A Comparative Evaluation of Supervised Machine Learning Classification Techniques for Engineering Design Applications

Conner Sharpe[1]
Walker Department of Mechanical Engineering,
The University of Texas at Austin,
Austin, TX 78701-2982
e-mail: c_sharpe@utexas.edu

Tyler Wiest[1]
Walker Department of Mechanical Engineering,
The University of Texas at Austin,
Austin, TX 78701-2982
e-mail: tylerwiest@utexas.edu

Pingfeng Wang
Department of Industrial and Enterprise Systems Engineering,
University of Illinois at Urbana-Champaign,
Champaign, IL 61801
e-mail: pingfeng@illinois.edu

Carolyn Conner Seepersad
Walker Department of Mechanical Engineering,
The University of Texas at Austin,
Austin, TX 78701-2982
e-mail: ccseepersad@mail.utexas.edu

Supervised machine learning techniques have proven to be effective tools for engineering design exploration and optimization applications, in which they are especially useful for mapping promising or feasible regions of the design space. The design space mappings can be used to inform early-stage design exploration, provide reliability assessments, and aid convergence in multiobjective or multilevel problems that require collaborative design teams. However, the accuracy of the mappings can vary based on problem factors such as the number of design variables, presence of discrete variables, multimodality of the underlying response function, and amount of training data available. Additionally, there are several useful machine learning algorithms available, and each has its own set of algorithmic hyperparameters that significantly affect accuracy and computational expense. This work elucidates the use of machine learning for engineering design exploration and optimization problems by investigating the performance of popular classification algorithms on a variety of example engineering optimization problems. The results are synthesized into a set of observations to provide engineers with intuition for applying these techniques to their own problems in the future, as well as recommendations based on problem type to aid engineers in algorithm selection and utilization.
[DOI: 10.1115/1.4044524]

Keywords: design automation, simulation-based design, classifiers, machine learning, design exploration

1 Introduction

Design engineers are often required to synthesize information, solve intertwining problems, and elegantly present the best designs. The nature of this design process has changed dramatically over the past several decades, driven largely by growth in computational capabilities. By harnessing computational models, an engineer can solve increasingly difficult problems and gather a wealth of information about them, but designing in a computational landscape requires an expanded suite of tools for generating information and learning from it to make better design decisions. In light of this challenge, machine learning algorithms, and specifically classification techniques, have emerged as a particularly useful class of tools for engineering design exploration and optimization. This paper is intended to help design engineers develop a basic understanding of classification and build intuition for the suitability of common classification algorithms for engineering design exploration and optimization applications.

In engineering design exploration and optimization, classification algorithms or classifiers can be used for identifying, bounding, or mapping the feasible design space. In a typical engineering design optimization problem, the task is to identify values of design variables, $\boldsymbol{x}$, that satisfy a set of constraints, $\boldsymbol{g}(\boldsymbol{x})$, and maximize or minimize a set of potentially conflicting objectives, $\boldsymbol{f}(\boldsymbol{x})$. The constraint and objective functions could be based on simulation-based predictions, experimental or historical data, or human assessment. Classifiers that map the feasible design space are focused on identifying feasible and/or preferable values of the design variables that satisfy the constraints and/or meet preferred thresholds for the objective functions, respectively. This task is an example of *inverse* mapping, which is very different from the *forward* mapping strategies that are used pervasively in engineering design. An inverse mapping of the design space seeks to identify the comprehensive set of design variable values that meet a set of performance requirements or thresholds. A forward mapping, in contrast, accepts unique design variable values as input and predicts their performance. Surrogate or meta models (e.g., kriging, regression) are often used to establish computationally efficient forward mappings, but they have limited applicability to inverse mappings that are often nonunique (i.e., a specific level of performance is associated with more than one candidate design) and discontinuous (i.e., disjoint regions of the design space may provide similar levels of performance). Classification methods are well suited to this task because they are designed to predict whether candidate designs are members of a specific class, which could include the set of feasible designs or the set of feasible designs that satisfy preferred performance thresholds [1].

A design engineer may seek to map the design space directly for several reasons. One example is the set-based design, which focuses on solving distributed design problems by delaying commitment to a single point solution and preserving a diversity of options for identifying mutually satisfactory cross-disciplinary solutions [2]. In this context, classifiers have been used to solve distributed, multidisciplinary design problems, which are decomposed into interdependent subproblems that share coupled design variables [3]. A similar set-based approach may be applied to a multiscale or multilevel design problem in which it is important to map the input design space for an upper-level subproblem because it may define the performance constraints of a lower-level subproblem. This approach has been utilized for both materials design [4–6] and additive manufacturing [7] applications.

Classifiers can also be used as a means of guiding and potentially improving the efficiency of design exploration. Classification can

[1]Conner Sharpe and Tyler Wiest share first authorship.

Contributed by the Design Automation Committee of ASME for publication in the Journal of Mechanical Design. Manuscript received March 6, 2019; final manuscript received August 5, 2019; published online September 4, 2019. Assoc. Editor: Mark Fuge.

be combined with active learning and other adaptive sampling strategies, for example, to map the boundaries of the feasible design space accurately and efficiently [6,8–10]. Mapping the boundaries of the feasible design space accurately can be especially important for improving the efficiency of design exploration because it prevents potentially expensive simulation- or experiment-based exploration of infeasible designs [11]. For highly nonlinear, nonconvex design problems, these spaces can be disjoint and assume arbitrary shapes, making it difficult to capture them with simple techniques such as intervals [3,8]. These types of classifiers can also be used as computationally efficient filters for identifying robust designs that meet feasibility constraints or performance thresholds even when the designs themselves are subject to uncontrolled variation, such as manufacturing tolerances [5].

Classification methods are used widely to map feasible design spaces as a means of improving the efficiency of reliability analysis [12–17] and reliability-based [18–20] design. For example, support vector machines (SVMs) can be used as a classification tool to construct mappings of the limit state surfaces that identify safe design regions for reliability assessment [12,13]. Classification with enhanced probabilistic neural networks (NNs) has demonstrated significant improvements in computational efficiency for reliability analysis of structural systems [14]. Similarly, adaptive classification tools and sequential sampling strategies have been used to focus on important design regions and to gradually identify implicit constraint boundaries, thereby improving the efficiency of solving reliability-based design problems [19,21]. To leverage their efficiency advantages, classification methods have also been employed for reliability analysis for high-dimensional problems [15], system reliability analysis considering multiple failure modes [16], reliability-based design considering time-variant probabilistic constraints [18], and reliability-based design of aircraft wings [20] and thermal protections systems in planetary entry vehicle design [22].

Given that classification tools have proven useful in several engineering design exploration and optimization contexts, the aim of this paper is to aid the engineering design community in leveraging them by providing insight into algorithm selection and utilization. Toward this end, we compare the performance of four common classifiers on six example problems with characteristics of broad interest in the engineering design optimization community. Section 3 describes important aspects to consider when implementing a classifier and how the attributes of the design optimization problem can inform algorithm selection and configuration. Section 4 provides a brief technical overview of the popular classifiers considered in this work. Section 5 outlines the various design problems selected to test classifier performances. These problems are simple to implement and contain different combinations of the attributes discussed in Sec. 3, allowing them to serve as benchmarks for future research in the field. Section 6 presents results with varying levels of training data and algorithm adjustment. Finally, Secs. 7 and 8 summarize the results into insights to inform future design activities.

2 General Background on Classification Techniques

As described in Sec. 1, classification can be used in engineering design exploration and optimization problems to map regions of the design space that satisfy requirements or constraints of interest. These requirements can be formulated mathematically as an inequality constraint as follows:

$$\text{Decide} \quad c = c_1 \text{ if } f(x) \leq f_{\text{thresh}}\,; \text{ else decide } c = c_2 \qquad (1)$$

where x is a candidate design, f is the performance function of interest, and c_1 and c_2 represent two classes of interest, for example, feasible and infeasible, respectively. The true classification designation can be known exactly by evaluating a particular design for a deterministic performance function of interest. The performance function is often expensive to evaluate, however, motivating a more efficient alternative for predicting whether a candidate design meets the desired inequality condition. Many classifiers use previously evaluated points to estimate the probability that a new design meets the desired criteria without explicitly evaluating the performance function. A typical probabilistic decision criterion for classification algorithms is shown as follows:

$$\text{Decide } c = c_1 \text{ if } p(c_1|x) > p(c_2|x);\ \text{else decide } c = c_2 \qquad (2)$$

where $p(c|x)$ is the conditional probability of the class given the candidate design, x.

