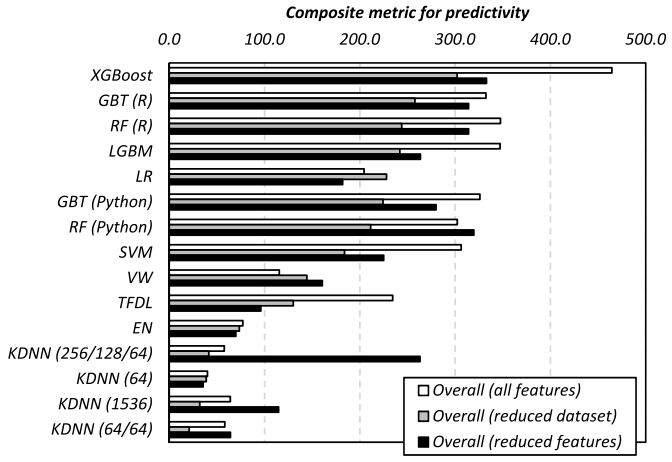
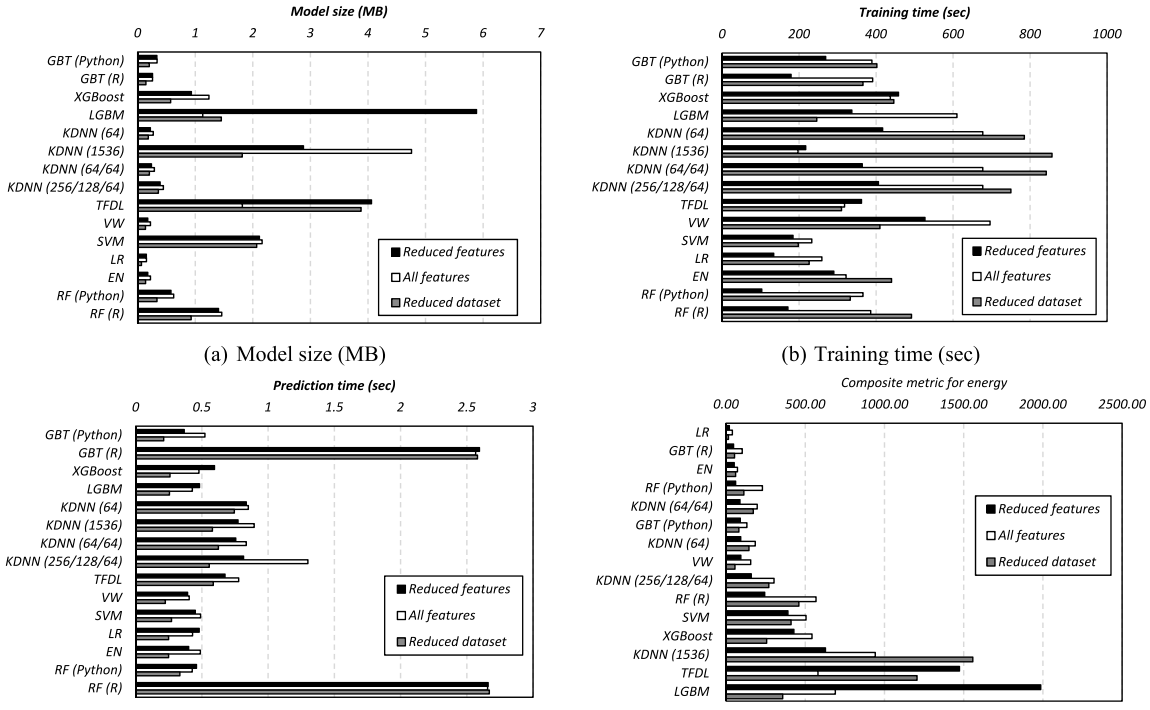


Fig. 4. Comparision of predictivity metric.

The highest performing model of the KDNN variant in the case of the full dataset was that of one layer and 1536 units, followed by two layers (64/64), three layers (256/128/64), and one layer of 64 units. Fig. 4 shows that the predictivity of the KDNN (256/128/64) variant significantly improved, bypassing all observed improvements of other models when applied to the case of reduced features. Furthermore, KDNN (64) seems to be the most stable variant in terms of predictivity. Acknowledging the limited number of programming languages and model architectures examined herein, it would be interesting to extend this investigation to other cases.

3.2. Insights from an energy perspective

The performance of the selected ML models was also examined from an energy perspective wherein three metrics were explored, namely, model size (in MB), training time (in seconds), and prediction time to score 1000 rows (in seconds). The outcome of this examination is shown



(a) Model size (MB)

(b) Training time (sec)

(c) Prediction time (sec)

(d) Composite metric [lower values are favorable]

Fig. 5. Comparison of metrics.

in Fig. 5 and Table 1. As one can see, the largest and smallest model size for the case of the full dataset was obtained from a KDNN (with 1536 units[7]) and LR, respectively. Overall, all ML models were of size within 0.152–4.757 MB and completed training within 197–696 s. On the other hand, prediction times varied between 0.404 and 2.657 s (with VW and RF being at these extremes, respectively).

Noting how faster training and prediction times and smaller model size (for the same workstation or cloud service) can be tied to lower energy expenditure, then all ML models were to be compared on this basis (Ba and Caruana, 2014; García-Martín et al., 2019; Frankle and Carbin, 2018). A new metric is then developed to account for the above rationale. This metric combines model size with training and prediction time such that:

$$Estimated\ Energy\ (E) = MS \times (TT + PT) \tag{5}$$

where, MS: model size, TT: training time, and PT: prediction time. Smaller values are favorable with a hypothetical minimum value = 1.0 MB × 10 s = 10 MB s.

Fig. 5 shows the results of applying this new metric. It is clear that the most energy efficient model is that of LR, followed by EN and the R version of GBT. In addition, once all models were normalized by the KDNN (with 1536 units), which happens to score the highest in the case of the full dataset analysis, one can appreciate the reduction of energy consumed by LR, which is estimated at 4.0% of that of the KDNN (with 1536 units).

