Computer-Aided Design 131 (2021) 102948

Contents lists available at ScienceDirect

Computer-Aided Design

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

Geometric Stability Classification: Datasets, Metamodels, and Adversarial Attacks

Emma Lejeune

Department of Mechanical Engineering, Boston University, Boston, MA 02215, United States of America

ARTICLE INFO

Article history:
Received 27 April 2020
Received in revised form 10 August 2020
Accepted 20 September 2020

Keywords:
Machine learning
Buckling
Open data

ABSTRACT

Many recent advances in machine learning have been motivated by classification problems. For example, classification methods are used to differentiate between "spam" and "non-spam" emails, identify hand written digits, and recognize the content of photos. For each application, a different model and model architecture will often perform best. Therefore, machine learning research has been enabled by readily available benchmark datasets. In particular, benchmark datasets have been used by researchers to demonstrate that novel methods can achieve high accuracy, and to demonstrate common vulnerabilities of classification methods to adversarial attacks. In the recent mechanics literature, there has been substantial interest in machine learning driven metamodels. Metamodels, or models of models, are appealing because once trained, they typically require orders of magnitude less compute time than full fidelity simulations. However, a better understanding of which machine learning methods and model architectures will perform best on mechanical data has been limited. Here we introduce an open source dataset "BIC" (Buckling Instability Classification) where a heterogeneous column is subject to a fixed level of applied displacement and is classified as either "Stable" or "Unstable." In addition to introducing this benchmark dataset, we show baseline metamodel performance, and show two different types of adversarial attack. We anticipate that the open source BIC dataset will enable the future development of improved methods for classification problems in mechanics.

1. Introduction

Statistical classification in machine learning is a type of supervised learning where the goal is to predict what class a new example will belong to [1,2]. Many recent advances in machine learning have been motivated by statistical classification [3]. For example, classification models are used to differentiate between "spam" and "non-spam" emails [4], identify hand written digits in zip codes [5], recognize the subject of photos [6], and predict disease risk [7]. Critically, pragmatic advances in machine learning methods have been enabled by the availability of benchmark datasets [8], and the efficacy of new methods is often demonstrated by reporting model performance on a benchmark dataset [9]. Likewise, the shortcomings of different methods such as sensitivity to noise or susceptibility to adversarial attacks are often showcased by demonstrations on benchmark datasets [10]. For each type of data (ex: email text, facial images), a different model and model architecture will often perform best [11]. And, each type of problem has unique associated vulnerabilities [12].

In the mechanical design literature, machine learning methods have become increasingly popular [13–16]. In particular, there has been substantial recent literature that relies on the use of metamodels [17–24]. Metamodels, sometimes referred to as surrogate models, are computationally cheap models of models that relate specifically defined model inputs to specifically defined model outputs [25,26]. The computationally cheap nature of metamodels has enabled sensitivity analysis, optimization, and multiscale modeling beyond what would be feasible with direct numerical simulation alone [27–29]. However, the lack of benchmark data has meant that tools for pragmatic metamodel development specific to mechanics and mechanical design problems have been largely siloed [30]. In this work, we present an open source mechanics-based classification dataset with the intention that it will first aid in pragmatic metamodel development, and second, offer a platform to demonstrate vulnerabilities specifically relevant to mechanics problems.

Our open source dataset "BIC" (Buckling Instability Classification) contains finite element simulation inputs and results for a heterogeneous column subject to a fixed level of applied displacement [31]. For each pattern of heterogeneous material properties, the column is classified as either "Stable" or "Unstable." We generate three datasets that differ only in the patterns allowed in their input domain (BIC-1, BIC-2, and BIC-3). This is illustrated in Fig. 1. Then, we investigate three methods for metamodeling:

E-mail address: elejeune@bu.edu.

https://doi.org/10.1016/j.cad.2020.102948
0010-4485/© 2020 Elsevier Ltd. All rights reserved.

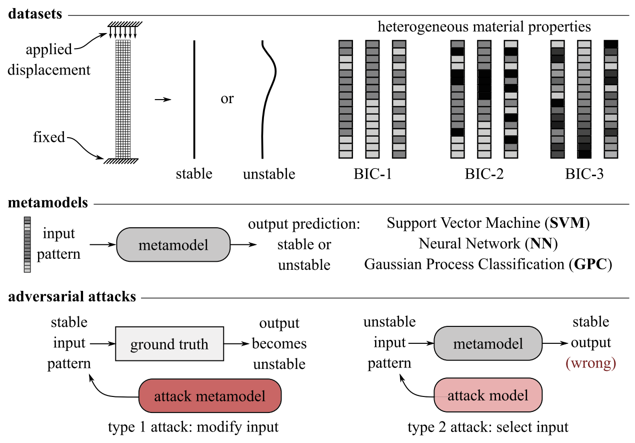

Fig. 1. In the BIC datasets, columns with heterogeneous material properties are subject to a fixed level of applied displacement and are classified as either "stable" or "unstable." Each BIC input is a 16×1 vector that informs the material stiffness along the length of the column. In BIC-1, there are two discrete options. In BIC-2, there are three discrete options. In BIC-3, there is a continuous stiffness range. Here we show the performance of three different types of metamodel on each dataset. Then, we show two ways that metamodels can be used to launch adversarial attacks. In a type 1 attack, an attack metamodel is used to find unstable inputs that are very similar to stable inputs. In a type 2 attack, an attack model finds inputs that a trained metamodel will erroneously label as stable.

support vector machines, neural networks, and Gaussian process classification. Finally, we define two types of adversarial attack, illustrated in Fig. 1, and evaluate the performance of these attacks on the BIC-1 dataset. Notably, we show that of all the machine learning models tested, Gaussian process classification is able to achieve the best performance. However, despite near perfect performance, the results of our adversarial attacks show that there is room for future improvement.

The remainder of the paper is organized as follows. In Section 2 we describe the finite element simulations used to generate our benchmark dataset, introduce three baseline methods for creating classification metamodels, and define two types of adversarial attacks that are relevant to this mechanical problem. In Section 3, we show baseline metamodel performance, and baseline attack results. Finally, we conclude in Section 4 with some examples of future investigation that will be directly enabled by this work. In Section 5, details for accessing the BIC datasets and all code relevant to this paper are provided.

2. Methods

In Section 2.1, we describe the finite element simulations used to generate the BIC datasets. In Section 2.2, we describe three baseline classification metamodels applied to the BIC datasets: support vector machines, neural networks, and Gaussian process classification. Then, in Section 2.3, we introduce the concept of type 1 and type 2 adversarial attacks and describe why BIC-1 is a useful dataset for exploring adversarial attacks related to mechanics-based classification problems. Notably, all work described here is performed with open source software, and details for accessing the relevant code are provided in Section 5.

2.1. Simulation overview

The basic premise of the BIC datasets is shown in Fig. 1. In this Section, we more formally define the problem, describe the finite element simulations required to solve the problem, and describe how the simulation inputs and outputs are represented in the BIC datasets.

2.1.1. Problem definition

To generate the BIC datasets, we model heterogeneous columns and subject each to applied compression through a fixed level of applied displacement. Then, we determine if the column has exceeded its critical load at the level of applied displacement and denote it as either "stable" or "unstable." Each column has a width h of 2 units, and a height L of 16 units. The material properties are piecewise constant and vary along the height of the column following the prescription in the BIC input vector. This is schematically illustrated in Fig. 1. The bottom of the column is fixed (Dirichlet boundary condition), the left and right edges of the column are free, and the top of the domain is moved to a fixed level of applied displacement. In all cases, the fixed level of applied displacement d_{applied} is equal to

$$d_{\text{applied}} = -1.025\, L\, \varepsilon_{\text{cr}} \qquad \text{where} \qquad \varepsilon_{\text{cr}} = \frac{\pi^2 h^2}{3L^2} \tag{1}$$

which corresponds to the critical Euler buckling strain for a homogeneous column with identical dimensions [32].