A number of factors should be considered when implementing a classification approach to support the decision criteria defined in Eqs. (1) and (2). The first consideration is the type of classifier to implement. There are two broad categories of classifiers: generative and discriminative [23]. Examples of generative classifiers include naive Bayes (NB) classifiers and Bayesian network classifiers. Support vector machines and neural networks are examples of discriminative classifiers. Discriminative classifiers directly model the conditional probability $p(c|x)$ that a data point, x, is a member of a specific class, c. Generative classifiers instead model the joint probability of data and class $p(c, x)$ which can then be transformed into the posterior probability of a class by using Bayes' rule $p(c|x) = ((p(x|c)p(c))/p(x))$. In many cases, the direct approach of the discriminative classifiers yields better classification accuracy [23], but not in all cases. The generative classifiers can offer different advantages, such as utilizing the joint probability provided by generative classifiers for sampling combinations of class and data, which may be of interest in sequential sampling approaches. In this study, both generative and discriminative classifiers are implemented and compared.

Computational expense is also an important concern. The computational expense for training classifiers grows with the number of training points and the number of design variables, and the scaling of training time with respect to these factors varies for different algorithm types. Table 1 summarizes the worst case theoretical time complexity of the algorithms considered in this work, where n is the number of training points, m is the number of variables, t is the number of trees in a random forest (RF), p is the number of variables randomly sampled at each node of the random forest decision trees, h is the number of neurons (assumed to be constant in each layer for simplicity), k is the number of layers in a neural network, and i is the number of backpropagation iterations [24–26]. The linear time scaling of Gaussian naive Bayes (GNB) makes it the most efficient algorithm to train. Random forests and support vector machines exhibit exponential scaling with the number of training points, causing their computational expense to increase quickly for large datasets. The efficiency of the fully connected neural network depends heavily on the number of layers and neurons present in the network, but many modern networks utilize large numbers of parameters that result in computational expense significantly greater than the other algorithms presented here. It is important to note that the computational complexity in Table 1 refers to the cost of training the classifiers. All of these classifiers require less computational expense for prediction relative to the computational cost of training them.

In addition to training complexity, many algorithms also require significant tuning of hyperparameters to achieve good performance. The tuning process requires searching through various hyperparameter values to determine the values that lead to the best classification performance for a specific problem. This parameter search requires retraining the classifier for each combination of hyperparameters considered, thus making the tuning effort required to reach acceptable performance an important factor in the overall

Table 1 Computational complexity for training classifiers

Algorithm	Computational complexity
Gaussian naive Bayes	$O(nm)$
SVM	$O(n^2m)$ to $O(n^3m)$
Neural network (perceptron)	$O(nmh^ki)$
Random forest	$O(pn^2t\log(n))$

computational expense of implementing an algorithm. Tuning is commonly performed using a cross-validation scheme, in which rotating subsets of the training data are excluded from the training set and used to evaluate the classifier (Chap. 5 of Ref. [1] provides more in-depth background discussion of cross-validation). The performance is then aggregated to find the hyperparameter values that yield the best results throughout the entire space. Cross-validation schemes are used to prevent overfitting data on a single test set. Hyperparameter values can be tuned by simply testing a fixed grid of hyperparameter values or a random set of hyperparameter values, or by implementing sequential techniques such as Bayesian optimization to direct the search toward higher performing hyperparameter values at the expense of longer training time. A simple threefold cross-validation grid search approach is used in this work. Some algorithms are more robust to hyperparameters and offer relatively consistent performance across a range of settings, while others are very sensitive to parameter settings. The importance of tuning will be discussed further in Sec. 6.

Every classification approach expresses a set of underlying assumptions about the data via its mathematical form and the types of hyperparameters that help define its underlying model. Consequently, classifier performance can vary widely depending on the characteristics of the problem, including the number of variables, variable type (continuous and discrete), the strength of interactions between variables, and the modality of the design space. The benchmark design problems in this study are selected as representative engineering design optimization and/or exploration problems that satisfy a few important criteria. Table 2 in Sec. 4 presents the problems and associated characteristics examined in this study. First, they are intuitive engineering problems that require only brief explanation, making prolonged study of the problems unnecessary. Second, each problem embodies one or more features that are often encountered in design optimization problems and are known to affect classifier performance. For thoroughness, there are multiple problems that share many of the important characteristics. In particular, the study investigates the effects of dimensionality, variable types, multimodality in the design space, dependencies between design variables, and the presence of multiple objectives. The presence of multiple objectives does not necessarily affect how the classifier algorithms perform mathematically, but applying different constraints or thresholds for multiple objectives can change the size, shape, and connectivity of the design space to be mapped. Finally, the problems are amenable to binary classification of designs based on a performance threshold. Classifying designs in this way enables the classifiers' predictions to be compared with the true class of each design for the evaluation of the classifiers' performance.

A multitude of scoring metrics have been defined to quantify classifier performance [27], but they all seek to express the degree of similarity between the true and predicted classes of each test sample. The simplest statement of the similarity between the predicted and true classes of a sample set is represented by a confusion matrix [1], as shown in Fig. 1. P and N are the two classes: *positive* and *negative*. The positive class is typically associated with feasible or promising designs in an engineering design context. A correctly identified member of class P is called a true positive and represented by TP. FP is a false positive, which corresponds to a member of class N that is incorrectly identified as a member of class P. It follows that TN and FN are true negative and false negative predictions, respectively.

Most metrics used to score classifiers use some combination of the entries in the confusion matrix. For many classification tasks, accuracy (ACC), as defined in Eq. (3), is a useful starting point for evaluating overall classifier performance, and it is reported for all of the benchmark problems in Sec. 5.

$$\text{ACC} = \frac{\text{TP} + \text{TN}}{(\text{TP} + \text{FP} + \text{FN} + \text{TN})} \tag{3}$$

There are some cases for which other classifier scoring metrics are more appropriate. For example, when there are many more instances of one class compared to the other in a data set, the classification problem is imbalanced. In these cases, an accuracy metric can be misleading, as a classifier that blindly predicts the majority class will show high accuracy but have limited predictive value. Specifically for designers, scoring metrics formulated to prioritize the prediction of feasible or high-performing solutions may be preferred at the expense of overall accuracy. One such metric that provides information on the classification of the positive (P) class is true positive rate (TPR), also commonly referred to as sensitivity or recall. Maximizing TPR increases the likelihood of identifying a true member of the positive class of interest.

$$\text{TPR} = \text{recall} = \frac{\text{TP}}{\text{P}} = \frac{\text{TP}}{(\text{TP} + \text{FN})} \tag{4}$$

Another metric that may be of interest is precision or positive predictive value. The precision metric provides a measure of the prediction accuracy specific to the positive class by reporting the proportion of positive predictions that match with true class membership.

$$\text{precision} = \frac{\text{TP}}{(\text{TP} + \text{FP})} \tag{5}$$

The $F1$ score is a common scoring metric for binary classification that balances precision and recall using a harmonic mean. The $F1$ score is especially popular for assessing classifier performance in cases with a class imbalance or with greater importance placed on classifier performance for the positive class.

$$F1 = 2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}} \tag{6}$$

Finally, false negative rate (FNR) can also be utilized to assess whether the classifier is missing potential members of positive class. Minimizing FNR allows designers to maximize their chances of identifying all potential designs of interest.

$$\text{FNR} = \frac{\text{FN}}{\text{P}} = \frac{\text{FN}}{\text{TP} + \text{FN}} = 1 - \text{TPR} \tag{7}$$

In combination, these metrics capture how well a classifier is identifying regions of the design space containing positive (typically, feasible high performance) designs. Section 5 provides an example of the importance of these metrics, in addition to accuracy, as the design process progresses to later design stages in which tighter design requirements result in greater imbalance between classes.

3 Description of Classification Techniques

In this section, four of the most popular classification techniques for engineering design exploration and optimization problems are briefly reviewed. These techniques represent the pool of algorithms to be used for the comparison study in subsequent sections.