3.2.1. Influence of dataset size

Surprisingly, reducing the dataset size was not always reflected by reducing model size, training time, or prediction time (see Fig. 5). While in large models such as KDNN (with 1536 units), this reduction results in a comparable reduction in model size, the same resulted in a significant increase in training time. In some instances, reducing the dataset size did not seem to affect model size much (i.e., SVM, KDNN (64, and 64/64)), nor prediction time (GBT (R), RF (R)). In all cases, the best performing models in terms of energy expenditure yielded the lowest composite metric, e.g., LR, GBT (R), and EN.

3.2.2. Influence of number of features

Fig. 5 shows that the selected ML models behaved differently in response to feature reduction. For example, the model size of LGBM significantly increased despite a 40–50% reduction in training time. TFDL also underwent a similar performance to that seen in LGBM from a model size point of view. Finally, some models seem to have higher stability across all scenarios (i.e., XGBoost, LR, EN, among others, as seen in Fig. 5d).

3.2.3. Influence of programing language

Unlike results from the conducted analysis in the case of predictivity, the influence of different programming languages was not as apparent when compared from an energy consumption perspective. In this event, GBT (of both Python and R versions) seems to achieve a lower energy consumption than that of the RF variants. In fact, the RF (R) variant is noted to consume about 64% times more energy than the Python variant (as opposed to the GBT (Python) variant needing 48% more energy than the R variant) when compared based on the full dataset. Collectively, the Python variants achieved a smaller model size and shorter prediction times than the R variants. When compared to the energy consumption of LR, both R and Python variants consumed a more considerable amount of energy in the range of 2.31–30.78 more than that of LR (with the Python variants scoring in the range of 2.95–7.50 and the R variants being on the higher side of the aforenoted range).

3.2.4. Influence of model architecture

The four previously discussed architectures of the KDNN model were also investigated to evaluate the influence of model architecture from an energy consumption perspective. As expected, our analysis indicates that simple architectures seem to consume less energy than those of deeper and heavier nature. More specifically, KDNN (64) and KDNN

[7] This model also happens to be the fastest training model.

(1536) scored an E metric of 196.57, and 941.39 (for full dataset), 144.58, and 1557.37 (for reduced dataset), and 92.34, and 627.19 (for reduced features). Similarly, KDNN (64) and KDNN (64/64) scored an E metric of 196.57 and 196.57 (for full dataset), 144.58, and 171.90 (for reduced dataset), and 92.34, and 87.18 (for reduced features). A close examination of the trends shown in Fig. 5 reveals that energy consumption is positively correlated to the total number of units in KDNN variants. Noting the LR scored the lowest score as per the E metric, then a comparison between KDNN variants and LR shows that these variants require 4.66–23.87%, 4.29–30.88%, and 9.68–104.31% more energy for the whole dataset, reduced dataset, and reduced features, respectively. To echo the previous section, future works are invited to further examine the performance of different architectures and programming languages.

3.3. Insights from a Holistic perspective

In order to tie the predictivity of ML models to their energy consumption, this section presents a visual aid employing a combined chart (see Fig. 6). This chart divides the models into four clusters and identifies models according to their combination of predictivity and energy consumption.

It is clear from Fig. 6 that RF (Python), GBT (Python), and GBT (R) consistently lay in the quadrant of high predictivity/low energy consumption. In addition, KDNN (1536) was found to reside in the low predictivity and high energy consumption, and the rest of the models were split into low predictivity/low energy consumption and high predictivity/high energy consumption. More exotic models, including XGBoost, LGBM, TFDL, and SVM, managed to cluster into one group (high predictivity/high energy consumption), and TFDL fell into the low predictivity/high energy consumption on two occasions.

It is worth noting that the XGBoost model achieved the highest predictivity in all cases and scored 0.58 on energy consumption (a slight 8% increase over the cut-off of 50%) in the case of the full dataset (while being on the lower energy side in the other two cases). As per the previous section, the LR model scored the lowest energy consumption in all cases.

A look into Fig. 6 further shows interesting observations with regard to the programing language used and model architecture. For example, both GBT variants landed on the orange cluster (high predictivity/low energy consumption). The R version of RF deviated from this trend, given its high predictivity score. In addition, most KDNN variants (with the exception of KDNN (1536)) were clustered in the red cluster (low predictivity and low energy consumption), with the common notion of all variants being on the low predictivity side. The trend of varying programming languages seems to be negligible in the case of the reduced dataset and reduced features. The variation of model architecture remains consistent in the cases of the reduced dataset and reduced features.

3.3.1. A rough estimation of carbon emission

The above investigation can be further used to *roughly* estimate the amount of carbon emissions generated to train the selected models, and then this estimation can be tied to the predictivity and energy consumption of each model. One must realize that the open literature does not seem to contain a straightforward method to relate energy consumption of model development to carbon emissions as model development is governed by complex factors and is highly dependent upon the combination of software/hardware (e.g., type and model of CPU, GPU, processing units, etc.), geographical location of workstations (wherein some regions advocate for carbon neutrality by imposing a "carbon tax" for carbon generation and capture while others do not), and resources used in generating electricity (renewable resources vs. nonrenewable resources), etc. (Strubell et al., 2020; García-Martín et al., 2019; Brownlee et al., 1763). As such, our analysis will be based on a hypothetical scenario that applies available values as collected from their original resources.

According to the U.S. Energy Information Administration, 0.42 Kg of CO2 is emitted per one kWh (EIA, 2021). A study by the Lawrence Berkeley National Laboratory, USA (Desroches et al., 2014), estimated the unit annual energy consumption (AEC) for desktops to be 194 kWh/yr (with a median = 125 kWh/yr), with 20% of desktops consuming more than 300 kWh/yr and high-end computing desktops reaching 600 kWh/yr. While most structural engineers will be using an above average desktop in their modeling or a cloud computing service, this analysis opts to adopt the above estimate of 300 kWh/yr. As such, a typical computer is expected to generate $0.42 \times 300 = 126$ Kg of CO2 annually.