For all BIC datasets, we consider a compressible neo-Hookean material with strain energy function

$$\psi = \frac{1}{2}\mu\left[\mathbf{F}:\mathbf{F} - 3 - 2\ln(\det\mathbf{F})\right] + \frac{1}{2}\lambda\left[\frac{1}{2}((\det\mathbf{F})^2 - 1) - \ln(\det\mathbf{F})\right] \tag{2}$$

where ψ is strain energy, $\mathbf{F}$ is the deformation gradient, and μ and λ are Lamé parameters equivalent to Young's modulus E and Poisson's ratio ν:

$$E = \frac{\mu\,(3\lambda + 2\mu\,)}{\lambda + \mu} \qquad \nu = \frac{\lambda}{2\,(\,\lambda + \mu\,)}\,. \tag{3}$$

Because the columns have heterogeneous material properties, some are unstable at d_{applied} while others are not.

2.1.2. Finite element implementation

In the finite element setting, we discretize the columns with a 960×120 structured mesh of quadratic triangular elements. This mesh size was deemed sufficient after mesh refinement studies. To determine if the column is unstable at d_{applied}, we follow a well established procedure [33,34]. First, we solve the finite element problem with a nonlinear variational solver. Then, we compute the smallest eigenvalue of the corresponding stiffness matrix. If the smallest eigenvalue is less than 0, d_{applied} exceeds the critical buckling condition and the column is marked "unstable" [35]. Otherwise, the structure is marked as "stable." All simulations are run in FEniCS [36,37] and all eigenvalue analysis is performed using PETSc data structures and the SLEPc library [38–41]. These software tools are all freely available.

2.1.3. Data curation

Our open source datasets, BIC-1, BIC-2, and BIC-3, differ only in the scope of allowable material properties. In other words, the difference between each dataset is the constraints placed on the input domain. For all datasets, each input is a 16×1 vector where the entries of the vector dictate the Young's Modulus E of the corresponding portion of the physical column domain. In BIC-1, we only allow two possible discrete values for E: $E = 1$ or $E = 4$. In BIC-2, we allow three possible discrete values for E: $E = 1$, $E = 4$, or $E = 7$. In BIC-3, we allow continuous values (to three digits of precision) of E in the range $E = 1 - 8$. For all simulations, $\nu = 0.3$. BIC-1 consists of 65,536 simulation results. This exhausts the entire possible input domain. BIC-2 consists of 100,000 simulation results. This is less than 1% of the entire possible input domain. BIC-3 also consists of 100,000 simulation results. This is a tiny fraction of the entire possible input domain.

We note that for BIC-2 and BIC-3, input parameters are selected randomly, and the results presented in Section 3 are thus based on a random sampling approach. It is possible that an alternative approach, such as Latin Hypercube Sampling [42,43] or a Sobol sequence [44] would subsequently improve model performance.

For each 16×1 vector input, there is a single output that indicates if the column was stable or unstable at the fixed level of applied displacement. An output value of 0 indicates stable, and an output value of 1 indicates unstable. For the purpose of presenting the metamodel and attack model results in Section 3 (where we report true positive rate and false positive rate), unstable is a "positive" result and "stable" is a "negative" result. In the version of BIC that we release open source, we randomly separate the data into test and training sets where the test sets contain 10% of the data. We also note that the input values are standardized across datasets. This means that the BIC datasets or parts of the BIC datasets can be combined without any additional processing.

2.1.4. Note on symmetry

Due to the problem definition, vector inputs will yield identical outputs to their own reflection. Therefore, when generating BIC-1, we only simulated one of each symmetric pair and subsequently matched the simulation output to both pairs. This means that we ran 32,896 simulations to cover the whole domain rather than the 65,536 simulations required without accounting for symmetry. Although we do not propose metamodels that are constrained to give identical results for symmetric input domains in this paper, it is our hope that the open source nature of the BIC dataset will enable future exploration in this direction.

2.2. Baseline metamodels

At present, there are multiple available methods for approaching classification problems. Typically, method selection is pragmatic and dataset dependent [11]. In the mechanics literature specifically, multiple different methods of creating computationally cheap metamodels have been explored [45–47]. Here we introduce three popular methods for classification: support vector machines, neural networks, and Gaussian process classification [11,48]. In Section 3.1 we show baseline performance of each method on the BIC-1, BIC-2, and BIC-3 datasets.

2.2.1. Support vector machines

Support vector machines are a popular method for classification problems [49]. Support vector machines function by constructing hyperplanes in high dimensional space that separate the data according to predicted class [50]. Training a support vector machine model involves algorithmically selecting a hyperplane, defined by "support vectors", that separates the two classes of data as represented in the high dimensional space. Here we implement our support vector machine model with the python scikit-learn support vector classification module [51]. We use the default function parameters with a radial basis function kernel. In our non-exhaustive investigation, altering the default settings did not improve the model's predictive ability on the test data set. The baseline performance of a support vector machine model on the BIC-1, BIC-2, and BIC-3 datasets is presented in Section 3.1.

2.2.2. Neural networks

Neural networks are a popular method for both classification and regression problems [52]. Neural networks function by connecting an input layer of nodes, one or more hidden layers of nodes, and an output layer of nodes via a collection of weighted activation functions. For the binary classification problem relevant to the BIC datasets, the input layer has 16 nodes, and the output layer has 1 node that indicates if the pattern is either "stable" or "unstable." Training a neural network model involves defining a network architecture and algorithmically selecting the network weights that lead to the best delineation between data classes. Here we implement a standard feedforward neural network, the multi-layer perceptron classifier, with three hidden layers with 200 nodes each. We implement our neural network with the python scikit-learn multi-layer perceptron classifier module with a 'lbfgs' solver, a L2 penalty of $\alpha = 10^{-5}$, and otherwise default settings, which includes rectified linear unit activation functions for the hidden layers [51]. In our non-exhaustive investigation, the network architecture and fitting parameters that we selected generalized best to our test data set. The baseline performance of a neural network model on the BIC-1, BIC-2, and BIC-3 datasets is presented in Section 3.1.

2.2.3. Gaussian process classification

Gaussian process classification is the probabilistic classification counterpart to Gaussian process regression [53]. Notably, Gaussian process regression has been a very popular metamodeling strategy in the recent computational mechanics literature [1,45]. Fundamentally, Gaussian process classification is a non-parametric Bayesian method for interpolating data [53]. Training a Gaussian process classification model involves defining a kernel structure and algorithmically optimizing the kernel hyperparameters. We implement our Gaussian process classification model with the python scikit-learn Gaussian process classification module with a radial basis function kernel with an initially set length scale of 10 and otherwise default settings [51]. In our non-exhaustive investigation, our choice of kernel form and fit optimization parameters generalized best to our test data set. The baseline performance of a Gaussian process classification model on the BIC-1, BIC-2, and BIC-3 datasets is presented in Section 3.1. Although we do not explore it directly in this work, Gaussian processes are especially appealing in part because they make probabilistic predictions and thus offer a natural opportunity for uncertainty quantification [54].

2.3. Adversarial attacks

Recently, the concept of adversarial attacks has gained substantial attention in the broader machine learning literature [55,56]. The idea behind an adversarial attack in the context of machine learning is that often machine learning based models that are highly successful at prediction remain "brittle" or vulnerable to the effects of small perturbation on the input parameter space [57]. For example, it has been demonstrated that by altering a single pixel in an image, it is possible to fool deep neural networks which otherwise achieve high accuracy on the ImageNet test data [10]. Here we define two types of adversarial attack that we can explore with the BIC datasets: using an attack metamodel to inform subtle changes to the physical system (type 1 attack, Section 2.3.1) and using an attack model to select inputs that will trick a pre-trained metamodel (type 2 attack, Section 2.3.2).