3.1 Support Vector Machine. The SVM is a machine learning technique for classification that separates incoming data by a maximum margin [28]. A two-class case with an optimal separating hyperplane and a maximum margin is shown in Fig. 2. If the two classes are linearly separable, as shown in Fig. 2, the optimal

		Predicted Class	
		P	N
True Class	P	TP	FN
	N	FP	TN

Fig. 1 Confusion matrix for organizing classifier predictive performance

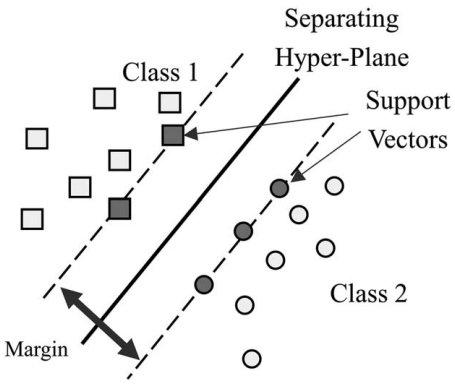

Fig. 2 Concept of a two-class support vector machine (adapted from Ref. [29])

hyperplane separating the data can be expressed as

$$g(x) = w^T * x + b = 0 \tag{8}$$

where w is a normal vector that is perpendicular to the hyperplane and b is the offset of the hyperplane. As shown in Fig. 2, the parameter $b/||w||$ determines the offset of the hyperplane from the origin along the normal vector w. In the training procedure, the SVM optimizes w and b to maximize the margin (or distance) between the parallel hyperplanes while still separating the data.

The optimization problem and the corresponding hyperplane constraint for nonlinear separable classes can be formulated as

$$\begin{aligned} &\min \ \frac{1}{2}w^T w + C\sum_{i=1}^{m}\varepsilon_i \\ &\text{subject to } y_i(w^T x_i + b) \geq 1 - \varepsilon_i \\ &\varepsilon_i \geq 0, \ \ i = 1, 2, \ldots, m \end{aligned} \tag{9}$$

where the regularization parameter, C, specifies the error penalty; ε_i is a slack variable that represents the error; and y is the classification output. Classification involves determining on which side of the separating hyperplane a test instance x lies and assigning the corresponding class.

SVMs can also be constructed as nonlinear classifiers using kernel functions to replace the product of w and x. To this end, the normal vectors are written as a linear combination of the training data with a weight factor α and a feature map $\phi(x)$ introduced.

$$w = \sum_{i}^{N}\alpha_i y_i x_i \tag{10}$$

$$g(x) = \sum_{i}^{N}\alpha_i y_i (x_i^T x_j) + b \tag{11}$$

$$g(x) = \sum_{i}^{N}\alpha_i y_i \phi(x_i)^T \phi(x_j) + b \tag{12}$$

The scalar product of the feature map of each training point x_i with the test point x_j can then be expressed as a kernel function of the training and testing samples, as shown here in the form of the Gaussian radial basis function.

$$\phi(x_i)^T \phi(x_j) = k(\boldsymbol{x}, \boldsymbol{x}') = \exp(-\gamma \boldsymbol{x} - \boldsymbol{x}'^2) \tag{13}$$

By using nonlinear kernels, classification accuracy improves on data that is not linearly separable in the space. The SVM is not limited to two-class classification problems but can also be used for classification of multiclass problems. The primary approach for the classification of multiclass problems is to reduce them to multiple binary classification problems.

3.2 Random Forest. The RF is a supervised learning method that uses an ensemble of decision trees for classification. RF builds multiple decision trees and merges them to generate a more accurate and stable prediction of class membership [1]. Figure 3 shows the working principle of the RF classifier. The training algorithm applies the general technique of bootstrap aggregating [1] or bagging. Given a training set $X = \{x_1, \ldots, x_n\}$ with responses $Y = \{y_1, \ldots, y_n\}$, bagging selects a random sample with replacement of the training set repeatedly for B times and fits decision trees to these samples. After training, the class prediction for an unseen sample, x', is determined by taking a majority vote [30] as follows:

$$C(x') = \arg\max_i \sum_{j=1}^{B} w_j I(h_j(x') = i) \tag{14}$$

where $w_1, \ldots, w_B$ are weights that sum to 1, which are usually set to $1/B$; $I(\cdot)$ is an indicator function; and $h_j(\cdot)$ is the prediction function for the jth decision tree. In general, employing an RF classifier for classification includes four steps: (1) select random samples from a given dataset, (2) construct a decision tree for each sample and obtain a prediction result from each decision tree, (3) take a vote from each decision tree, and (4) select the prediction with the most votes as the final prediction of class membership.

The RF technique is generally very effective in preventing overfitting, even when the ensemble contains thousands of individual trees. The error rate of RF on unseen samples tends to converge slowly to a limiting value when the number of trees grows rapidly.

3.3 Gaussian Naive Bayes. Bayesian classifiers, which are based on Bayes' theorem, have been found to perform well on a variety of classification problems [3,31,32]. The NB classifier is a well-known representative of the Bayesian classifiers. To formulate a NB classification problem, $X = \{x_1, x_2, \ldots, x_n\}$ is assumed to be the feature vector and $Y = \{y_1, y_2, \ldots, y_n\}$ the class variable. According to Bayes' theorem,

$$P(Y = y_k | x_1, \ldots, x_n) = \frac{P(Y = y_k)P(x_1, \ldots, x_n | Y = y_k)}{\sum_j P(Y = y_j)P(x_1, \ldots, x_n | Y = y_j)} \tag{15}$$

Equation (15) requires an accurate estimate of conditional probability $P(x_1, \ldots, x_n | Y = y_k)$, but this requirement can be eliminated by assuming conditional independence as follows:

$$P(Y = y_k | x_1, \ldots, x_n) = \frac{P(Y = y_k)\prod_i P(x_i | Y = y_k)}{\sum_j P(Y = y_j)\prod_i P(x_i | Y = y_j)} \tag{16}$$

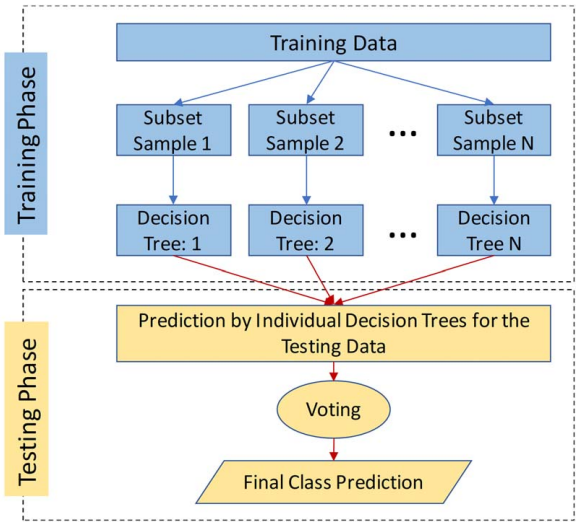

Fig. 3 The working principle of the random forest classifier

Equation (16) requires only an estimate of the probability of each feature value conditioned upon each class in order to estimate the conditional probability, and therefore, the calculation of joint probabilities can be avoided. Thus, in the training stage, the naive Bayes classifier estimates the $P(Y=y_k)$ for all classes and $P(x_i|Y=y_k)$ for all features $i=1,\ldots,n$ and all feature values x_i from the training set.

In the test stage, the class of a test instance X' is predicted with label Y if it leads to the largest value among all the class labels as

$$C(X') = \arg\max_{y_k} P(Y=y_k)\prod_{i=1}^{n} P(x'_i|Y=y_k) \tag{17}$$

A GNB algorithm is a special type of NB algorithm, which is used specifically when the features are defined by continuous values. It assumes that all the features follow a Gaussian distribution. Thus, the conditional probability distribution can be written precisely as

$$P(x_i=x|Y=y_k) = \frac{1}{\sqrt{2\pi\sigma_{ik}^2}} e^{-(1/2)((x-\mu_{ik})/\sigma_{ik})^2} \tag{18}$$

where σ_{ik} represents the kernel width parameter for each feature i and class k. The kernel width can be set heuristically or with an optimization scheme. σ_{ik} can be defined to scale with respect to the features of the training data. One effective way to define σ_{ik} is

$$\sigma_{ik} = \frac{\alpha\hat{\sigma}_{ik}}{N_k^{1/n}} \tag{19}$$

where α is a heuristic scalar, $\hat{\sigma}_{ik}$ is the measured standard deviation of feature i for designs belonging to class k, N_k is the number of samples in class k, and n is the number of features (design variables).