While the number of structural engineers in the world is not well established, Barter (2014) notes that there are close to 50,000 structural engineers in the USA. Assuming that 2.5% of these engineers are proficient and dedicated ML users who use the above desktops, then applying the above calculation yields $0.025 \times 50{,}000 \times 126 = 126{,}000$ Kg of CO2 annually. This is equivalent to emissions generated from 27.4 passenger vehicles driven for one year (about 509,619.7 km) as per the published estimations made by the United States Environmental Protection Agency (EPA) (EPA, 2021).

Now, revisiting results obtained from Table 1 and Fig. 2 clearly shows that all selected models can be used to accurately predict if a concrete mixture can develop its design strength at in-situ conditions. However, a look at Fig. 5 infers the amount of possible energy savings that can be attained by using models of low energy consumption.

For illustration purposes, we take the highest energy consumption model (KDNN (1536)) in the case of the full dataset as a base for normalization, then a comparison between all models used in this particular case study can be established. This comparison is shown in Table 2 and Fig. 7 and infers that adopting LR, GBT (R), XGBoost, or LGBM can lead to 96%, 89%, 42%, and 27% reduction in CO2 as opposed to KDNN (1536). This clearly shows the significant reductions that can be attained by adopting simple models. Table 2 and Fig. 7 draw a similar comparison to the cases of the reduced dataset and the reduced features.

3.4. Insights into green ML

The presented analysis in this study highlights the need for advocating the adoption of responsible and Green ML early into fully integrating ML into the structural engineering domain. As such, one can be ready and prepared to maximize the benefits of ML while negating some of the complications that might arise (e.g., unresponsible energy consumption), which may delay harnessing the full potential of ML in the years to come.

The presented analysis can also be thought of as a first step towards a more articulated diagnostic investigation that dives in-depth to account for other dimensions that were not explored herein; especially those related to software/hardware, storage, and upkeep of models, data transfer, data type and size, type of observations, geographical location of working stations, resources used in generating electricity, governance and policy aspects, etc. In a future analysis, further details on the number of structural engineers (and the ratio of those who are practicing ML in their works), preferred working stations/cloud services, and use of centralized vs. distributed computing, to name a few, are to be investigated. The open literature does contain a few works tackling the accuracy-energy trade-off front by exploring the influence of much more complex ML models (i.e., deep learning and natural language processing), a similar series of investigations on such models derivatives that can be deployed in structural engineering problems are of value to explore (Jayakodi et al., 2018, 2020; Yang et al., 2018). One particular work of interest is that carried out by García-Martín et al. (2019) ,[8]

[8] Other works (Lacoste et al., 2019; Henderson et al., 2020), have made solid advancements on this front and are worthy of review.

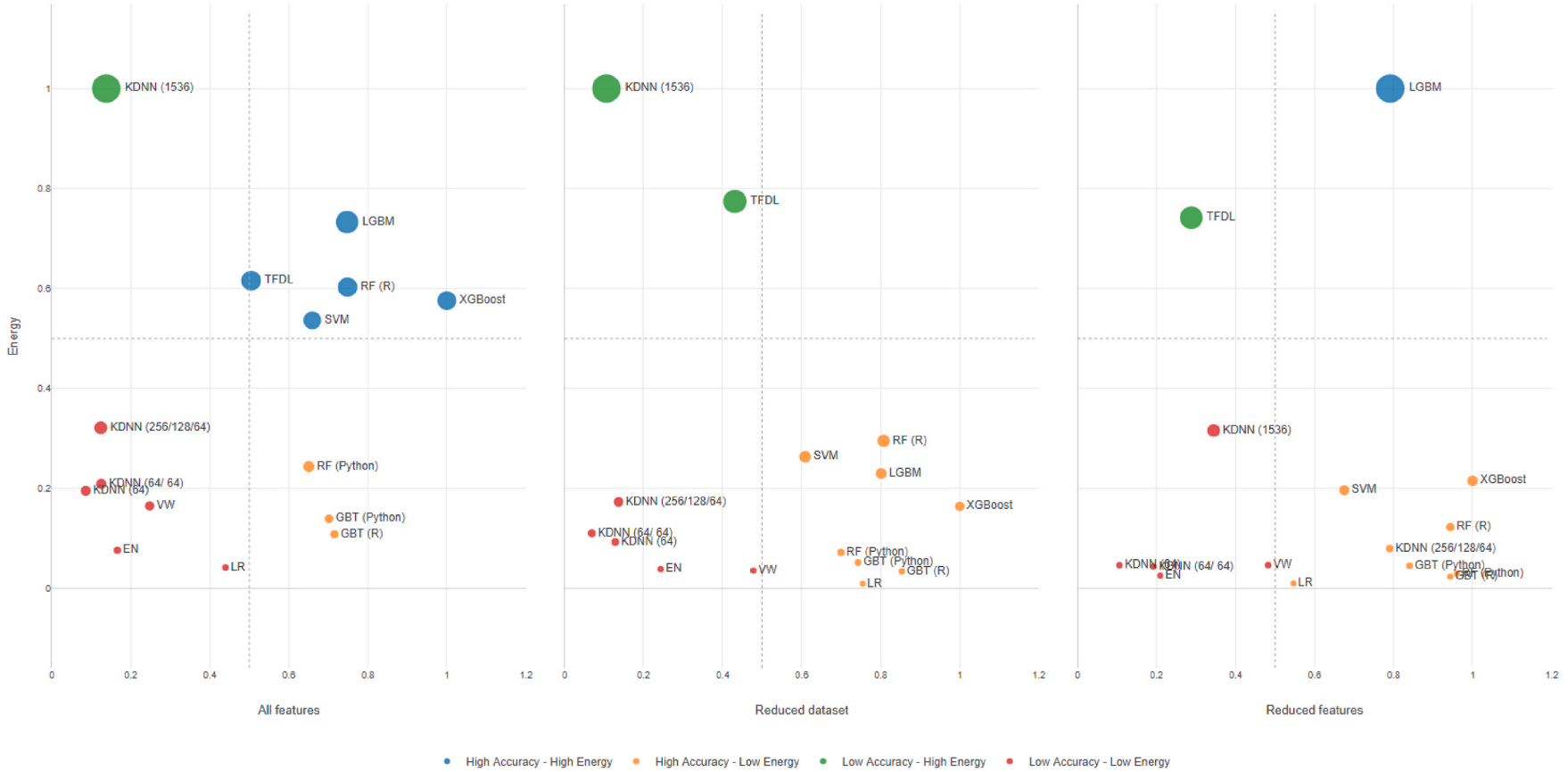


Fig. 6. Comparison of ML models when normalized by the highest attainable predictivity and energy consumption [size of the bubble represents the magnitude of estimated carbon emssisons].

