Evaluating Nonlinear Decision Trees for Binary Classification Tasks with Other Existing Methods

Yashesh Dhebar[*], Sparsh Gupta[†], and Kalyanmoy Deb[*]
[*]Computational Optimization and Innovation (COIN) Laboratory, Michigan State University, East Lansing, Michigan, USA
{dhebarya, kdeb}@egr.msu.edu
[†]Department of Mechanical Engineering, Indian Institute of Technology, Kanpur, Uttar Pradesh, India, sparshg@iitk.ac.in

COIN Report Number 2020016

Abstract—**Classification of datasets into two or more distinct classes is an important machine learning task. Many methods are able to classify binary classification tasks with a very high accuracy on test data, but cannot provide any easily interpretable explanation for users to have a deeper understanding of reasons for the split of data into two classes. In this paper, we highlight and evaluate a recently proposed nonlinear decision tree approach with a number of commonly used classification methods on a number of datasets involving a few to a large number of features. The study reveals key issues such as effect of classification on the method's parameter values, complexity of the classifier versus achieved accuracy, and interpretability of resulting classifiers.**

Index Terms—**Interpretable AI, Classification, Genetic programming, Nonlinear decision trees, Generalized additive method.**

I. Introduction

The task of a binary classification algorithm is to arrive at a classifier involving one or more features from a set of two-labelled dataset, so that the resulting classifier is able to correctly classify unseen test datasets of similar type into two classes with near 100% accuracy. The classifier can be a mathematical function of features, or a network in which features act as input to the network and a binary output reveals the class of a data point, or a decision tree in which a data point flows from root node to internal nodes according to the decisions made at each node and ending up with a class identification at one of the leaf nodes. Each representation (a mathematical function, a network or a decision tree) can be *simple*, involving fewer terms and structure, or complex. However, it is well understood that the complexity of a classifier and its achievable testing accuracy are closely linked. A classifier which is simple most likely cannot be very accurate and vice versa. Fortunately, most classification methods are involved with one or more algorithmic parameters that can be tuned to achieve a desired above-mentioned accuracy-complexity trade-off.

An important matter which is getting a lot of attention in the classification literature is the interpretability of obtained classifiers. Besides accurately classifying new data into its true class, the users are getting more interested in learning how the classifier is able to classify a data into its true class with an easy-to-explain logic. If a classifier has a complex structure (to achieve a high enough classification accuracy), the resulting classifier may be too complex to interpret and explain. Hence, a classification method capable of producing a good balance between accuracy and interpretability is desired.

In this paper, we consider a number of popular classification methods – a linear decision tree (CART), support vector machines (SVMs), generalized additive models (GAMs), genetic programming (GP), and a recently proposed nonlinear decision tree (NLDT) approach. We discuss their working principles in brief and provide their advantages and disadvantages in Section II. After providing the effect of their parameters on the obtained accuracy-complexity trade-off, we compare them on 19 different binary classification problems (described in Section III) having two to 500 features in Section IV. Finally, conclusions are drawn in Section V.

II. Existing Binary Classification Methods

In this section, we provide a brief description of a few popular existing classification methods pertaining to binary classification tasks.

A. Classification and Regression Trees (CART)

Classification and regression trees or CART have been thought of as a popular choice, since the resulting classifier assumes the structure of a *decision tree*. Decision trees make decision using a logical hierarchical representation, which is also common to the way in which a human mind operates. The overall structure is represented in an inverted tree format, with the root node at the top and leaf nodes as the terminals. The data in the root node undergoes recursive binary splitting [17], [18] to create child nodes in the decision tree. One restriction of the CART approach is that splits in decision trees are axis parallel in nature and operate on only one feature (i.e. $x_i \leq \tau_i^*$), as shown in Figure 1.

The spilt rule $x_i \leq \tau$ splits the data in the conditional node (P) (the node where split is occurring) into two non-overlapping subsets: left child node (L) and right child node (R). The quality of split is computed by using an *impurity* metric, like the Gini score, entropy, or others. An impurity

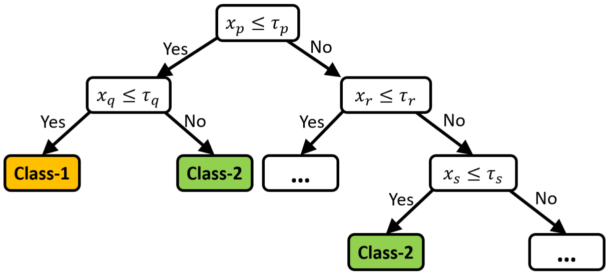


Fig. 1: A CART decision tree splitting the flow of a data into one of the two branches, finally leading to a class identification at its terminal leaf nodes.