3.4 Neural Network. An artificial neural network is a common supervised learning method for classification. It employs a network learning structure that consists of three types of layers: an input layer, an output layer, and some number of hidden layer(s) [33,34]. The size of the input layer depends on the dimensions of the inputs for the classification problem, while the size of the output layer changes based on the number of different output classes of interest. The size of each hidden layer and the number of hidden layers are determined based on the complexity of the problem. A general feed-forward neural network model is shown in Fig. 4. The output of the kth output node of the network can be represented as

$$y_k = \phi_o\left\{\alpha_k + \sum_j w_{jk}\phi_h\left(\alpha_j + \sum_i w_{ij}x_i\right)\right\} \tag{20}$$

where w and α denote the weights and biases of the neural network, respectively, and ϕ represents the activation function. Common activation functions include linear mappings, such as the identity function that returns the input value, and nonlinear mappings, such as rectified linear units (ReLUs) and logistic functions that can model more complex mappings [35]. When training a neural network, the weights and biases are determined with the use of training sample points together with a training algorithm. The backpropagation neural network (BNN) is one type of the artificial neural network that uses backpropagation as a supervised learning technique for the network training [36]. In the BNN training, the inputs are propagated to the output layer via the hidden layers, and the errors are back propagated to the input layer. Errors are reduced iteratively by adjusting the synaptic weights and biases. To use the BNN as a classification technique, a new data point to be classified serves as an input to the trained BNN model, and the outputs of the trained BNN are the conditional probabilities of class membership that are interpreted according to the decision criterion of Eq. (2) to produce the predicted binary class label.

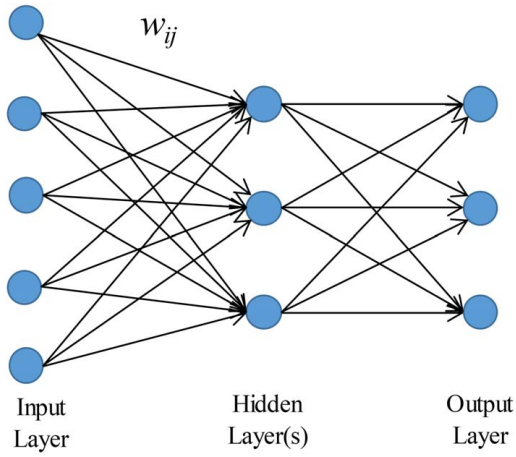

Fig. 4 A general feed-forward neural network model

Neural networks can be structured in many ways. A general fully connected feed-forward neural network model known as the multilayer perceptron (schematic shown in Fig. 4) is utilized in this work for simplicity and generality. Convolutional neural networks are alternate network architectures that have gained popularity when working with spatially structured data such as images or topologies [37,38]. Since spatially structured data are not the focus of the example problems, they are not implemented in this study. Deep learning [39] refers to neural networks with a large number of hidden layers. Deep learning has gained significant popularity in recent years as advances in computing power, such as massively parallel GPU architectures, and custom software, such as TENSORFLOW [40], have enabled the training of deep neural networks to process large datasets with impressive results in the machine learning [38,41] and design communities [37,42]. However, training deep learning networks requires access to very large amounts of training data to fit thousands of associated network parameters. In contrast, as would be the case in many engineering design applications, this study focuses on training classifiers with training sets of only a few hundred to a few thousand samples, which are not sufficient for training deep learning networks. As a result, neural networks with only one or two hidden layers are considered in this work.

4 Methodology for the Comparison Studies

A set of example problems was selected to test the performance of the algorithms outlined in Sec. 3 on problems that represent a cross section of the characteristics discussed in Sec. 2. As described in the Supplemental Materials on the ASME Digital Collection, all problems are specified with continuous and/or discrete variables and performance objectives and constraints that make them amenable to design optimization and exploration. The problem attributes are summarized in Table 2. Further details on problem formulations and performance thresholds for classification can be found in Supplemental Material A available in the Supplemental Materials on the ASME Digital Collection.

For each problem (with the exception of the heavily constrained welded beam problem, see Supplemental Material A.5 for more details), a set of 10,000 samples was generated to serve as training data. Continuous variables were sampled from the Hammersley sequence, and integer variables were assigned from a pseudorandom uniform distribution using the *randint* function in SciPy. Grid search with threefold cross-validation was used to tune the hyperparameters shown in Table 3, and the best model configuration was chosen based on mean accuracy. The available settings for the grid search were chosen based on the Sci-kit learn user guide and previous experience with these classifiers. Higher

Table 2 Problem characteristics for the test bed of example problems

Problem	Characteristics
Rastrigin	Single objective Continuous variables Multimodal Scalable dimensionality
Truss	Multiobjective Mixed continuous/discrete variables Scalable dimensionality
Welded beam	Single objective Continuous variables Heavily constrained
Solar heat exchanger	Multiobjective Mixed continuous/discrete variables Monotonic performance
Disjointed	Single objective Continuous variables Multimodal
Thin plate with hole	Single objective Continuous variables Monotonic performance

resolution parameter grids would be expected to yield further performance improvements at the cost of greater computational expense associated with tuning.

After each algorithm was tuned, its performance was evaluated by predicting the class membership of a static test set of 5000 points. The continuous variables in the test set were sampled from the Halton sequence so that the test set would be distinct from the training set, and integer variables were assigned by the *randint* function in SciPy. All variables were scaled between 0 and 1 for classifier training and testing.

As discussed in Sec. 2, the proportion of training data in each class can have a significant effect on classifier performance and the informational value of classification metrics. The performance thresholds used for binary classification in the example problems tested in this study were chosen to yield reasonably balanced distributions between high- and low-performance training points. The problem specific figures for the proportion of total training data falling in each class can be seen in Table 4. No more than 70% of the training points fell within a given class for any example problem. The presence of reasonably balanced datasets motivated the use of accuracy as the primary metric for comparing classifier performance.

The customizability of classifier algorithms leads to slight differences in implementation. As a result, it is often difficult to replicate results without utilizing the exact same software and computational environment of the original implementation. This conflict has led to the development of standardized software packages with the goal of increasing repeatability. To provide a meaningful benchmarking study of the classifiers, it is important for the study to be repeatable. For that reason, the SVM, NN, RF, and GNB algorithms are implemented from the very popular Sci-kit learn package for PYTHON [35]. The one exception is a custom implementation of the GNB algorithm for tuning purposes, because the GNB algorithm in Sci-kit learn uses maximum likelihood estimation to automatically learn kernel widths and does not permit manual tuning.

Table 3 Tunable parameters for each classification algorithm

Algorithm	Tuned parameters	Available settings
SVM	C	1, 10, 100, 1000
	γ (gamma)	0.01, 0.1, 1, 10
RF	Number of trees	200, 400, 600, 800, 1000
	Maximum tree depth	10, 20, 50, none
NN	Number of neurons in hidden layers	(100), (500), (100, 100), (500, 500)
	Activation function	Logistic, ReLU, identity
GNB	α (alpha)	0.001, 0.005, 0.01, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.8, 1, 2, 3, 5, 6

Table 4 Class balance of the training data for the performance thresholds chosen for each example problem in the study

Problem	High performance (%)	Low performance (%)
2D Rastrigin	44.79	55.21
4D Rastrigin	42.7	57.3
6D Rastrigin	40.78	59.22
3 bar truss	30.15	69.85
5 bar truss	38.85	61.15
20 bar truss	39.26	60.74
Welded beam	43.94	56.06
Solar heat exchanger	46.3	53.7
Disjointed	30.94	69.06
Thin plate	49.0	51.0

5 Classifier Performance on Example Problems

The first investigation focuses on the effect of tuning on the performance of the classification algorithms.[2] Although tuning is recommended for enhancing the performance of a classifier, it is helpful to understand the impact of that tuning on classifier performance versus the use of default tuning parameter values. The selected tuning parameters can vary based on the random data splits used for cross-validation, yielding variations in model performance across instantiations. Classifier performance and sensitivity to tuning parameters can also vary drastically depending on the amount of available training data for the problem. In order to capture this variation, tuning runs were repeated 15 times for convergence plots with low training data (ranging from 50 to 1000 points in increments of 50) and 5 times for convergence plots with high training data (ranging from 500 to 10,000 points in increments of 500). The resulting mean accuracies were plotted with error bars corresponding to the standard error as calculated in Eq. (21), where s is the sample standard deviation and n is the number of observations in the sample.

$$\sigma_x = \pm \frac{s}{\sqrt{n}} \tag{21}$$

Some classification algorithms enable more self-tuning than others, leading to different levels of robustness to tuning among the various classifiers. Figure 5 illustrates this phenomenon, as RF dominates on the illustrated example problems when the algorithms are untuned, indicating that RF provides reasonably good performance without manual tuning. However, SVMs become competitive and typically outperform the RF when tuned, and tuned NNs also exhibit performance on par with or better than RF on two of the problems. Because the GNB includes fewer tunable parameters, its performance increases only modestly with tuning.