Table 2
Comparison between ML models[a].

Model	*P* metric	*E* metric	Normalized *P*	Normalized *E*	% Reduction of max. CO2 emitted (per case)
All features					
LR	204.3	39.43	0.44	0.04	0.96
EN	77.2	71.92	0.17	0.08	0.92
GBT (R)	332.3	101.93	0.72	0.11	0.89
GBT (Python)	325.9	131.27	0.70	0.14	0.86
VW	115.4	155.30	0.25	0.16	0.84
KDNN (64)	40.0	183.70	0.09	0.20	0.80
KDNN (64/64)	58.3	196.57	0.13	0.21	0.79
RF (Python)	302.2	229.38	0.65	0.24	0.76
KDNN (256/128/64)	57.7	302.52	0.12	0.32	0.68
SVM	306.3	504.81	0.66	0.54	0.46
XGBoost	*464.5*	542.04	1.00	0.58	0.42
RF (R)	347.6	567.44	0.75	0.60	0.40
TFDL	234.5	579.54	0.50	0.62	0.38
LGBM	347.1	689.78	0.75	0.73	0.27
KDNN (1536)	64.0	*941.39*	0.14	1.00	0.00
Reduced dataset					
LR	227.9	14.93	0.75	0.01	0.99
EN	257.8	53.08	0.85	0.03	0.97
GBT (R)	73.5	60.31	0.24	0.04	0.96
GBT (Python)	211.3	112.00	0.70	0.07	0.93
VW	20.8	171.90	0.07	0.11	0.89
KDNN (64)	224.3	80.84	0.74	0.05	0.95
KDNN (64/64)	38.8	144.58	0.13	0.09	0.91
RF (Python)	144.3	55.79	0.48	0.04	0.96
KDNN (256/128/64)	41.4	269.45	0.14	0.17	0.83
SVM	244.0	459.55	0.81	0.30	0.70
XGBoost	183.8	409.83	0.61	0.26	0.74
RF (R)	*302.0*	255.71	1.00	0.16	0.84
TFDL	32.0	*1557.37*	0.11	1.00	0.00
LGBM	130.1	1205.38	0.43	0.77	0.23
KDNN (1536)	241.9	357.56	0.80	0.23	0.77
Reduced features					
LR	181.93	20.31	0.55	0.01	0.99
GBT (R)	314.03	46.85	0.94	0.02	0.98
EN	69.85	51.11	0.21	0.03	0.97
RF (Python)	319.65	59.90	0.96	0.03	0.97
KDNN (64/64)	64.12	87.18	0.19	0.04	0.96
GBT (Python)	279.91	89.70	0.84	0.05	0.95
KDNN (64)	35.50	92.34	0.11	0.05	0.95
VW	160.62	92.82	0.48	0.05	0.95
KDNN (256/128/64)	263.13	158.66	0.79	0.08	0.92
RF (R)	314.11	243.65	0.94	0.12	0.88
SVM	224.93	390.11	0.68	0.20	0.80
XGBoost	*332.93*	427.87	1.00	0.22	0.78
KDNN (1536)	114.73	627.19	0.34	0.32	0.68
TFDL	96.02	1472.82	0.29	0.74	0.26
LGBM	263.48	*1986.42*	0.79	1.00	0.00

[a] Values used for normalization in each particular case are underlined and in *italic*.

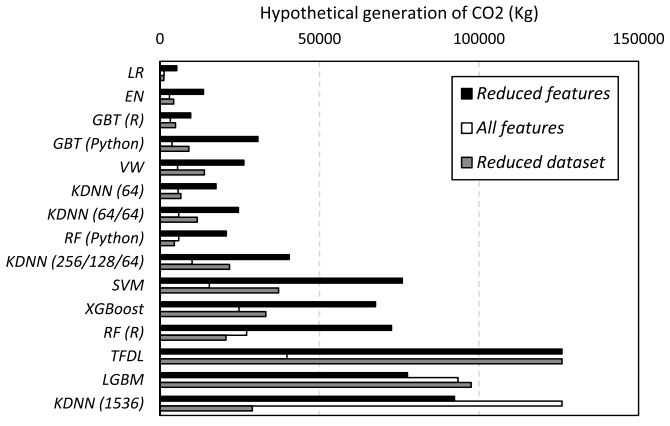


Fig. 7. Estimated carbon generated per ML model.

wherein this group presents options for hardware and software energy prediction methods – some of which can be used by structural engineers. These researchers also emphasized the need to investigate ML model energy consumption during the training phase and the inference phase.

In addition, a comparison between the energy consumption of traditional methods (such as FE modeling) to that of ML, when both are applied to the same problem, can indeed be interesting to investigate. While both methods can potentially use similar computing stations, it would be interesting to examine differences in setting up such stations to optimize their performance. Traditionally, FE models not only require large computational capacities but also necessitate the presence of large storage units, which can be a fraction of that required by a simple ML model for the same problem – especially if the FE model utilized finer mesh and nonlinear effects.