metric quantifies the purity (or impurity) of data distribution in a given node:

$$\text{Gini} = 1 - \sum_{i}^{c} \frac{N_i}{N}, \tag{1}$$

where c is the number of classes (which is two in our case), N is the total number of data points in the node and N_i is the number of data points in the given node belonging to class i. The quality of split (S) can then be computed using the following equation:

$$S = \frac{N_L}{N_P}\text{Gini}(L) + \frac{N_R}{N_P}\text{Gini}(R), \tag{2}$$

where N_P is the total number of points in the given parent node undergoing a split, N_L and N_R are number of points belonging to left child node (for which $x_i \leq \tau$ is TRUE) and right child node (for which $x_i \leq \tau$ is FALSE), respectively. The optimal feature x_i and its optimal threshold value τ_i are determined using a greedy algorithm, or through a univariate optimization method. The (x_i, τ_i) combination giving the lowest S-value (Eq. 2) is chosen to conduct the split. A recursive algorithm ID3 [18] or C4.5 [3], [17] is employed to grow the tree.

The tree is allowed to grow up to a prespecified maximum depth when the node under consideration meets one of the termination criteria. The nodes that do not undergo any further split are referred to as *leaf* nodes. The leaf node is assigned with a *class* based on the distribution of data within the node. Since the split rule at each conditional node assumes a very simple linear structure, i.e. $x_i \leq \tau$, many splits are required for a complex classification task, thereby resulting into a complicated decision tree topology, which may not be fathomable by a human.

Some advantages and disadvantages of the CART method for binary classification are listed below:

Advantages:

- Fast to train.
- Easily interpretable rules (linear and each rule involves only one of the features) in each node.
- Many source codes and packages available for quick implementation.

Disadvantages:

- The execution requires a number of tunable parameters: (i) maximum depth of the tree, (ii) total number of splits, (iii) threshold impurity level and (iv) minimum number of classified data points in a node for terminating any further split and declaring it as a leaf node. Available codes come with default values, which may not produce a desired accuracy or end up with a huge decision tree.
- The method has a tendency to overfit the training data, leading to poor performance on test data. Pruning and other methods, like bagging and boosting, are suggested [12], [13], [22] to overcome this effect.
- The tree eventually grows as a result of many hierarchical successive spitting and becomes topologically very complex for humans to fathom.
- Clearly, the method is not suitable for datasets which require a complex, nonlinear, and linked feature relationships for achieving an accurate classification.

In our experiments in this paper, we use Matlab's *fitctree()* routine with its default parameter settings to generate CART based classifiers.

B. Support Vector Machines (SVMs)

For a separable dataset, support vector machine (SVM) algorithm attempts to derive a decision boundary in the form of a single mathematical equation as shown below:

$$y(\mathbf{x}) = \mathbf{w}^T \boldsymbol{\phi}(\mathbf{x}) + b, \tag{3}$$

where $\boldsymbol{\phi}(\mathbf{x})$ is a set of feature transformation functions which can be either linear or non-linear functions of feature vector $\mathbf{x}$, $\mathbf{w}$ is a weight vector and b is a bias term. A conceptual understanding of SVM is provided in Figure 2. For a binary classification task involving class labels $t = -1$ or $t = 1$, an optimal hyper-surface is derived by maximizing the margin between two classes, as shown by $y = 0$ line in the figure. Points with $y \leq -1$ belong to one class and points with $y \geq 1$ belong to another class. The points which fall on $y = 1$ and $y = -1$ are called support vectors, as they alone decide the classifier. However, for non-separable datasets, such as the scenario shown in Figure 3, a *soft* margin approach is used to allow some data points within $|y| < 1$ (margin) while training the SVM. These points are also declared as support vectors in addition to the points on the margin.

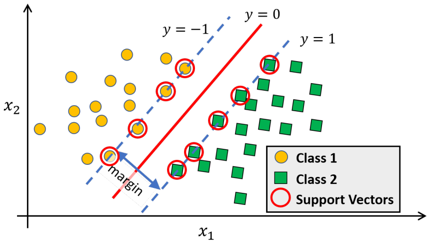


Fig. 2: SVM on separable datasets with a hard margin.

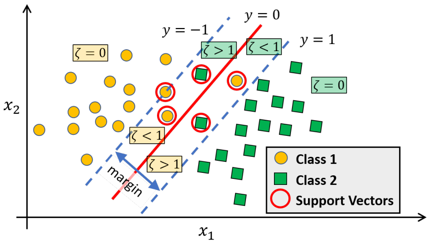


Fig. 3: SVM with non-separable datasets with a soft margin.