An interesting observation from Fig. 5 is that the accuracy of some of the algorithms did not increase monotonically with the number of training points. Despite the use of grid-based tuning

[2]Convergence plots of classifier accuracy were generated with small and large quantities of training data, N_{train}, for every problem described in Section 4 and Supplemental Material A. Selected convergence plots and other specific investigations are included in this section for discussion of classifier performance. Since there are too many plots to display and discuss individually, all remaining convergence plots are included as supplementary data in Supplemental Material B.

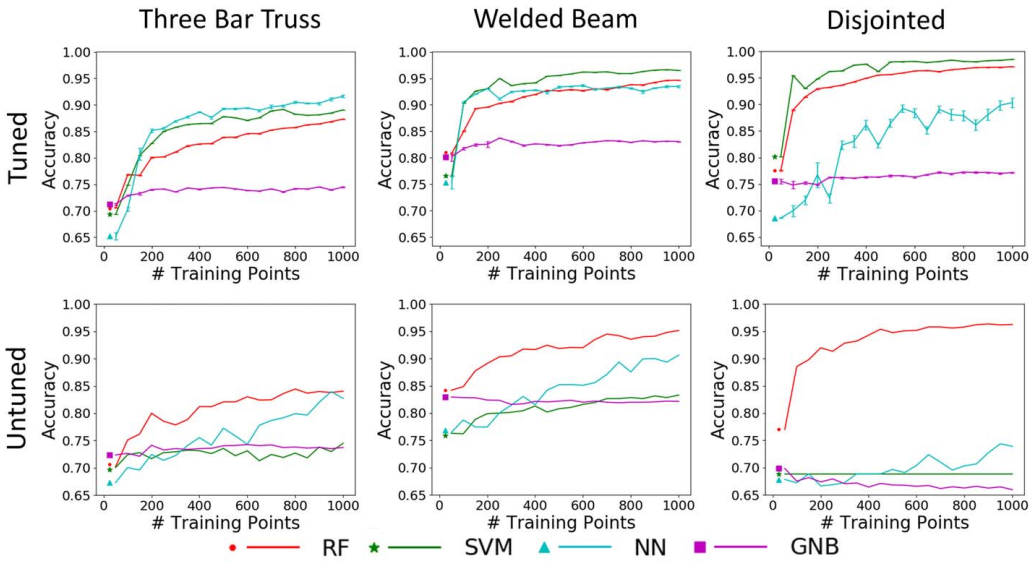

Fig. 5 Convergence plots displaying accuracies for tuned and untuned algorithm instantiations at varying levels of training data for three of the example problems

and cross-validation, some of the classifiers, especially the NN and SVM algorithms, showed significant variation and dips in accuracy, while others such as the RF algorithm showed smoother convergence curves. To investigate this phenomenon, hyperparameter settings and cross-validation mean accuracy scores were examined for classifiers that showed significant variation. It was found that in many of these cases, hyperparameter settings selected during the tuning process changed significantly from point to point as N_{train} varies. Figure 6 shows an example of this examination for a tuned NN classifying the disjointed problem. As shown in the figure, the decrease in accuracy at 1000 training points is accompanied by a reasonably large value for cross-validation accuracy (0.88) relative to neighboring trials. It is likely that these local decreases in accuracy are the result of overfitting, such that the classifier performs well on the training data but more poorly on the test data. The degree of overfitting can vary with the locations of new data points in the training and testing sets and the randomness of the division of training data into cross-validation training sets. Other authors in the engineering design literature have observed similar nonmonotonic decreases in classification performance (e.g., [6,8]).

The next investigation focused on the performance of the tuned classification algorithms with increasing problem dimensionality. Figure 7 plots the accuracy of the tuned algorithms with respect to the number of training points for the 3 bar, 5 bar, and 20 bar truss problems. The Bayesian approach exhibits high asymptotic error for these problems, which may be due to the independence assumptions inherent in its formulation. The smoothing kernel-based approach may also be a poor fit with the presence of coarsely discretized material-type variables in these problems. The SVM and NN perform the best in this mixed-variable problem, performing better than GNB and RF for the 3 and 5 bar problems at all levels of training data. The classifiers are largely bunched together at low training data for the most complex case of 20 truss links with 40 design variables, but the NN and SVM eventually converge to higher accuracies with thousands of training points. The NN, in particular, takes thousands of data samples to obtain competitive accuracy on the 20 bar problem, which may be attributed to the difficulty of fitting the thousands of weights and biases in the larger architectures for the more complex higher-dimensional setting.

The next investigation focuses on a highly multimodal problem; in this case, the 6D Rastrigin function. Figure 8 shows the accuracy of the tuned classification algorithms with respect to the number of training points. Mapping the highly nonlinear function as dimensionality increases is a difficult challenge. Most of the algorithms reach only a modest accuracy of approximately 75% and are unable to improve their mapping of the space with additional training data. The only exception is the GNB approach, which is naturally well suited to represent the regularly distributed, sinusoidal modes of the function with its Gaussian kernels.

Identifying regions of the design space with classifiers becomes more difficult as the performance requirements become stricter and the size of the design space of interest decreases.

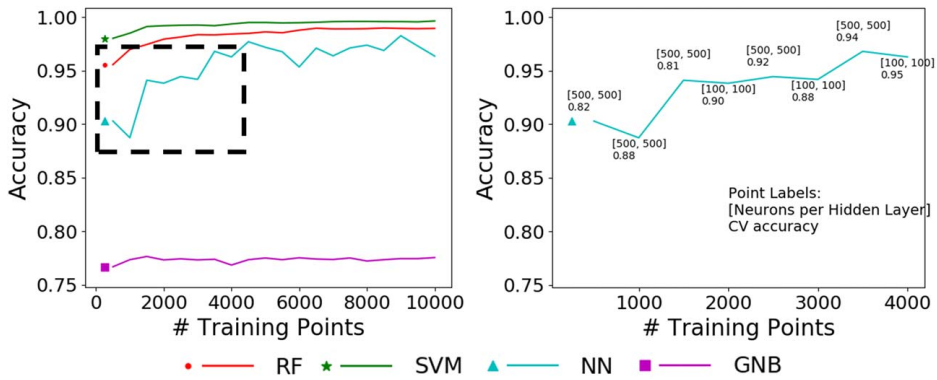

Fig. 6 Variation in hyperparameter settings and cross-validation accuracy for a neural network trained at varying levels of training data for the disjointed example problem

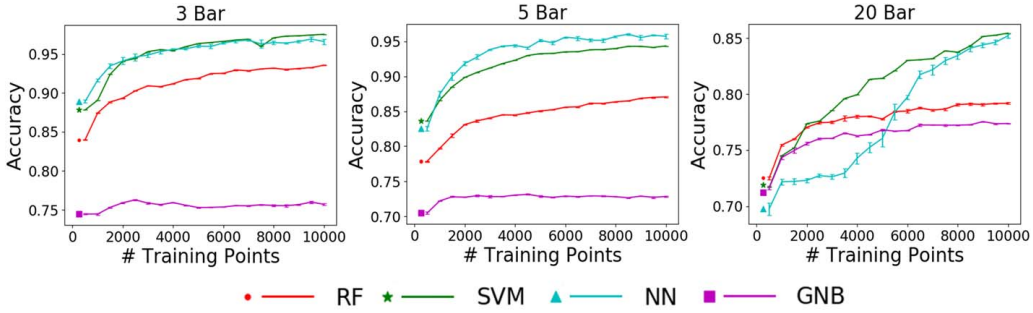

Fig. 7 Accuracy convergence plots for the classification of mixed-variable truss structure problems as the number of design variables increases

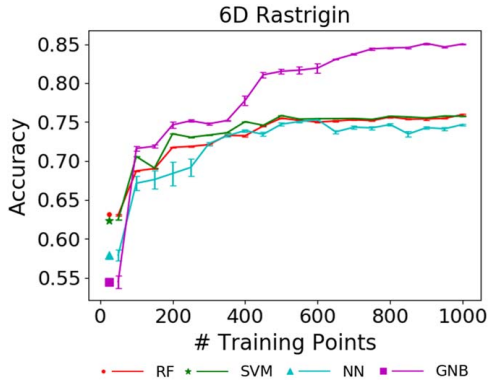

Fig. 8 Accuracy convergence at low data for the highly nonlinear 6D Rastrigin function

The combination of strict performance requirements and training sets with low sampling density may lead to a classifier that predicts very few instances of the positive class even if designs of positive class exist. In multimodal design spaces, this phenomenon can lead to a failure to identify regions of interest. By increasing the performance threshold on the disjointed problem such that high-performance regions shrink and then disappear, the change in each classifier's ability to identify those regions can be observed. For this purpose, the FNR metric is useful to indicate whether each classifier fails to identify high-performance designs. Figure 9 shows the response surface and contour of the disjointed function at the arbitrary performance threshold, $G = 0.9$, such that all candidate designs with performance $G \geq 0.9$ are considered high-performance designs. The high-performance regions are numbered, so they can be referenced during the discussion.