For example, a three-dimensional nonlinear FE model to predict the deformation history of reinforced concrete beams under fire conditions was developed by the author in earlier work. This model was composed of 50,125 elements and of a size of 64.58 MB (Hawileh et al., 2009). The results from running this model for a typical beam yielded a file of a size of 700 MB. In a separate work (Naser, 2019), a ML model was also

developed to predict the same phenomenon. This ML model was 305 KB. While both models were developed using different working stations approximately ten years apart, the truth is that both models still require storage. The American Council for an Energy-Efficient Economy estimates that it takes 5.12 kWh of electricity per gigabyte of transferred data. This implies that one GB (i.e., 1000 MB) emits 0.42 Kg $\times$ 5.12 kWh $=$ 2.15 Kg of CO2. Thus, carbon emissions arising solely from ***storing*** the FE model (without the result file), excluding those arising from model development and simulation time, are 215 times higher than that of the ML model.

Based on the presented analysis, the following are some of the big ideas that can be followed to create GML models:

- Care should be taken when developing new ML models. In all cases, engineers should consider the adequacy of simple models to tackle a problem instead of blindly selecting complex models. The same consideration can be applied to exploring issues arising from the accuracy-energy trade-off.
- Engineers are to consider how accurate and detailed their developed model should be. In a similar analogy to FE modeling, the adoption of a very detailed model involves a significant increase in the resources allocated to developing such a model, which yields higher monetary and environmental costs.
- Establish and aim for a "functional" level of model performance. Chasing accuracy metrics does not guarantee optimal performance, as such metrics reflect upon the model's performance against the available data and may not negate falling into model behavioral issues (e.g., overfitting, biasness, etc.).
- Explore pre-trained models and transfer learning techniques as a means to avoid the unnecessary retraining of models.
- Whenever possible, ML users are expected to share details of their conducted analysis and dataset and disclose energy performance metrics.
- Consider participating in carbon emission reduction initiatives by opting for computing services that employ environmental-friendly facilities.

4. Conclusions

This paper carries out a series of comparisons between simple and exotic ML algorithms to report on their predictive performance, model size, energy consumption, and storage need. More specifically, this work examines the influence of algorithm architecture, processing language, number of features, as well as dataset size on model predictivity, energy consumption, and expected carbon emissions. The following list of inferences can be drawn from the findings of this study:

- Structural engineers are expected to be cognizant of the accuracy-energy trade-offs when developing future ML models.
- Between 23 and 99% reduction in energy consumption and carbon emissions (while maintaining a similar level of accuracy) can be attained by adopting simple ML models and reduced-order modeling strategies.
- Of all examined ML models, the XGBoost ranked highest in terms of predictivity, and the LR ranked lowest in terms of energy consumption (and carbon emittance).
- There is an apparent influence on accuracy and energy consumption in similar ML models developed using different programming languages and ML models of the same origin but with different architectures. As such, care is needed when selecting such algorithms.
- Additional experiments specifically designed to examine the accuracy-energy trade-off in terms of system architecture, computing expenditure, etc., are warranted.
- Proactively setting the stage towards responsible and green ML is of merit to this community, and hence efforts in this direction are to be invited and appreciated.

CRediT authorship contribution statement

M.Z. Naser: Conceptualization, Methodology, Writing.

Declaration of competing interest

The author declare that he has no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

References

Abadi, M., 2016. TensorFlow: Learning Functions at Scale. ACM SIGPLAN Not. https://doi.org/10.1145/3022670.2976746.

Abubakar, A.U., Tabra, M.S., 2019. Prediction of compressive strength in high performance concrete with hooked-end steel fiber using K-nearest neighbor algorithm. Int. J. Integr. Eng. https://doi.org/10.30880/ijie.2019.11.01.016.

Adeli, H., 2001. Neural networks in civil engineering: 1989-2000. Comput. Civ. Infrastruct. Eng. https://doi.org/10.1111/0885-9507.00219.

Alavi, A.H., Gandomi, A.H., Sahab, M.G., Gandomi, M., 2010. Multi expression programming: a new approach to formulation of soil classification. Eng. Comput. 26, 111–118. https://doi.org/10.1007/s00366-009-0140-7.

Alwanas, A.A.H., Al-Musawi, A.A., Salih, S.Q., Tao, H., Ali, M., Yaseen, Z.M., 2019. Load-carrying capacity and mode failure simulation of beam-column joint connection: application of self-tuning machine learning model. Eng. Struct. https://doi.org/10.1016/j.engstruct.2019.05.048.

Ashrafian, A., Shokri, F., Taheri Amiri, M.J., Yaseen, Z.M., Rezaie-Balf, M., 2020. Compressive strength of Foamed Cellular Lightweight Concrete simulation: new development of hybrid artificial intelligence model. Construct. Build. Mater. https://doi.org/10.1016/j.conbuildmat.2019.117048.

Ashteyat, A., Obaidat, Y.T., Murad, Y.Z., Haddad, R., 2020. Compressive strength prediction of lightweight short columns at elevated temperature using gene expression programing and artificial neural network. J. Civ. Eng. Manag. https://doi.org/10.3846/jcem.2020.11931.

Avci, O., Abdeljaber, O., Kiranyaz, S., Hussein, M., Gabbouj, M., Inman, D.J., 2021. A review of vibration-based damage detection in civil structures: from traditional methods to Machine Learning and Deep Learning applications. Mech. Syst. Signal Process. https://doi.org/10.1016/j.ymssp.2020.107077.

Ba, L.J., Caruana, R., 2014. Do deep nets really need to be deep? Adv. Neural Inf. Process. Syst. 27, 1–9.

Babanajad, S.K., Gandomi, A.H., Alavi, A.H., 2017. New prediction models for concrete ultimate strength under true-triaxial stress states: an evolutionary approach. Adv. Eng. Software. https://doi.org/10.1016/j.advengsoft.2017.03.011.

Barter, M., 2014. 10 Obstacles to Meaningful Licensing of Structural Engineers, Struct. Mag. https://www.structuremag.org/?p=4039. (Accessed 28 April 2021).

Biswas, R., Rai, B., Samui, P., Roy, S.S., 2020. Estimating concrete compressive strength using MARS. LSSVM and GP, Eng. J. https://doi.org/10.4186/ej.2020.24.2.41.