2.3.1. Type 1 adversarial attack

The first type of adversarial attack that we introduce in this work (type 1 attack) is motivated by the idea that an adversarial actor may want to make a small change to the manufacturing input file that will cause a stable input pattern to become unstable. There has been recent literature showing that this is a potential vulnerability of parts made with additive manufacturing [58–61]. Essentially, adversarial actors may gain access to design files and make a small change to the file that is undetectable to a human observer once the part is manufactured, but will cause the part to mechanically fail once it bears load [62]. This is particularly

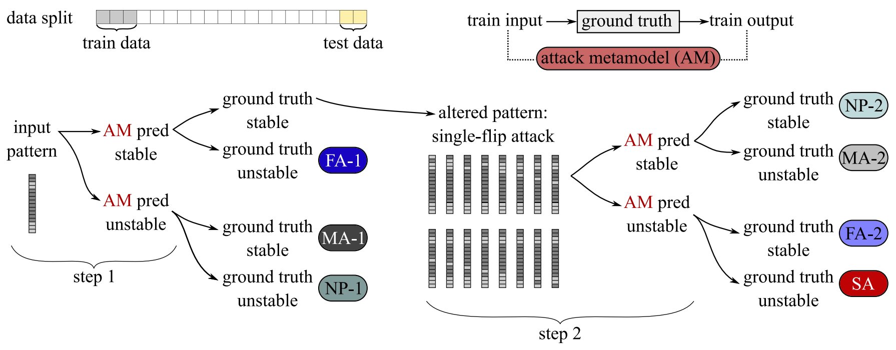

Fig. 2. The flow chart of a type 1 attack is shown here. To execute a type 1 attack, a certain number of model evaluations are first required to build an attack metamodel. In a type 1 attack, the goal is to use the attack metamodel to first identify inputs that are stable (step 1), then identify subtle changes (i.e. single entry changes) to the inputs that will make the true nature of the input design go from stable to unstable (step 2). A successful attack (SA) executes this process correctly with respect to the ground truth. For each candidate input pattern, the outcome can be attack not possible (NP-1, NP-2), a missed attack (MA-1, MA-2), a failed attack (FA-1, FA-2), or a successful attack (SA).

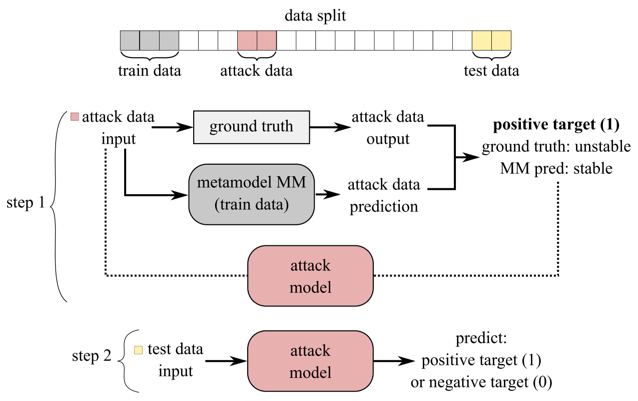

Fig. 3. The procedure for executing a type 2 attack on a trained metamodel is shown here. First, a new subset of data (attack data) is run through both the ground truth model and the trained metamodel. Input patterns that the metamodel classifies as "stable" while the ground truth is "unstable" are noted (false negatives). Then, an attack model is trained to predict these rare cases. The performance of the attack model is evaluated on its ability to predict the false negatives of the original trained metamodel.

relevant for parts where small changes may lead to catastrophic events such as unstable crack propagation or buckling.

To execute a type 1 attack, an adversarial actor will first sample the direct model (ground truth) a fixed number of times. Then, the adversarial actor will create an "attack metamodel" by training a machine learning model (see Section 2.2) on the data samples. Then, the adversarial actor will follow the steps illustrated in Fig. 2. First, the attack metamodel will identify stable designs. Then, for each stable design, it will test 16 altered designs where each altered design is identical to the original design except for one entry. A successful attack (SA) predicts that a sample will go from stable to unstable by changing a single entry and is correct with respect to the ground truth. As illustrated in Fig. 2, we define the other possible attack outcomes given an initial design: attack not possible (NP), missed attack (MA), and failed attack (FA). If the adversarial actor samples the ground truth a higher number of times, the attack metamodel will have a higher performance i.e. the number of successful attacks will increase while the number of missed attacks and failed attacks will decrease. However, the need to gather a high number of ground truth samples is likely undesirable for an adversarial actor. We are interested in understanding how many ground truth samples are required to make successful attack. Baseline attack model performance is demonstrated in Section 3.2.

2.3.2. Type 2 adversarial attack

The second type of adversarial attack that we introduce in this work (type 2 attack) is motivated by the growing popularity of computational models and metamodels in additive manufacturing [63–68]. Once a metamodel is trained, an adversarial actor may want to identify samples that will deliberately fool the metamodel. Specifically, a type 2 attack is an attack where an adversarial actor has identified an input pattern that according to the ground truth will be classified as unstable, but a trained metamodel of interest will incorrectly classify it as stable. In Fig. 3, we show our simple approach for executing a type 2 attack. Given the ground truth and a trained metamodel, we run new ground truth simulations to create a set of classified "attack data." Then, the ground truth class of the attack data is compared to the predicted class of the trained metamodel. Based on this comparison, we identify the input patterns where the ground truth class is unstable but the trained metamodel prediction is stable and mark them as the "positive" class. Then, we train an attack model to predict this positive class. We evaluate the performance of our attack model by measuring how successfully it is able to identify cases that fool the trained metamodel in a test dataset. Baseline attack model performance is demonstrated in Section 3.3.

2.3.3. Note on why the BIC-1 dataset is convenient for assessing adversarial attacks

In Sections 3.2 and 3.3, we show the baseline performance of both type 1 and type 2 attacks on the BIC-1 dataset. The BIC-1 dataset is a convenient dataset for benchmarking attack performance because of the constraints on the input domain. Due to these constraints, there are only 65,536 possible designs, and every possible input has been simulated. This means that an algorithm to select which patterns to test can be implementing without running new simulations. To test the efficacy of adversarial attacks on the BIC-2 and BIC-3 datasets, either the input domain will have to be constrained to already simulated designs, or new simulations will be required. Details for accessing the code required to run additional simulations are given in Section 5. Notably, our algorithms for implementing both the type 1 and type 2 attacks are not particularly sophisticated. It is our hope that by releasing the BIC datasets as open source, other researchers will be able to build on this initial work and exceed our baseline performance for both developing metamodels that are resistant to adversarial attacks, and developing attack metamodel and models.

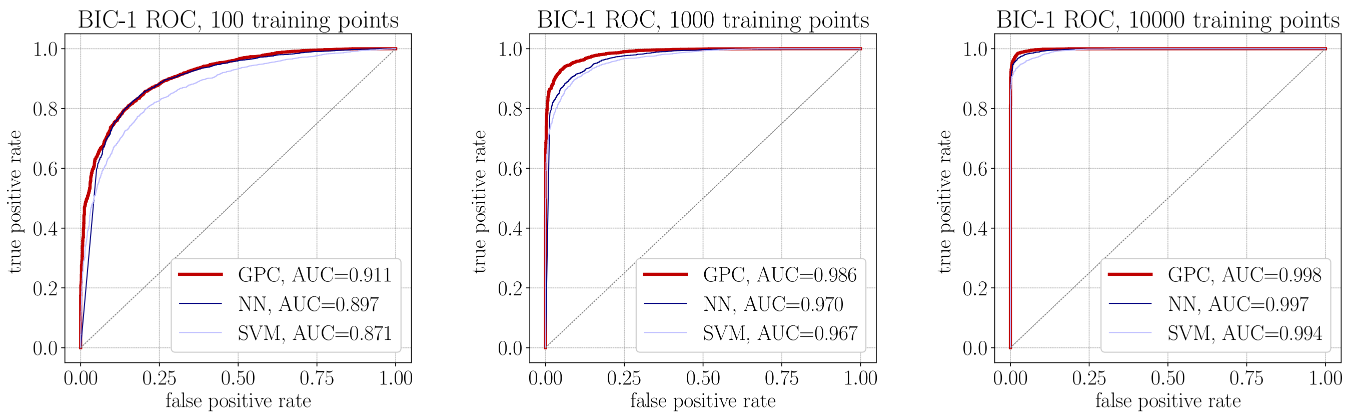

Fig. 4. ROC curves for metamodel performance on test data from the BIC-1 dataset. The left plot shows metamodels trained with 100 examples, the center plot shows metamodels trained with 1000 examples, and the right plot shows metamodels trained with 10,000 examples. For each set of training data, the performance of a Support Vector Machines (SVM), a Neural Network (NN), and a Gaussian Process Classifier (GPC) is shown. Consistently, the GPC metamodel performs best.