Figure 10(*a*) shows the high-performance regions shrinking as the performance threshold increases incrementally from $G = 0.9$

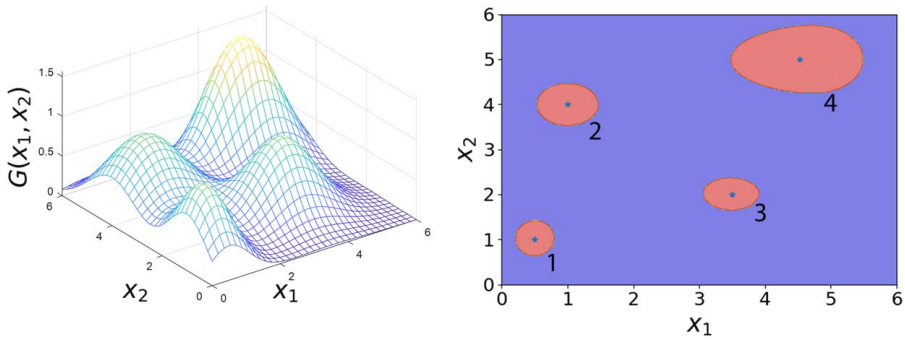

Fig. 9 Surface and contour plot of the disjoined problem at performance threshold $G = 0.9$ with high-performance regions numbered

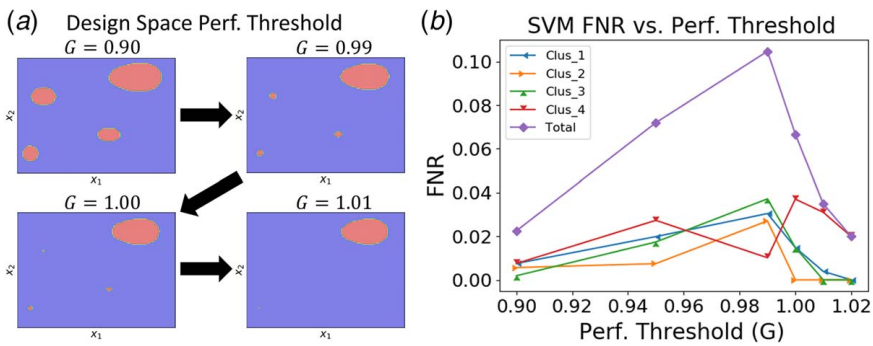

Fig. 10 Illustration of shrinking high-performance regions as performance threshold increases

Table 5 Class membership proportions for various performance thresholds

Performance threshold (G)	High performance (%)	Low performance (%)
$G = 0.054$	91.6	8.4
$G = 0.162$	81.0	19.0
$G = 0.324$	62.4	37.6
$G = 0.593$	31.5	68.5
$G = 0.862$	12.8	87.2
$G = 1.131$	3.8	96.2

to $G = 1.01$. Figure 10(*b*) shows the FNR produced by SVM classifiers trained with 1000 training points and tested with the static test set as performance requirements increase. The false negative rates are examined for each region separately based on the proximity of false negatives to the center of each cluster.

When regions of interest are small, the classifier struggles to identify them as evident in the cumulative FNR curve in Fig. 10(*b*). Notice that the total FNR reaches a peak for $G = 1.0$ when regions 1, 2, and 3 are extremely small and then declines for $G = 1.01$ when those regions have disappeared entirely and therefore do not contribute to the FNR.

Changing the performance threshold changes the balance of the class proportions, leading to imbalanced designs with an overwhelming majority of points belonging to one class, which can affect the predictive power of classifiers. However, some classifiers are more sensitive to class imbalance than others. For the disjointed problem, the degree of imbalance can be adjusted using a range of performance thresholds between the global maximum and minimum. Table 5 shows performance thresholds and the associated class proportions.

TPR and false positive rate (FPR) are compared to measure the predictive capability of a classifier when imbalanced. Figure 11 shows TPR versus FPR for the thresholds shown in Table 5. The SVM and RF algorithms appear to be most robust, as they maintain good classification performance with high TPR and low FPR across all performance thresholds for this problem. Recent work in the engineering design literature has focused on more efficiently sampling the design space for imbalanced problems similar to this one, with the goal of accurately defining the boundaries of a feasible design space with very limited data [6,8], but a comprehensive review of imbalanced classification problems is beyond the scope of this paper.

As mentioned in the introduction, classifiers can be used to identify promising regions of the design space, especially for set-based design applications. If those promising regions are defined by performance thresholds for multiple, conflicting objectives, the overall design space of interest becomes the intersection of promising design

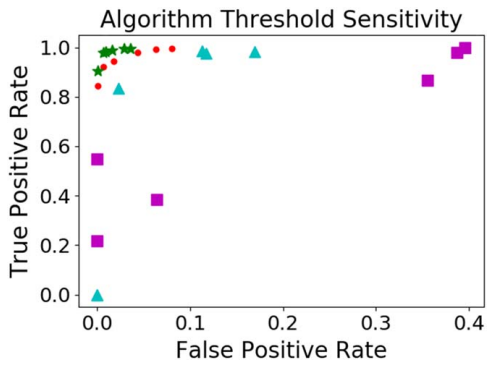

Fig. 11 Classifier performance for five different performance thresholds producing varying rates of imbalance between high- and low-performance classes

spaces for each performance objective. A simplified illustration of this concept is shown for the heat exchanger problem in Fig. 12. The abscissa is the pipe diameter, and the ordinate is the number of pipe loops in the heat exchanger. The top-left subfigure shows the region of the design space with an enclosure volume less than the allowable limit while the top-right subfigure shows the region of the design space that achieves the desired head loss. The bottom-left subfigure shows the mutually satisfactory intersection of these regions, which shrinks the region of interest more than either objective individually. The bottom-right figure shows the classification accuracies on the combined objectives at the current thresholds. This simple multiobjective design optimization problem has a continuous convex region of performance for each metric. More complex problems with nonlinear relationships between variables are likely to produce nonconvex disjoined regions of performance, further increasing the utility of the classifier mappings.

6 Aggregated Results Across Example Problems

Accuracy convergence plots for all example problems (cf. see Supplemental Material for plots in addition to those presented here) were examined to compare relative levels of performance for each algorithm at low, medium, and high levels of training data for each test problem. In addition to quantitative classification accuracy, relative performance of the classifiers on each problem is important since every problem presents a unique set of interacting characteristics. For this reason, the classifier accuracy is provided along with a qualitatively assigned cluster to show which classifiers performed similarly on each given problem. The clusters are assigned red, yellow, or green labels with green representing the best and red representing the worst performance. Figure 13 shows how classifier performance was scored for the disjoint problem with a moderate amount of training data ($N = 1000$). As shown in the figure, the classifiers with the highest accuracy earn the green labels. Classifiers with accuracy approximately 5 percentage points lower than those with the highest accuracy earn the yellow label, and those with accuracy approximately 5 percentage points lower than those with moderate accuracy (yellow) or 10 percentage points lower than those with high accuracy (green) earn the red label. Classifiers that cluster together with similar levels of accuracy (i.e., with accuracies within approximately 5 percentage points of one another) earn the same label. If the best classifiers achieve accuracies of less than 80% for a specific scenario, then none of them earn green labels. Tables 6–8 show the performance scores for all problems and classifiers with low ($N = 200$), medium ($N = 1000$), and high numbers of training points ($N = 10{,}000$) as determined with the same method as for Fig. 13.

As shown in Tables 6–8, none of the classifiers perform particularly well with small quantities of training data. The RF classifier performs better than the others, earning green or yellow labels for all problems. Highly multimodal problems, such as the 4D and 6D Rastrigin, and mixed continuous–discrete problems with relatively large numbers of variables, such as the truss problems, are particularly difficult to classify with small amounts of data. With larger amounts of training data, RF classifiers continue to perform relatively well, with fewer red labels than alternative classifiers. SVM and NN are a close second to the RF in terms of overall performance with larger amounts of training data. However, for highly multimodal and regular problems, such as the Rastrigin function, GNB performs exceptionally well and much better than other algorithms.