Boser, B.E., Guyon, I.M., Vapnik, V.N., 1992. Training algorithm for optimal margin classifiers. In: Proc. Fifth Annu. ACM Work. Comput. Learn. Theory. https://doi.org/10.1145/130385.130401.

Breiman, L., 2001. randomForest: Breiman and cutler's random forests for classification and regression. Mach. Learn. 45, 5–32. https://doi.org/10.1023/A:1010933404324.

Brownlee, A.E.I., Adair, J., Haraldsson, S.O., Jabbo, J.. Exploring the accuracy-energy trade-off in machine learning. n.d. https://www.kaggle.com/dinu1763/mortgage-loan-approval. (Accessed 28 April 2021).

Çevik, A., Kurtoğlu, A.E., Bilgehan, M., Gülşan, M.E., Albegmprli, H.M., 2015. Support Vector Machines in Structural Engineering: A Review. https://doi.org/10.3846/13923730.2015.1005021.

Chen, T., Guestrin, C., 2016. XGBoost: a scalable tree boosting system. In: Proc. ACM SIGKDD Int. Conf. Knowl. Discov. Data Min. https://doi.org/10.1145/2939672.2939785.

Cheng, J.C.P., Chen, W., Chen, K., Wang, Q., 2020. Data-driven predictive maintenance planning framework for MEP components based on BIM and IoT using machine learning algorithms. Autom. ConStruct. https://doi.org/10.1016/j.autcon.2020.103087.

Degtyarev, V.V., 2021. Neural networks for predicting shear strength of CFS channels with slotted webs. J. Constr. Steel Res. https://doi.org/10.1016/j.jcsr.2020.106443.

Desroches, L., Fuch, H., Greenblatt, J., Pratt, S., Willem, Hcl, Beraki, B., Nagaraju, M., Price, S., Young, S., 2014. Computer Usage and National Energy Consumption: Results from a Field-Metering Study. https://www.osti.gov/servlets/purl/1166988. (Accessed 28 April 2021).

Diez, A., Khoa, N.L.D., Makki Alamdari, M., Wang, Y., Chen, F., Runcie, P., 2016. A clustering approach for structural health monitoring on bridges. J. Civ. Struct. Heal. Monit. 6, 429–445. https://doi.org/10.1007/s13349-016-0160-0.

D'Amico, B., Myers, R.J., Sykes, J., Voss, E., Cousins-Jenvey, B., Fawcett, W., Richardson, S., Kermani, A., Pomponi, F., 2019. Machine Learning for Sustainable