3. Results and discussion

In Section 3.1, we show the performance of a support vector machine metamodel, a neural network metamodel, and a Gaussian process classification metamodel on the BIC-1, BIC-2, and BIC-3 datasets. Then, in Sections 3.2 and 3.3, we show the baseline performance of type 1 and type 2 adversarial attacks on the BIC-1 dataset.

3.1. Baseline metamodel performance

In Section 2.2, we introduced three different types of metamodels that are suitable for making predictions with the BIC datasets. Here, we show the results of these metamodels for predicting the test portion of each BIC dataset. To demonstrate metamodel performance, we show the receiver operating characteristic (ROC) curves and state the area under the curve (AUC) for each model. This is a standard way of reporting the performance of classification algorithms [69]. Essentially, the output of a classification algorithm is a continuous variable. Thus, denoting between positive and negative criterion requires selecting a threshold criterion. In an ROC curve, true positive rate is plotted with respect to false positive rate at different decision thresholds. For the unfamiliar reader, we provide a brief introduction to ROC curves and AUC in Appendix C. Most critically, $AUC = 0.5$ represents a classification algorithm that performs no better than a random guess while $AUC = 1.0$ represents a classification algorithm that performs perfectly.

In Fig. 4, we show baseline metamodel performance on the BIC-1 dataset. We show performance of all three metamodel types with different amounts of training data: 100 training points, 1000 training points, and 10,000 training points. For all three metamodels, performance (as measured by AUC) improves as the amount of training data increases. At all three levels of training data, the Gaussian process classification algorithm performs best with $AUC = 0.911$ at 100 training points, $AUC = 0.986$ at 1000 training points, and $AUC = 0.998$ at 10,000 training points. In Fig. 5, we show the baseline metamodel performance on the BIC-2 dataset and in Fig. 6 we show the baseline metamodel performance on the BIC-3 dataset. Qualitatively, metamodel performance across all three datasets is fairly consistent. For BIC-2, the Gaussian process classification algorithm performs best with $AUC = 0.854$ at 100 training points, $AUC = 0.945$ at 1000 training points, and $AUC = 0.987$ at 10,000 training points. For BIC-3, the Gaussian process classification also performs best with $AUC = 0.870$ at 100 training points, $AUC = 0.956$ at 1000 training points, and $AUC = 0.988$ at 10,000 training points. These results suggest that in comparison to support vector machines and neural networks, Gaussian process classification may be the best option for mechanical stability classification problems. Though there is no direct evidence that this will be true beyond the scope of this dataset, these results indicate that Gaussian process classification is a method worthy of investigation for mechanics related classification problems moving forward. We anticipate that for many classification problems with more than 16 relevant input parameters, more training data will be required to make predictions with the same level of accuracy as what is shown here. We note that global sensitivity analysis enabled by metamodels may also help identify important model input parameters [70].

Overall, the relatively high performance of Gaussian process classification for 100s–1000s of training points is not surprising [53]. The efficacy of Gaussian process classification for physically-based problems is consistent with results reported elsewhere in the literature [71–73]. However, one notable limitation of Gaussian processes is that, unless specialized methods are implemented, complexity scales cubically with the number of training points [74]. This, combined with the fact that neural networks often perform substantially better with a higher number of training points, means that if more training points are used there will potentially be a crossover point where neural networks become the preferred method for constructing metamodels on the BIC dataset.

3.2. Type 1 attack example

In Section 2.3.1, we introduced the concept of a type 1 attack where an adversarial actor will use the information that they know about the ground truth model (training data) to construct an adversarial metamodel that will be used to (1) identify stable designs and then (2) identify single entry changes to the input parameter vector that will cause the stable designs to become unstable. Of the 6553 designs in the BIC-1 test data set, 1838 are stable. Therefore, according to the definition illustrated in Fig. 2, there are 4715 examples of attack not possible 1 (NP-1) in the test data ground truth. For design that are initially stable, we consider each single entry flip. If the single entry flip also leads to a stable design, as is the case for 21,091 examples in the test data ground truth, it is an example of attack not possible 2 (NP-2). If the single entry flip leads to an unstable design, as is the case for 8317 examples in the test data ground truth, it is an example of a successful attack (SA). In Fig. 7, we show that as the number of training samples is increased, the attack metamodel is better able to capture the ground truth. In Fig. 7 we also show

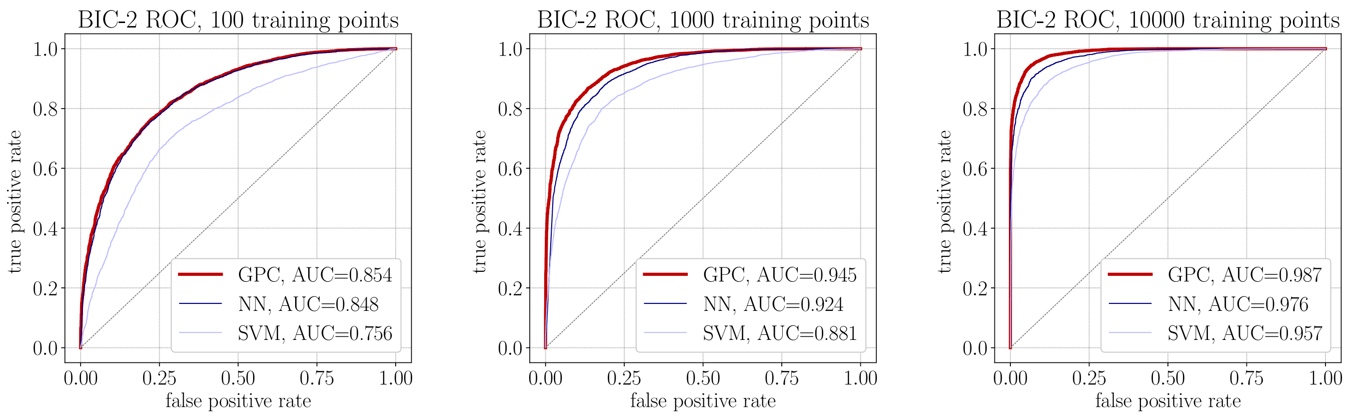

Fig. 5. ROC curves for metamodel performance on test data from the BIC-2 dataset. The left plot shows metamodels trained with 100 examples, the center plot shows metamodels trained with 1000 examples, and the right plot shows metamodels trained with 10,000 examples. For each set of training data, the performance of a Support Vector Machines (SVM), a Neural Network (NN), and a Gaussian Process Classifier (GPC) is shown. Consistently, the GPC metamodel performs best.