7 Discussion

As a general rule, the GNB algorithm is only competitive for highly complex problems with a small amount of data available for training. If computational resources are available to obtain a high number of samples, GNB is likely not the best choice for accuracy. An exception to this rule is the Rastrigin function, where the

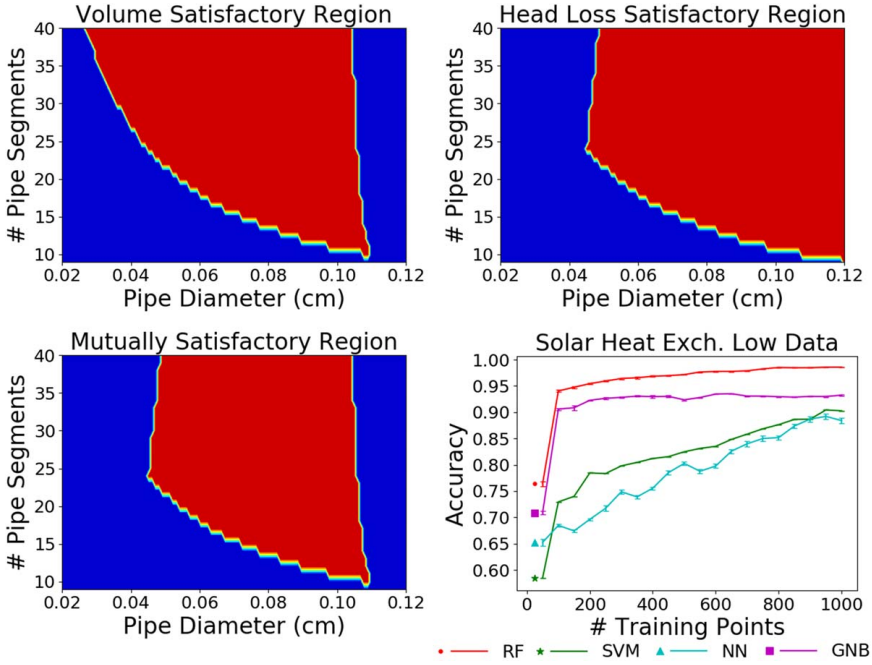

Fig. 12 Design space mappings and convergence plot for the multiobjective heat exchanger problem

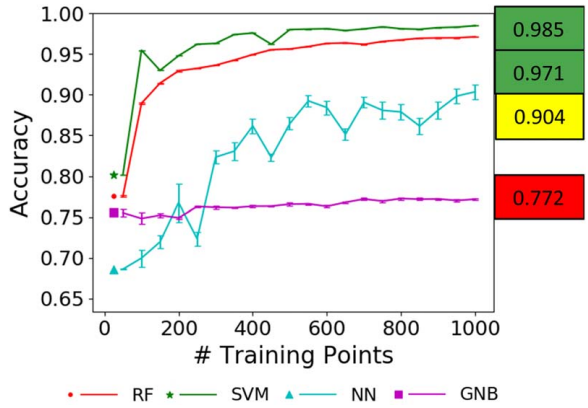

Fig. 13 Tuned classifier performance with example scores as seen in aggregated results table (Color version online.)

tunable Gaussian kernels capture the high nonlinearity in the sinusoidal space quite well. Another exception is the suitability of GNB for sequential sampling as mentioned in the discussion of generative algorithms in Sec. 2. The NN performs well on the mixed-variable problems for large datasets, but the convergence plots show that the NN often requires more data than other algorithms to converge to a relatively high accuracy on these problems and is among the worst performers for moderate and small quantities of data. This is a common theme for NNs, as the large number of model parameters often requires a significant amount of data to obtain a reasonable fit.

The RF shows its versatility by performing well on all problems and particularly well on the nonlinear problems such as the Rastrigin and the disjoint example. In addition, the RF is generally the most robust algorithm in the sense that it performs the best without tuning the hyperparameters. RF may, therefore, be a good candidate for beginners or time-crunched users who do not wish to undertake the tuning process to achieve satisfactory accuracy. However, the performance of RF seems to degrade as the number of design variables increases. Finally, the SVM also performs well on most problems with the exception of the Rastrigin. The SVM converges quickly on simple problems and can continue converging to higher accuracies for complex problems as thousands of training points are accrued. However, the SVM is very sensitive to parameter tuning and engineers should keep this in mind when selecting the best approach for their needs.

Overfitting is a concern for any classifier algorithm but some are more susceptible than others. It is very difficult to quantify and compare algorithms' likeliness to overfit because there is no general definition that is model independent. As a good practice, tuning hyperparameters with cross-validation works to mitigate overfitting by incentivizing out-of-sample performance. For better results, a high resolution grid of hyperparameter settings or an advanced sequential selection technique should be used when tuning. Observed variation with respect to the number of training points in convergence plots shows that algorithms that self-tune such as RF generally suffer less from overfitting to training data. On the other hand, algorithms that self-tune but have a very large number of parameters (like NN) may require prohibitively large quantities of training data.

Table 6 Relative classifier performance on each example problem with 200 training points (Color version online.)

	3 bar	5 bar	20 bar	2D Rastrigin	4D Rastrigin	6D Rastrigin	Disjointed	Welded beam	Heat exchanger	Thin plate
RF	0.799	0.708	0.684	0.800	0.759	0.717	0.929	0.896	0.955	0.969
NN	0.797	0.671	0.651	0.735	0.750	0.720	0.767	0.931	0.696	0.981
SVM	0.768	0.656	0.661	0.756	0.752	0.735	0.948	0.931	0.785	0.974
GNB	0.741	0.664	0.674	0.844	0.764	0.733	0.749	0.826	0.923	0.917

Table 7 Relative classifier performance on each example problem with 1000 training points (Color version online.)

	3 bar	5 bar	20 bar	2D Rastrigin.	4D Rastrigin.	6D Rastrigin.	Disjointed	Welded beam	Heat exchanger	Thin plate
RF	0.849	0.757	0.753	0.842	0.789	0.760	0.971	0.947	0.986	0.969
NN	0.897	0.828	0.695	0.755	0.754	0.747	0.904	0.935	0.884	0.982
SVM	0.888	0.792	0.714	0.755	0.761	0.757	0.985	0.965	0.903	0.982
GNB	0.727	0.684	0.742	0.940	0.883	0.849	0.772	0.831	0.933	0.952

Table 8 Relative classifier performance on each example problem with 10,000 training points (Color version online.)

	3 bar	5 bar	20 bar	2D Rastrigin	4D Rastrigin	6D Rastrigin	Disjointed	Welded beam	Heat exchanger	Thin plate
RF	0.933	0.871	0.792	0.951	0.829	0.792	0.990	0.999	0.995	0.973
NN	0.963	0.958	0.852	0.755	0.766	0.763	0.978	0.976	0.994	0.991
SVM	0.965	0.944	0.854	0.755	0.767	0.769	0.996	0.986	0.948	0.996
GNB	0.711	0.729	0.774	0.971	0.951	0.943	0.774	0.832	0.926	0.966

All of the convergence studies in this work are based on the assumption that training data are acquired from a simple space-filling sampling strategy. More focused sampling strategies could provide superior information to the algorithms and improve classification performance for smaller data sets, especially for imbalanced data sets. For instance, some recent work has been conducted on heuristic strategies that sample near existing class boundaries to help resolve these boundaries more clearly within a limited sampling budget (e.g., [6,8]).

8 Conclusion

This work presented a set of simple engineering-related example problems that can serve as testbeds for future research in machine learning for engineering design exploration and optimization. The problems covered a range of characteristics that designers often encounter in engineering optimization applications. The accuracy of four common classification algorithms was tested on each problem with varying levels of training data. No single classification approach emerged as universally dominant, underscoring the need to select the appropriate machine learning tool depending on the nature of the problem.

The results of this study can help inform engineers in choosing an appropriate algorithm for their problem by inspecting classifier performance on similar problems. The results can also give engineers an intuition for the amount of training data that may be required to reach satisfactory accuracy levels depending on problem complexity and dimensionality. This knowledge would be useful for planning experiments in the early stages of design. Further work extending the study to include more design problems would be valuable to increase the coverage of the example problems and ensure that engineers can glean even more relevant insight from performance across a wider array of problem characteristics. Those design problems could include design spaces of higher dimensionality, greater imbalance between classes in the training data set, and problems for which greater numbers of classes might be useful (e.g., infeasible, feasible with moderate performance, and feasible with high performance).