Structures: A Call for Data. Structures. https://doi.org/10.1016/j.istruc.2018.11.013.
EIA, 2021. Frequently Asked Questions (FAQs) - U.S. Energy Information Administration (EIA), U.S. Energy Inf. Adm. https://www.eia.gov/tools/faqs/faq.php?id=74&t=11. (Accessed 28 April 2021).
EPA, 2021. Greenhouse Gas Equivalencies Calculator. EPA. https://www.epa.gov/energy/greenhouse-gas-equivalencies-calculator. (Accessed 28 April 2021).
Farfan, J., Fasihi, M., Breyer, C., 2019. Trends in the global cement industry and opportunities for long-term sustainable CCU potential for Power-to-X. J. Clean. Prod. https://doi.org/10.1016/j.jclepro.2019.01.226.
Farrar, C.R., Worden, K., 2012. Structural Health Monitoring: A Machine Learning Perspective. https://doi.org/10.1002/9781118443118.
Frankle, J., Carbin, M., 2018. The Lottery Ticket Hypothesis: Finding Sparse, Trainable Neural Networks. ArXiv.
Freischlad, M., Schnellenbach-Held, M., 2005. A machine learning approach for the support of preliminary structural design. Adv. Eng. Inf. https://doi.org/10.1016/j.aei.2005.07.001.
Fu, F., 2020. Fire induced progressive collapse potential assessment of steel framed buildings using machine learning. J. Constr. Steel Res. https://doi.org/10.1016/j.jcsr.2019.105918.
García-Martín, E., Rodrigues, C.F., Riley, G., Grahn, H., 2019. Estimation of energy consumption in machine learning. J. Parallel Distr. Comput. https://doi.org/10.1016/j.jpdc.2019.07.007.
Gola, J., Webel, J., Britz, D., Guitar, A., Staudt, T., Winter, M., Mücklich, F., 2019. Objective microstructure classification by support vector machine (SVM) using a combination of morphological parameters and textural features for low carbon steels. Comput. Mater. Sci. https://doi.org/10.1016/j.commatsci.2019.01.006.
Golafshani, E.M., Behnood, A., 2019. Estimating the optimal mix design of silica fume concrete using biogeography-based programming. Cem. Concr. Compos. 96, 95–105. https://doi.org/10.1016/J.CEMCONCOMP.2018.11.005.
Hasni, H., Alavi, A.H., Lajnef, N., Abdelbarr, M., Masri, S.F., Chakrabartty, S., 2017. Self-powered piezo-floating-gate sensors for health monitoring of steel plates. Eng. Struct. https://doi.org/10.1016/j.engstruct.2017.06.063.
Hawileh, R.A., Naser, M.Z., Zaidan, W., Rasheed, H.A., 2009. Modeling of insulated CFRP-strengthened reinforced concrete T-beam exposed to fire. Eng. Struct. 31, 3072–3079. https://doi.org/10.1016/j.engstruct.2009.08.008.
Henderson, P., Hu, J., Romoff, J., Brunskill, E., Jurafsky, D., Pineau, J., 2020. Towards the systematic reporting of the energy and carbon footprints of machine learning. J. Mach. Learn. Res. 21, 1–43.
Huang, H., Burton, H.V., 2019. Classification of in-plane failure modes for reinforced concrete frames with infills using machine learning. J. Build. Eng. https://doi.org/10.1016/j.jobe.2019.100767.
Iandola, F.N., Han, S., Moskewicz, M.W., Ashraf, K., Dally, W.J., Keutzer, K., 2016. SqueezeNet: AlexNet-Level Accuracy with 50x Fewer Parameters and <0.5MB Model Size. https://doi.org/10.48550/arxiv.1602.07360.
Jackson, R., 2019. A Roadmap to Reducing Greenhouse Gas Emissions 50 Percent by 2030. https://earth.stanford.edu/news/roadmap-reducing-greenhouse-gas-emissions-50-percent-2030#gs.zij3zf.
Jayakodi, N.K., Chatterjee, A., Choi, W., Doppa, J.R., Pande, P.P., 2018. Trading-off accuracy and energy of deep inference on embedded systems: a co-design approach. IEEE Trans. Comput. Des. Integr. Circuits Syst. https://doi.org/10.1109/TCAD.2018.2857338.
Jayakodi, N.K., Belakaria, S., Deshwal, A., Doppa, J.R., 2020. Design and optimization of energy-accuracy tradeoff networks for mobile platforms via pretrained deep models. ACM Trans. Embed. Comput. Syst. https://doi.org/10.1145/3366636.
Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., Ye, Q., Liu, T.Y., 2017. LightGBM: a highly efficient gradient boosting decision tree. Adv. Neural Inf. Process. Syst. 30, 1–9.
Keras, 2020. GitHub - Keras-Team/keras: Deep Learning for Humans. https://github.com/keras-team/keras. (Accessed 9 February 2021).
Ketkar, N., Ketkar, N., 2017. Introduction to Keras. In: Deep Learn. With Python. https://doi.org/10.1007/978-1-4842-2766-4_7.
Lacoste, A., Luccioni, A., Schmidt, V., Dandres, T., 2019. Quantifying the Carbon Emissions of Machine Learning. https://doi.org/10.48550/arxiv.1910.09700.
Laszczyk, M., Myszkowski, P.B., 2019. Survey of quality measures for multi-objective optimization: construction of complementary set of multi-objective quality measures. Swarm Evol. Comput. 48, 109–133. https://doi.org/10.1016/J.SWEVO.2019.04.001.
Li, H., Xu, Z., Taylor, G., Studer, C., Goldstein, T., 2018. Visualizing the loss landscape of neural nets. Adv. Neural Inf. Process. Syst. 31, 1–11.
Liao, W., Huang, Y., Zheng, Z., Lu, X., 2022. Intelligent generative structural design method for shear wall building based on "fused-text-image-to-image" generative adversarial networks. Expert Syst. Appl. 210, 118530 https://doi.org/10.1016/J.ESWA.2022.118530.
Liaw, A., Wiener, M., 2002. Classification and Regression by RandomForest. https://www.researchgate.net/publication/228451484. (Accessed 8 April 2019).
LightGBM, 2020. Welcome to LightGBM's Documentation! — LightGBM 3.1.1.99 Documentation. https://lightgbm.readthedocs.io/en/latest/. (Accessed 9 February 2021).
Lu, Y., Zhang, A., Li, Q., Dong, B., 2018. Beyond Finite Layer Neural Networks: Bridging Deep Architectures and Numerical Differential Equations | OpenReview, ICLR 2018 Conf. https://openreview.net/forum?id=ryZ283gAZ. (Accessed 23 April 2021).
Lundberg, S.M., Lee, S.I., 2017. A unified approach to interpreting model predictions. Adv. Neural Inf. Process. Syst. 30, 1–10.
Luor, D.C., 2015. A comparative assessment of data standardization on support vector machine for classification problems. Intell. Data Anal. https://doi.org/10.3233/IDA-150730.
Mahendran, M., 2015. Applications of finite element analysis in structural engineering. Int. Conf. Comput. Aided Eng. 38–46.
Mangalathu, S., Burton, H.V., 2019. Deep learning-based classification of earthquake-impacted buildings using textual damage descriptions. Int. J. Disaster Risk Reduc. 36, 101111 https://doi.org/10.1016/j.ijdrr.2019.101111.
Marsland, S., 2014. Machine Learning: an Algorithmic Perspective. https://doi.org/10.1201/b17476.
Mohammadi, M., Al-Fuqaha, A., 2018. Enabling cognitive smart cities using big data and machine learning: approaches and challenges. IEEE Commun. Mag. https://doi.org/10.1109/MCOM.2018.1700298.
Naser, M.Z., 2019. AI-based cognitive framework for evaluating response of concrete structures in extreme conditions. Eng. Appl. Artif. Intell. 81, 437–449. https://www.sciencedirect.com/science/article/pii/S0952197619300466. (Accessed 1 April 2019).
Naser, M.Z., 2020a. Machine learning assessment of fiber-reinforced polymer-strengthened and reinforced concrete members. ACI Struct. J. https://doi.org/10.14359/51728073.
Naser, M.Z., 2020b. The role of computational intelligence in realizing modern and autonomous fire evaluation methods. In: Struct. Congr. 2020 - Sel. Pap. From Struct. Congr. 2020. https://doi.org/10.1061/9780784482896.059.
Naser, M.Z., 2021. Mechanistically informed machine learning and artificial intelligence in fire engineering and sciences. Fire Technol. 1–44. https://doi.org/10.1007/s10694-020-01069-8.
Naser, M., 2022a. A faculty's perspective into infusing artificial intelligence to civil engineering education. J. Civ. Eng. Educ. https://doi.org/10.1061/(ASCE)EI.2643-9115.0000065.
Naser, M.Z., 2022b. Digital twin for next gen concretes: on-demand tuning of vulnerable mixtures through Explainable and Anomalous Machine Learning. Cem. Concr. Compos. 132, 104640 https://doi.org/10.1016/J.CEMCONCOMP.2022.104640.
M.Z.Z. Naser, A. Seitllari, Concrete under fire: an assessment through intelligent pattern recognition, Eng. Comput. 36 1–14. https://doi.org/10.1007/s00366-019-00805-1.
Natekin, A., Knoll, A., 2013. Gradient boosting machines, a tutorial. Front. Neurorob. 7 https://doi.org/10.3389/fnbot.2013.00021.
Pan, Y., Zhang, L., 2021. Roles of artificial intelligence in construction engineering and management: a critical review and future trends. Autom. ConStruct. https://doi.org/10.1016/j.autcon.2020.103517.
Panev, Y., Kotsovinos, P., Deeny, S., Flint, G., 2021. The Use of Machine Learning for the Prediction of Fire Resistance of Composite Shallow Floor Systems. Fire Technol. https://doi.org/10.1007/s10694-021-01108-y.
Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, É., Duchesnay, E., 2011. Scikit-learn: machine learning in Python. J. Mach. Learn. Res. 12, 2825–2830.
Rafiei, M.H., Adeli, H., 2017. A novel machine learning-based algorithm to detect damage in high-rise building structures. Struct. Des. Tall Special Build. https://doi.org/10.1002/tal.1400.
Ritchie, H., Roser, M., 2021. Emissions by Sector - Our World in Data. https://ourworldindata.org/emissions-by-sector. (Accessed 28 April 2021).
Schmidt, M.D., Lipson, H., 2010. Age-fitness Pareto Optimization. https://doi.org/10.1145/1830483.1830584.
Scikit, 2020a. sklearn.ensemble.GradientBoostingRegressor — Scikit-Learn 0.24.1 Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html. (Accessed 9 February 2021).
Scikit, 2020b. sklearn.ensemble.RandomForestClassifier — Scikit-Learn 0.24.1 Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. (Accessed 9 February 2021).
Scikit. sklearn.ensemble.GradientBoostingClassifier — scikit-learn 0.24.1 documentation (n.d.). https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html. (Accessed 26 April 2021).
Scikit, 2021b. sklearn.kernel_approximation.Nystroem — Scikit-Learn 0.24.1 Documentation. https://scikit-learn.org/stable/modules/generated/sklearn.kernel_approximation.Nystroem.html. (Accessed 27 April 2021).
Scikit, 2021c. lightning.classification.CDClassifier — Lightning Dev Documentation. http://contrib.scikit-learn.org/lightning/generated/lightning.classification.CDClassifier.html. (Accessed 27 April 2021).
Smarter, G., 2020. GeSI SMARTer 2020: the Role of ICT in Driving a Sustainable Future GeSI SMARTer 2020 2 the Role of ICT in Driving a Sustainable Future Independent Review by.
Strubell, E., Ganesh, A., McCallum, A., 2020. Energy and policy considerations for modern deep learning research. Proc. AAAI Conf. Artif. Intell. https://doi.org/10.1609/aaai.v34i09.7123.
Sun, H., Burton, H.V., Huang, H., 2021. Machine learning applications for building structural design and performance assessment: state-of-the-art review. J. Build. Eng. 33, 101816 https://doi.org/10.1016/j.jobe.2020.101816.
Taffese, W.Z., Sistonen, E., 2017. Machine learning for durability and service-life assessment of reinforced concrete structures: recent advances and future directions. Autom. ConStruct. https://doi.org/10.1016/j.autcon.2017.01.016.
Tapeh, A., Naser, M.Z., 2022. Artificial intelligence, machine learning, and deep learning in structural engineering: a scientometrics review of trends and best practices. Arch. Comput. Methods Eng. https://doi.org/10.1007/s11831-022-09793-w.
Tarawneh, A.N., Tarawneh, A.N., Tarawneh, A.N., Ross, B.E., Ross, B.E., Ross, B.E., Cousins, T.E., Cousins, T.E., Cousins, T.E., 2020. Shear behavior and design of post-