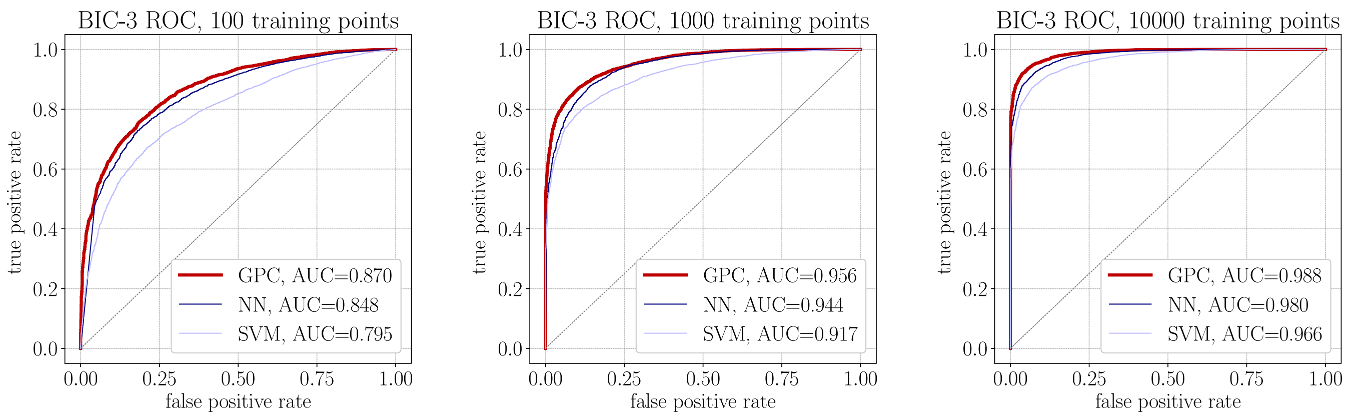

Fig. 6. ROC curves for metamodel performance on test data from the BIC-3 dataset. The left plot shows metamodels trained with 100 examples, the center plot shows metamodels trained with 1000 examples, and the right plot shows metamodels trained with 10,000 examples. For each set of training data, the performance of a Support Vector Machines (SVM), a Neural Network (NN), and a Gaussian Process Classifier (GPC) is shown. Consistently, the GPC metamodel performs best.

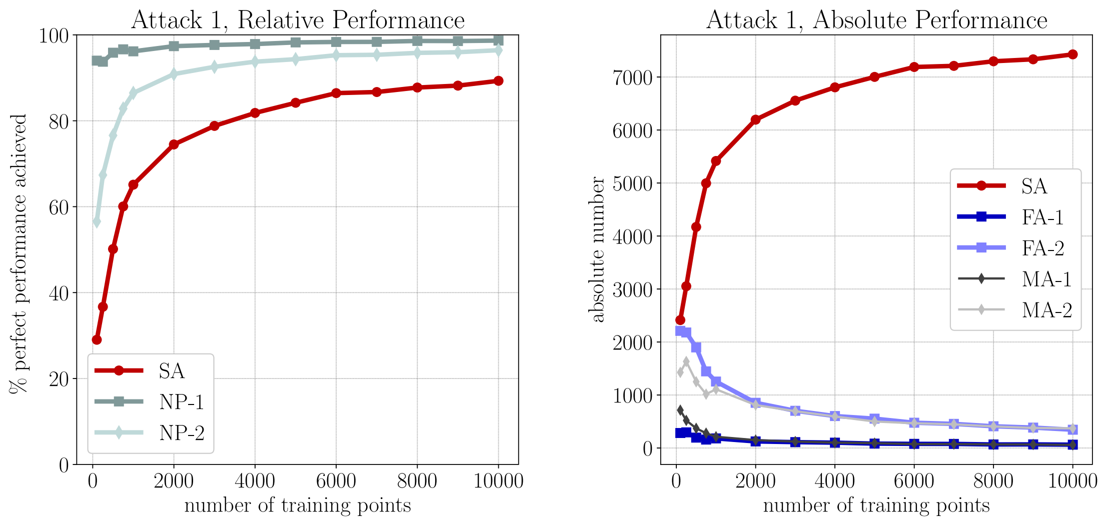

Fig. 7. These plots show the performance of attack metamodels executing type 1 attacks on test data from the BIC-1 dataset. The left plot shows the attack performance compared to the ground truth with respect to the number of training examples used to train the metamodel. The right plot shows the absolute attack performance with respect to the number of training examples used to train the metamodel. The results shown here are for a GPC metamodel. For the BIC-1 test data, there are 8317 possible successful attacks (SA). Each potential outcome (SA, NP-1, NP-2, MA-1, MA-2, FA-1, FA-2) is illustrated in Fig. 2 flowchart.

that as the number of training samples is increased the number of failed attacks and missed attacks decreases. Notably, there is a sharp increase in the number of successful attacks between 100 and 2000 training points. Based on the performance of the different metamodeling techniques shown in Section 3.1, we chose Gaussian process classification for our attack metamodel.

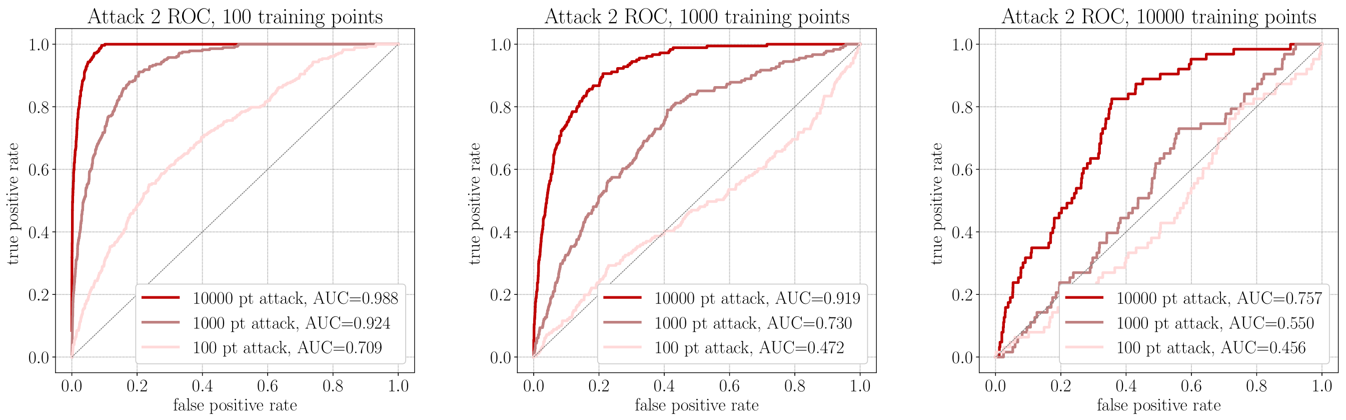

Fig. 8. ROC curves that show the performance of a type 2 attack model on test data from the BIC-1 dataset where a GPC metamodel is attacked. In the left plot, the GPC metamodel is trained with 100 examples, in the center plot the GPC metamodel is trained with 1000 examples, and in the right plot the GPC metamodel is trained with 10,000 examples. In each plot, the performance of a Neural Network attack model trained on 100, 1000, and 10,000 examples is shown.

Given the opportunity afforded by the open source BIC datasets, it would be interesting to see if a different metamodel selection identified by other researchers could outperform the baseline results reported here by achieving higher performance with fewer training points. Furthermore, here we chose the training data points at random. It would be interesting to see if an informed process for selecting training points could lead to a better attack metamodel performance.

3.3. Type 2 attack example

In Section 2.3.2, we introduced the concept of a type 2 attack where an adversarial actor will construct an attack model to intentionally fool a trained metamodel into predicting that an unstable design is actually stable. Our straightforward procedure for constructing an attack model is illustrated in Fig. 3. Here we show the performance of an attack model at executing a type 2 attack on our trained metamodel. Based on the results shown in Section 3.1, we choose Gaussian process classification as our trained metamodel. When the trained metamodel is trained with 100 data points, there are 283 possible type 2 attacks in the 6553 point test dataset. With 1000 training points there are 181 possible attacks, and with 10,000 training points there are 63 possible attacks. For each trained metamodel (100 training points, 1000 training points, 10,000 training points) we train three attack models. As illustrated in Fig. 3, the attack model requires new data for training. Our three attack models are trained with 100, 1000, and 10,000 new data points respectively.

For our attack model, we use a neural network with the structure and parameters described in Section 2.2.2. Based on our non-exhaustive investigation, a neural network with this architecture outperformed other neural networks, support vector machines, and Gaussian process classifiers. Similar to Section 3.1, we report the performance of our attack model by plotting ROC curves. ROC curves work well here in particular because the "positive attack" class is a rare event. The performance of our attack model is shown in Fig. 8. For the metamodel trained with 100 points, all three attack models are able to make predictions that are better than random. For the metamodel trained with 1000 points, the attack models trained with 1000 and 10,000 points were able to make predictions that are better than random. For the metamodel trained with 10,000 points, only the attack model trained with 10,000 points was able to make predictions that are better than random. In all cases, an attack model trained with sufficient points is able to correctly identify new points where a trained metamodel will incorrectly predict that an unstable design is stable. These results show that even nearly perfect classifiers (see Fig. 4) are vulnerable to type 2 attacks.