Acknowledgment

The authors gratefully acknowledge funding from the National Science Foundation (NSF) (Grant No. EFRI-1641078; Funder ID: 10.13039/501100008982) and the Defense Advanced Research Projects Agency (DARPA; Funder ID: 10.13039/100000185). The DARPA-sponsored work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory (Grant No. DE-AC52-07NA27344; Funder ID: 10.13039/100006227) with funding from the DARPA. The views, opinions, and/or findings expressed are those of the authors and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. government.

References

[1] James, G., Witten, D., Hastie, T., and Tibshirani, R., 2013, *An Introduction to Statistical Learning*, Springer, New York.
[2] Sobek, D., Ward, A., and Liker, J., 1999, "Toyota's Principles of Set-Based Concurrent Engineering," Sloan Manage. Rev., **40**(2), pp. 67–84.
[3] Shahan, D., and Seepersad, C., 2012, "Bayesian Networks for Set-Based Collaborative Design," ASME J. Mech. Des., **134**(7), p. 071001.
[4] Matthews, J., Klatt, T., Morris, C., Seepersad, C., Haberman, M., and Shahan, D., 2016, "Hierarchical Design of Negative Stiffness Metamaterials Using a Bayesian Network Classifier," ASME J. Mech. Des., **138**(4), p. 041404.
[5] Morris, C., Bekker, L., Haberman, M., and Seepersad, C., 2018, "Design Exploration of Reliably Manufacturable Materials and Structures With Applications to Negative Stiffness Metamaterials and Microstereolithography," ASME J. Mech. Des., **140**(11), p. 111415.
[6] Galvan, E., Malak, R., Gibbons, S., and Arroyave, R., 2016, "A Constraint Satisfaction Algorithm for the Generalized Inverse Phase Stability Problem," ASME J. Mech. Des., **139**(1), p. 011401.
[7] Rosen, D., 2015, "A Set-Based Design Method for Material-Geometry Structures by Design Space Mapping," ASME IDETC, Boston, MA. Paper No. DETC2015-46760.
[8] Chen, W., and Fuge, M., 2017, "Beyond the Known: Detecting Novel Feasible Domains Over an Unbounded Design Space," ASME J. Mech. Des., **139**(11), p. 111405.
[9] Galvan, E., and Malak, R., 2015, "P3GA: An Algorithm for Technology Characterization," ASME J. Mech. Des., **137**(1), p. 011401.
[10] Malak, R., and Paredis, C., 2010, "Using Support Vector Machines to Formalize the Valid Input Domain of Predictive Models in Systems Design Problems," ASME J. Mech. Des., **132**(10), p. 101001.
[11] Backlund, P., Shahan, D., and Seepersad, C., 2015, "Classifier-Guided Sampling for Discrete Variable, Discontinuous Design Space Exploration: Convergence and Computational Performance," Eng. Optim., **47**(5), pp. 579–600.
[12] Basudhar, A., and Missoum, S., 2013, "Reliability Assessment Using Probabilistic Support Vector Machines," J. Reliab. Saf., **7**(2), pp. 156–173.
[13] Basudhar, A., Missoum, S., and Sanchez, A., 2008, "Limit State Function Identification Using Support Vector Machines for Discontinuous Responses and Disjoint Failure Domains," Probabilistic Eng. Mech., **23**(1), pp. 1–11.
[14] Patel, J., and Choi, S.-K., 2014, "An Enhanced Classification Approach for Reliability Estimation of Structural Systems," J. Intell. Manuf., **25**(3), pp. 505–519.
[15] Song, H., Choi, S.-K., Lee, I., Zhao, L., and Lamb, D., 2013, "Adaptive Virtual Support Vector Machine for Reliability Analysis of High-Dimensional Problems," Struct. Multidiscipl. Optim., **47**(4), pp. 479–491.
[16] Wang, Z., and Wang, P., 2015, "An Integrated Performance Measure Approach for System Reliability Analysis," ASME J. Mech. Des., **137**(2), p. 021406.
[17] Hu, Z., and Du, X., 2018, "Integration of Statistics- and Physics-Based Methods—A Feasibility Study on Accurate System Reliability Prediction," ASME J. Mech. Des., **140**(7), p. 074501.
[18] Wang, P., Wang, Z., and Almaktoom, A., 2014, "Dynamic Reliability-Based Robust Design Optimization With Time-Variant Probabilistic Constraints," Eng. Optim., **46**(6), pp. 784–809.
[19] Zhuang, X., and Pan, R., 2012, "A Sequential Sampling Strategy to Improve Reliability-Based Design Optimization With Implicit Constraint Functions," ASME J. Mech. Des., **134**(2), p. 021002.
[20] Wang, Y., Yu, X., and Du, X., 2015, "Improved Reliability-Based Design Optimization With Support Vector Machine and Its Application in Aircraft Wing Design," Math. Probl. Eng., **1**, pp. 1–14.

[21] Xiong, B., and Tan, H., 2017, "New Structural Reliability Method With Focus on Important Region and Based on Adaptive Support Vector Machines," Adv. Mech. Eng., **9**(6), pp. 1–12.
[22] White, L. M., West, T. K., and Brune, A. J., 2018, "Reliability-Based Design of Thermal Protection Systems With Support Vector Machines," 2018 Joint Thermophysics and Heat Transfer Conference, p. 3440.
[23] Ng, A. Y., and Jordan, M. I., 2002, "On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes," Advances in Neural Information Processing Systems, pp. 841–848.
[24] sci-kit learn Developers, 2019, "scikit-learn v0.20.3 User Guide," https://scikit-learn.org/stable/user_guide.html, Accessed February 2019.
[25] Louppe, G., 2014, "Understanding Random Forests From Theory to Practice," PhD dissertation, University of Liege, Liege.
[26] Elkan, C., 1997, *Boosting and Naive Bayesian Learning*, University of California, San Diego, CA.
[27] Saliya, N., Khoshgotaar, T., and Van Hulse, J., 2009, "A Study on the Relationships of Classifier Performance Metrics," IEEE International Conference on Tools With Artificial Intelligence, Newark, NJ, pp. 59–66.
[28] Cristianini, N., and Shawe-Taylor, J., 2000, *An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods*, Cambridge University Press, Cambridge.
[29] Wang, P., Tamilselvan, P., and Hu, C., 2014, "Health Diagnostics Using Multi-Attribute Classification Fusion," Eng. Appl. Artif. Intell., **32**, pp. 192–202.
[30] Tamilselvan, P., and Wang, P., 2013, "Failure Diagnosis Using Deep Belief Learning Based Health State Classification," Reliab. Eng. Syst. Saf., **115**, pp. 124–135.
[31] Rish, I., 2001, "An Empirical Study of the Naive Bayes Classifier," Proceedings of IJCAI-01 Workshop on Empirical Methods, Aug. 4, pp. 41–46.
[32] Zhang, M., Pena, J., and Robles, V., 2009, "Feature Selection for Multi-Label Naive Bayes Classification," J. Inform. Sci., **179**(19), pp. 3218–3229.
[33] Heermann, P., and Khazenie, N., 1992, "Classification of Multispectral Remote Sensing Data Using a Back-Propagation Neural Network," IEEE Trans. Geosci. Remote Sens., **30**(1), pp. 81–88.
[34] Li, J., Du, Q., and Li, Y., 2016, "An Efficient Radial Basis Function Neural Network for Hyperspectral Remote Sensing Image Classification," Soft Comput., **20**(12), pp. 4753–4759.
[35] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., and Vanderplas, J., 2011, "Scikit-learn: Machine Learning in Python," J. Mach. Learn. Res., **12**(Oct.), pp. 2825–2830.
[36] Rumelhart, D., Hinton, G., and Williams, R., 1986, "Learning Representations by Back-Propagating Errors," Nature, **323**(6088), pp. 533–536.
[37] Banga, S., Gehani, H., Bhilare, S., Patel, S., and Kara, L., 2018, "3D Topology Optimization Using Convolutional Neural Networks," e-print arXiv:1808.07440.
[38] Krizhevsky, A., Sutskever, I., and Hinton, G., 2012, "ImageNet Classification With Deep Convolutional Neural Networks," NIPS, Lake Tahoe, NV, pp. 1097–1105.
[39] LeCun, Y., Bengio, Y., and Hinton, G., 2015, "Deep Learning," Nature, **521**(7553), pp. 436–444.
[40] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., and Ghemawat, S., 2015, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems,".
[41] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y., 2014, "Generative Adversarial Networks," Advances in Neural Information Processing Systems, pp. 2672–2680.
[42] Chen, W., Chiu, K., and Fuge, M., 2019, "Aerodynamic Design Optimization and Shape Exploration Using Generative Adversarial Networks," AIAA Scitech Forum.