installed anchors in thin concrete members. ACI Struct. J. https://doi.org/10.14359/51723508.
TensorFlow, 2020. GitHub - Tensorflow/tensorflow: an Open Source Machine Learning Framework for Everyone. https://github.com/tensorflow/tensorflow. (Accessed 9 February 2021).
Thai, H.T., 2022. Machine Learning for Structural Engineering: A State-Of-The-Art Review, Structures. https://doi.org/10.1016/j.istruc.2022.02.003.
VowpalWabbit, 2021. vowpalwabbit.sklearn — VowpalWabbit 8.11.0 Documentation. https://vowpalwabbit.org/docs/vowpal_wabbit/python/latest/vowpalwabbit.sklearn.html. (Accessed 26 April 2021).
Wang, Y., Sun, S., Chen, X., Zeng, X., Kong, Y., Chen, J., Guo, Y., Wang, T., 2021. Short-term load forecasting of industrial customers based on SVMD and XGBoost. Int. J. Electr. Power Energy Syst. 129, 106830 https://doi.org/10.1016/j.ijepes.2021.106830.
XGBoost Python Package, 2020. Python Package Introduction — Xgboost 1.4.0-SNAPSHOT Documentation. https://xgboost.readthedocs.io/en/latest/python/python_intro.html#early-stopping. (Accessed 10 February 2021).
Xie, Y., Ebad Sichani, M., Padgett, J.E., DesRoches, R., 2020. The promise of implementing machine learning in earthquake engineering: a state-of-the-art review. Earthq. Spectra. https://doi.org/10.1177/8755293020919419.
Yang, T.J., Chen, Y.H., Emer, J., Sze, V., 2018. A method to estimate the energy consumption of deep neural networks. In: Conf. Rec. 51st Asilomar Conf. Signals. Syst. Comput. ACSSC 2017. https://doi.org/10.1109/ACSSC.2017.8335698.
Young, B.A., Hall, A., Pilon, L., Gupta, P., Sant, G., 2019. Can the compressive strength of concrete be estimated from knowledge of the mixture proportions?: new insights from statistical analysis and machine learning methods. Cement Concr. Res. 115, 379–388. https://doi.org/10.1016/j.cemconres.2018.09.006.
Yu, A., 2021. hyperparameter_tune/Clean_data.csv at Master · Huiziy/hyperparameter_tune · GitHub. https://github.com/huiziy/hyperparameter_tune/blob/master/Clean_data.csv. (Accessed 18 March 2021).
Ziegel, E.R., 2003. The Elements of Statistical Learning. Technometrics. https://doi.org/10.1198/tech.2003.s770.