4. Conclusion

In this paper, we introduce three datasets for mechanical stability classification: BIC-1, BIC-2, and BIC-3. These datasets map the heterogeneous material properties of a hyperelastic column to a class of either "stable" or "unstable" at a defined level of applied displacement. Critically, these datasets and the code to run the underlying simulations are available open source, with access details provided in Section 5. We then show baseline metamodel performance on all three datasets in Section 3.1. Notably, in this initial work, Gaussian process classification performed best. Then, in Section 3.2, we showed baseline examples of a type 1 attack, where an attack metamodel is used to selectively alter candidate designs to change their stability. In Section 3.3, we showed baseline examples of a type 2 attack, where an attack model is used to fool a trained metamodel. The efficacy of the type 2 attacks in particular demonstrates that even nearly perfect classification algorithms are vulnerable to targeted attacks.

Looking forward, we hope that the open source BIC datasets will enable further investigation into machine learning classification algorithms specific to mechanical data. Beyond what we have covered in this paper, we anticipate that future opportunities in improved metamodeling will arise from mixing the BIC datasets together, defining algorithms that inherently handle symmetry, defining algorithms informed by the mechanics of buckling problems, and intelligently selecting samples for the training data. For the BIC-1 dataset, the full design space is provided, therefore no additional model runs are required to implement a candidate sampling algorithm. For the BIC-2 and BIC-3 datasets, the code to generate new data is provided with access details in Section 5. We also anticipate future opportunities in creating improved attack models, and in creating metamodels that are more resistant to type 2 attacks. Because we take a "brute force" method for implementing both our type 1 and type 2 attacks, there is substantial room for improvement in this area. Beyond the scope of this paper, it would also be interesting to see if investigating stability as a regression problem rather than a classification problem could lead to more robust models. From an application perspective, we anticipate that the methods presented here will be useful for advancing the state of the art in additive manufacturing ranging from simulating slender support structures [75] to novel simulation techniques [76]. It will also be interesting to see if the methods that perform best on the BIC datasets also perform best on classifying other types of structural failure, and if models trained on the BIC datasets are transferable to other types of mechanically-based classification problems [77]. We hope that more open source benchmark datasets specific to mechanical design data will arise to meet the growing demand for machine learning based approaches to mechanical predictions.

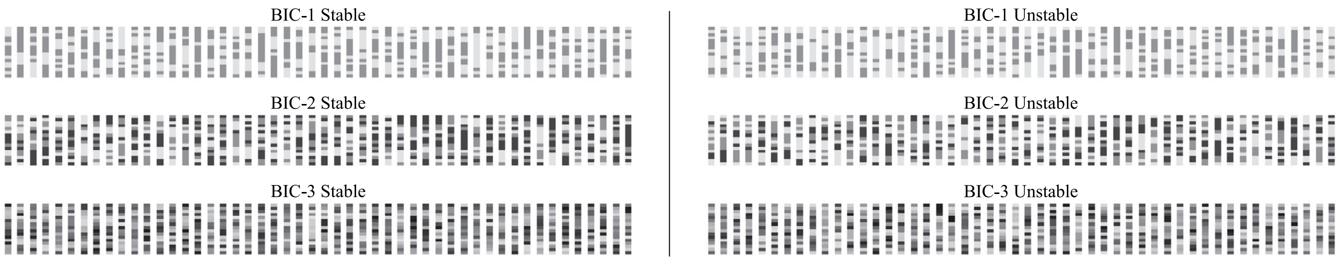

Fig. A.9. Visualization of the BIC datasets input vectors. For each dataset, 50 randomly chosen "stable" and "unstable" designs are shown. Darker colors correspond to segments with a higher modulus (range $E = 1 - 8$).

Table B.1
Summary of machine learning model parameters. All models investigated in this paper are created with scikit-learn [51], see Section 5 for access to the code.

Support vector machine (`svm.SVC`)	
Regularization parameter (C), $\mathcal{L}_2^2$ penalty	1.0
Kernel	Radial basis function
Kernel coefficient gamma	$1.0/(n_{features} \times X.var())$
Shrinking	True
Tolerance	10^{-3}
Class weight	Proportional to balance in training set
Maximum number of iteration	No limit
Neural network (`MLPClassifier`)	
Input layer	16 nodes (size of BIC input)
Hidden layers	3 layers, 200 nodes each
Output layer	1 node
Activation function	Rectified linear unit function ('relu')
Solver	'lbfgs' optimizer (quasi-Newton method)
L2 penalty regularization α	10^{-5}
Tolerance	10^{-4}
Maximum number of loss function calls	15 000
Gaussian process classification (`GaussianProcessClassifier`)	
Kernel type	Radial basis function
Initialized kernel length scale (optimized during fitting)	10.0
Kernel length scale bounds	$10^{-5} - 10^{5}$
Optimizer	'fmin_l_bfgs_b', `scipy.optimize.minimize`
Maximum number of iterations for approximating the posterior during predict	100

5. Supplementary materials

The BIC datasets are freely available under an open source license in the OpenBU digital repository: https://open.bu.edu/handle/2144/40085 [31]. All of the code to generate the BIC datasets and the results reported in this manuscript is available on Github: https://github.com/elejeune11/BIC.

Declaration of competing interest

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

Acknowledgments

We would like to thank the staff of the Boston University Research Computing Services and the OpenBU Institutional Repository (in particular Eleni Castro) for their invaluable assistance with generating and disseminating the BIC datasets. This work was made possible though start up funds from the Boston University Department of Mechanical Engineering, United States of America.

Appendix A. BIC example data

In Fig. A.9, we show 50 randomly selected examples of both "stable" and "unstable" input designs from each of the BIC datasets. For each group, there is no single input parameter that entirely dominates the stability behavior. However, for a given "stable" or "unstable" design, it is possible in some cases to intuit the first buckling mode based on the distribution of material properties and anticipate that it would lead to a "stable" or "unstable" column at the given level of applied displacement. For example, in some cases, distinctly stiffer and softer areas may lead to a first buckling mode that corresponds to bending concentrated in the softer region. We note that for the chosen level of applied displacement, the homogeneous column would be classified as "unstable" (see Section 2.1.2).

Appendix B. Metamodel details

Here, we provide additional detail on metamodel implementation beyond what is given in Section 2.2. All machine learning models described in this paper are created with scikit-learn [51]. Access details for the relevant code are provided in Section 5. A summary of model parameters is provided in Table B.1. For model selection and model parameter selection, our objective was to identify the model that would lead to the lowest test error. Broadly speaking, this means selecting the model that best balances the bias vs. variance tradeoff [11]. Of the models covered in this paper (support vector machines, neural networks, and Gaussian process classification), neural networks are typically most at risk for overfitting the training data [78]. When a model is overfitting, error on the training data will typically be lower than the error on the test data. In the context of this paper, where we were primarily interested in making predictions based on 100 s–1000 s of training datapoints, we found that neural networks performance degraded due to overfitting when we tested both deeper (more layers) and wider (more nodes per layer) networks. We welcome additional exploration with alternate neural network architectures, in particular neural network architectures that require more control than what is presently possible in the scikit-learn framework.

Appendix C. Background on receiver operating characteristic (ROC) curves and area under the curve (AUC)

In Section 3, we report both metamodel (Figs. 4, 5, 6) and attack model (Fig. 8) performance via receiver operating characteristic (ROC) curves. This is standard in the machine learning literature [69]. Here we briefly describe ROC curves in more detail specific to the context of the BIC datasets.

In this publication, all models are binary classification models. In the output files of the BIC datasets, the “unstable” case is denoted with “1” and is considered positive. The “stable” case is denoted with “0” and is considered negative. Therefore, for our metamodels, running our classification algorithm can lead to four possible outcomes: TP true positive (ground truth is unstable, algorithm predicts unstable), FP false positive (ground truth is stable, algorithm predicts unstable), FN false negative (ground truth is unstable, algorithm predicts stable), and TN (ground truth is stable, algorithm predicts stable). Given these possible outcomes, we define true positive rate (TPR) as

$$TPR = \frac{TP}{TP + FN} \tag{C.1}$$

and false positive rate (FPR) as

$$FPR = \frac{FP}{FP + TN}\,. \tag{C.2}$$

For our attack model (Section 2.3.2, Figs. 3, 8), we define positive as an “unstable” ground truth combined with a “stable” trained metamodel prediction and negative as all other cases.

The output of each classification algorithm is a continuous variable. Thus, denoting between a positive and a negative predicted outcome requires selecting a threshold criterion. The choice of threshold criterion effects the TPR and the FPR. For a perfect classification algorithm, it will be possible to select a threshold such that $TPR = 1$ and $FPR = 0$. For a classifier with random performance (i.e. a random guess) we will observe $TPR \approx FPR$ at any threshold criterion. A ROC curve is a plot of TPR (y axis) with respect to FPR (x axis). For an ideal random classifier, the ROC curve will be a straight line with formula $TPR = FPR$. This line is illustrated in all ROC curves in this manuscript. The choice of threshold will depend on user tolerance for both false positives and false negatives. The threshold that corresponds to the point on the curve closest to point (0, 1.0) will have the lowest error. Reporting the area under the curve (AUC), defined as the integral of TPR with respect to FPR, is a simple way to summarize model performance. For a perfect classifier, $AUC = 1.0$ and for a random classifier $AUC = 0.5$. The AUC also reflects the probability that the classification algorithm will score a randomly chosen positive example more highly than a randomly chosen negative example [69]. In this work, we only plot the ROC curves for the performance on test (unseen) data.

This method for evaluating metamodel (or attack model) performance is especially useful for models that attempt to predict rare events, where predictive algorithms can erroneously appear highly successful by never predicting the rare event. In this work, the attack model for a type 2 attack attempts to predict rare events. And, for all BIC datasets, there is an uneven split between the number of positive and negative examples. Further information on ROC curves and AUC for model evaluation is available in introductory machine learning literature [11].

References

[1] Costabal FS, Perdikaris P, Kuhl E, Hurtado DE. Multi-fidelity classification using gaussian processes: accelerating the prediction of large-scale computational models. Comput Methods Appl Mech Engrg 2019;357:112602.
[2] Peirlinck M, Costabal FS, Sack K, Choy J, Kassab G, Guccione J, et al. Using machine learning to characterize heart failure across the scales. Biomech Model Mechanobiol 2019;18(6):1987–2001.
[3] Krizhevsky A, Sutskever I, Hinton GE. Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. 2012, p. 1097–105.
[4] Sharma AK, Sahni S. A comparative study of classification algorithms for spam email data analysis. Int J Comput Sci Eng 2011;3(5):1890–5.
[5] LeCun Y, Bottou L, Bengio Y, Haffner P, et al. Gradient-based learning applied to document recognition. Proc IEEE 1998;86(11):2278–324.
[6] He K, Zhang X, Ren S, Sun J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. 2015, p. 1026–34.
[7] Cheng H-D, Shan J, Ju W, Guo Y, Zhang L. Automated breast cancer detection and classification using ultrasound images: A survey. Pattern Recognit 2010;43(1):299–317.
[8] Deng J, Dong W, Socher R, Li L-J, Li K, Fei-Fei L. Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. Ieee; 2009, p. 248–55.
[9] Sabour S, Frosst N, Hinton GE. Dynamic routing between capsules. In: Advances in neural information processing systems. 2017, p. 3856–66.
[10] Su J, Vargas DV, Sakurai K. One pixel attack for fooling deep neural networks. IEEE Trans Evol Comput 2019;23(5):828–41.
[11] James G, Witten D, Hastie T, Tibshirani R. An introduction to statistical learning, Vol. 112. Springer; 2013.
[12] Hendrycks D, Dietterich T. Benchmarking neural network robustness to common corruptions and perturbations. 2019, arXiv preprint arXiv:1903.12261.
[13] Bessa MA, Glowacki P, Houlder M. Bayesian machine learning in metamaterial design: Fragile becomes supercompressible. Adv Mater 2019;31(48):1904845.
[14] Lejeune E. Mechanical MNIST: A benchmark dataset for mechanical metamodels. Extreme Mech Lett 2020;100659.
[15] Zohdi T. A machine-learning framework for rapid adaptive digital-twin based fire-propagation simulation in complex environments. Comput Methods Appl Mech Engrg 2020;363:112907.
[16] Wu J, Qian X, Wang MY. Advances in generative design. Comput Aided Des 2019;111.
[17] Bessa M, Bostanabad R, Liu Z, Hu A, Apley DW, Brinson C, et al. A framework for data-driven analysis of materials under uncertainty: countering the curse of dimensionality. Comput Methods Appl Mech Engrg 2017;320:633–67.
[18] Cang R, Yao H, Ren Y. One-shot generation of near-optimal topology through theory-driven machine learning. Comput Aided Des 2019;109:12–21.
[19] Guilleminot J, Dolbow JE. Data-driven enhancement of fracture paths in random composites. Mech Res Commun 2020;103:103443.
[20] Gunpinar E, Coskun UC, Ozsipahi M, Gunpinar S. A generative design and drag coefficient prediction system for sedan car side silhouettes based on computational fluid dynamics. Comput Aided Des 2019;111:65–79.
[21] Teichert G, Natarajan A, Van der Ven A, Garikipati K. Machine learning materials physics: Integrable deep neural networks enable scale bridging by learning free energy functions. Comput Methods Appl Mech Engrg 2019;353:201–16.
[22] Teichert GH, Garikipati K. Machine learning materials physics: Surrogate optimization and multi-fidelity algorithms predict precipitate morphology in an alternative to phase field dynamics. Comput Methods Appl Mech Engrg 2019;344:666–93.
[23] Zhu Y, Zabaras N, Koutsourelakis P-S, Perdikaris P. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. J Comput Phys 2019;394:56–81.
[24] Yang Y, Perdikaris P. Conditional deep surrogate models for stochastic, high-dimensional, and multi-fidelity systems. Comput Mech 2019;1–18.
[25] Forrester AI, Keane AJ. Recent advances in surrogate-based optimization. Prog Aerosp Sci 2009;45(1–3):50–79.

[26] Queipo NV, Haftka RT, Shyy W, Goel T, Vaidyanathan R, Tucker PK. Surrogate-based analysis and optimization. Prog Aerosp Sci 2005;41(1):1–28.
[27] Alber M, Tepole AB, Cannon WR, De S, Dura-Bernal S, Garikipati K, et al. Integrating machine learning and multiscale modeling—perspectives, challenges, and opportunities in the biological, biomedical, and behavioral sciences. npj Digit Med 2019;2(1):1–11.
[28] Li A, Chen R, Farimani AB, Zhang YJ. Reaction diffusion system prediction based on convolutional neural network. Sci Rep 2020;10(1):1–9.
[29] Wang K, Sun W. A multiscale multi-permeability poroplasticity model linked by recursive homogenizations and deep learning. Comput Methods Appl Mech Engrg 2018;334:337–80.
[30] Lejeune E, Linder C. Interpreting stochastic agent-based models of cell death. Comput Methods Appl Mech Engrg 2019;112700.
[31] Lejeune E. Buckling instability classification (BIC). 2020, URL: https://open.bu.edu/handle/2144/40085.
[32] Timoshenko SP, Gere JM. Theory of elastic stability. Courier Corporation; 2009.
[33] Lejeune E, Javili A, Linder C. An algorithmic approach to multi-layer wrinkling. Extreme Mech Lett 2016;7:10–7.
[34] Lejeune E, Javili A, Linder C. Understanding geometric instabilities in thin films via a multi-layer model. Soft Matter 2016;12(3):806–16.
[35] Javili A, Dortdivanlioglu B, Kuhl E, Linder C. Computational aspects of growth-induced instabilities through eigenvalue analysis. Comput Mech 2015;56(3):405–20.
[36] Alnæs M, Blechta J, Hake J, Johansson A, Kehlet B, Logg A, et al. The fenics project version 1.5. Arch Numer Softw 2015;3(100).
[37] Logg A, Mardal K-A, Wells G. Automated solution of differential equations by the finite element method: The FEniCS book, Vol. 84. Springer Science & Business Media; 2012.
[38] Balay S, Abhyankar S, Adams M, Brown J, Brune P, Buschelman K, et al. PETSc users manual. Argonne National Laboratory; 2019.
[39] Balay S, Abhyankar S, Adams MF, Brown J, Brune P, Buschelman K, et al. PETSc web page. 2019, https://www.mcs.anl.gov/petsc, URL: https://www.mcs.anl.gov/petsc.
[40] Hernandez V, Roman JE, Vidal V. SLEPc: A scalable and flexible toolkit for the solution of eigenvalue problems. ACM Trans Math Softw 2005;31(3):351–62.
[41] Roman JE, Campos C, Romero E, Tomas A. SLEPc users manual. Tech. Rep. DSIC-II/24/02 - Revision 3.13, D. Sistemes Informàtics i Computació, Universitat Politècnica de València; 2020.
[42] McKay MD, Beckman RJ, Conover WJ. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 2000;42(1):55–61.
[43] Olsson A, Sandberg G, Dahlblom O. On latin hypercube sampling for structural reliability analysis. Struct Saf 2003;25(1):47–68.
[44] Sobol' IM. On the distribution of points in a cube and the approximate evaluation of integrals. Zh Vychisl Mat Mat Fiz 1967;7(4):784–802.
[45] Raissi M, Perdikaris P, Karniadakis GE. Machine learning of linear differential equations using gaussian processes. J Comput Phys 2017;348:683–93.
[46] Wang K, Sun W. Meta-modeling game for deriving theory-consistent, microstructure-based traction–separation laws via deep reinforcement learning. Comput Methods Appl Mech Engrg 2019;346:216–41.
[47] Wang K, Sun W, Du Q. A cooperative game for automated learning of elasto-plasticity knowledge graphs and models with ai-guided experimentation. Comput Mech 2019;1–33.
[48] Friedman J, Hastie T, Tibshirani R. The elements of statistical learning, Vol. 1. Springer series in statistics New York; 2001.
[49] Joachims T. Text categorization with support vector machines: Learning with many relevant features. In: European conference on machine learning. Springer; 1998, p. 137–42.
[50] Hsu C-W, Chang C-C, Lin C-J, et al. A practical guide to support vector classification. Taipei; 2003.
[51] Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, et al. Scikit-learn: Machine learning in python. J Mach Learn Res 2011;12:2825–30.
[52] Gurney K. An introduction to neural networks. CRC press; 2014.
[53] Rasmussen CE. Gaussian processes in machine learning. In: Summer school on machine learning. Springer; 2003, p. 63–71.
[54] Lee T, Turin SY, Gosain AK, Bilionis I, Tepole AB. Propagation of material behavior uncertainty in a nonlinear finite element model of reconstructive surgery. Biomech Model Mechanobiol 2018;17(6):1857–73.
[55] Papernot N, McDaniel P, Goodfellow I, Jha S, Celik ZB, Swami A. Practical black-box attacks against machine learning. In: Proceedings of the 2017 ACM on Asia conference on computer and communications security. 2017, p. 506–19.
[56] Tramèr F, Kurakin A, Papernot N, Goodfellow I, Boneh D, McDaniel P. Ensemble adversarial training: Attacks and defenses. 2017, arXiv preprint arXiv:1705.07204.
[57] Balda E, Behboodi A, Mathar R. Perturbation analysis of learning algorithms: generation of adversarial examples from classification to regression. IEEE Trans Signal Process 2019.
[58] Straub J. 3d printing cybersecurity: detecting and preventing attacks that seek to weaken a printed object by changing fill level. In: Dimensional optical metrology and inspection for practical applications VI, Vol. 10220. International Society for Optics and Photonics; 2017, p. 102200O.
[59] Yampolskiy M, King W, Pope G, Belikovetsky S, Elovici Y. Evaluation of additive and subtractive manufacturing from the security perspective. In: International conference on critical infrastructure protection. Springer; 2017, p. 23–44.
[60] Yampolskiy M, King WE, Gatlin J, Belikovetsky S, Brown A, Skjellum A, et al. Security of additive manufacturing: Attack taxonomy and survey. Addit Manuf 2018;21:431–57.
[61] Yu S-Y, Malawade AV, Chhetri SR, Al Faruque MA. Sabotage attack detection for additive manufacturing systems. IEEE Access 2020;8:27218–31.
[62] Belikovetsky S, Yampolskiy M, Toh J, Gatlin J, Elovici Y. Dr0wned–cyber-physical attack with additive manufacturing. In: 11th {USENIX} workshop on offensive technologies ({WOOT} 17). 2017.
[63] Chen C-T, Gu GX. Generative deep neural networks for inverse materials design using backpropagation and active learning. Adv Sci 2020;1902607.
[64] Yang C, Kim Y, Ryu S, Gu GX. Prediction of composite microstructure stress-strain curves using convolutional neural networks. Mater Des 2020;189:108509.
[65] Zohdi T. Modeling and simulation of cooling-induced residual stresses in heated particulate mixture depositions in additive manufacturing. Comput Mech 2015;56(4):613–30.
[66] Yu Y, Liu H, Qian K, Yang H, McGehee M, Gu J, et al. Material characterization and precise finite element analysis of fiber reinforced thermoplastic composites for 4d printing. Comput Aided Des 2020;122:102817.
[67] Zohdi T. Dynamic thermomechanical modeling and simulation of the design of rapid free-form 3d printing processes with evolutionary machine learning. Comput Methods Appl Mech Engrg 2018;331:343–62.
[68] Zohdi TI. Additive particle deposition and selective laser processing-a computational manufacturing framework. Comput Mech 2014;54(1):171–91.
[69] Fawcett T. An introduction to ROC analysis. Pattern Recognit Lett 2006;27(8):861–74.
[70] Saltelli A, Ratto M, Andres T, Campolongo F, Cariboni J, Gatelli D, et al. Global sensitivity analysis: the primer. John Wiley & Sons; 2008.
[71] Di Achille P, Harouni A, Khamzin S, Solovyova O, Rice JJ, Gurev V. Gaussian process regressions for inverse problems and parameter searches in models of ventricular mechanics. Front Physiol 2018;9:1002.
[72] Lee T, Bilionis I, Tepole AB. Propagation of uncertainty in the mechanical and biological response of growing tissues using multi-fidelity gaussian process regression. Comput Methods Appl Mech Engrg 2020;359:112724.
[73] Raissi M, Perdikaris P, Karniadakis GE. Machine learning of linear differential equations using gaussian processes. J Comput Phys 2017;348:683–93.
[74] Liu H, Ong Y-S, Shen X, Cai J. When gaussian process meets big data: A review of scalable gps. IEEE Trans Neural Netw Learn Syst 2020.
[75] Vaissier B, Pernot J-P, Chougrani L, Véron P. Genetic-algorithm based framework for lattice support structure optimization in additive manufacturing. Comput Aided Des 2019;110:11–23.
[76] Prabhune BC, Suresh K. A fast matrix-free elasto-plastic solver for predicting residual stresses in additive manufacturing. Comput Aided Des 2020;102829.
[77] Liu Z, Wu C, Koishi M. Transfer learning of deep material network for seamless structure–property predictions. Comput Mech 2019;64(2):451–65.
[78] Tetko IV, Livingstone DJ, Luik AI. Neural network studies. 1. Comparison of overfitting and overtraining. J Chem Inf Comput Sci 1995;35(5):826–33.