To identify the classifier and the support vectors, the underlying optimization problem is solved:

$$\begin{aligned} \text{Minimize:} \quad & \tfrac{1}{2}||w||^2 + C\sum_{i=1}^{N}\zeta_i, \\ \text{subject to}: \quad & t_i(\mathbf{w}^T\boldsymbol{\phi}(\mathbf{x}_i)+b) \geq 1-\zeta_i, \\ & \zeta_i \geq 0, \quad i=1,2,\ldots,N, \end{aligned} \tag{4}$$

where t_i is the *true* class label (either 1 or -1) of the datapoint, ζ_i is the distance of i-th data point from its representative margin, thus $\zeta_i = \max\,[0, 1-t_iy(\mathbf{x}_i)]$ (where value of $y(\mathbf{x}_i)$ is estimated from Eq. 3). C is a penalty parameter which is used to enhance *generalizability* by compromising with *training accuracy*. It is also aimed to balance the complexity of the classifier (described with the number of non-zero terms of $\mathbf{w}$) and soft support vectors within the margin and is an important parameter. With lower values of C, broader margin (with some misclassification of training datapoints) is achieved while for large values of C, misclassification of training datapoints is heavily penalized and so narrower margin is achieved.

Using a kernel trick [1] $k(\mathbf{x}_p,\mathbf{x}_q) = \boldsymbol{\phi}(\mathbf{x}_p)^T\boldsymbol{\phi}(\mathbf{x}_q)$ Eq. 3 is transformed into the following:

$$y(\mathbf{x}) = \sum_{i=1}^{N} a_it_ik(\mathbf{x},\mathbf{x}_i)+b, \tag{5}$$

where a_i is a Lagrange multiplier which is obtained by converting the optimization problem of maximizing the margin (Eq. 4) to a dual Lagrangian representation [1]:

$$\begin{aligned} \text{Min:} \quad & L(\mathbf{a}) = \sum_{i=1}^{N}a_i - \tfrac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}a_ia_jt_it_jk(\mathbf{x}_i,\mathbf{x}_j), \\ \text{s.t.} \quad & a_i \in [0,C], \quad \sum_{i=1}^{N}a_it_i = 0. \end{aligned} \tag{6}$$

Classical gradient based algorithms can then be employed to find a_i. In Eq. 5, data points $\mathbf{x}_i$ for which $a_i = 0$ do not contribute in the equation of the split rule (Eq. 5) and data points for which $a_i > 0$ are called support vectors for the SVM classifier and they dictate the overall length of the classifier's equation (Eq. 5). The penalty parameter C has to be tuned to efficiently derive the decision boundary. Lower value of C makes the classifier more generalizable. $C=\infty$ (hard margin) attempts to achieve near 100% training accuracy and hence is prone to overfitting. In our case, we use scikit-learn's [16] SVM module and set $C = 1,000$. We use RBF (or Gaussian) kernel function. Table I shows results for various settings of C on some datasets considered in our study.

Advantages:

- Good in many classification tasks and scales well with dimension of the dataset.
- Since a classical optimization solver is employed to solve the Lagrangian dual problem (Eq. 6), the training is fast.
- Can generalize well through an appropriate choice of C.
- Many source codes and packages are available for rapid implementation of SVM on various languages like python [16] and Matlab.

Disadvantages:

- The penalty parameter C acts as a regularization parameter and needs to be properly identified and tuned while working on different datasets.
- The knowledge regarding *separability* of datapoints belonging to different classes is required to properly tune C and in practical problems, this information is not available.
- The kernel function $k(\mathbf{x}_p,\mathbf{x}_q)$ (Eq. 5) needs to be chosen.
- Since only one rule is found through SVM, the resulting rule might involve many terms, thereby making the overall classifier uninterpretable.

C. Generalized Additive Models (GAMs)

For a binary classification task involving two classes: Class 1 ($y=0$) and Class 2 ($y=1$), the GAM based classifier [10], [21] estimates the probability of a data point belonging to class $y=1$ (i.e. $P(y=1|\mathbf{x})$)[1] as $\hat{y}(\mathbf{x})$ using the following equation

$$\hat{y}(\mathbf{x}) = \frac{1}{1+e^{-g(\mathbf{x})}}, \tag{7}$$

where $g(\mathbf{x})$ is referred to as *link function* [14]. The link function $g(\mathbf{x})$ in GAM is expressed as a sum of non-linear functions as shown below:

$$g(\mathbf{x}) = f_1(\mathbf{x}) + f_2(\mathbf{x}) + \cdots + f_M(\mathbf{x}) + \beta_0, \tag{8}$$

where β_0 is a constant and $f_i(\mathbf{x})$ are scalar valued nonlinear functions. The functional form of $f_i(\mathbf{x})$ and total number of such nonlinear functions is pre-specified by the user. Modelling of link function $g(\mathbf{x})$ using Eq. 8 makes GAMs more generalizable than its precursor: generalized linear models (GLMs) [15], which involves only linear terms.

In our experiments, we use penalized B-splines to model non-linearity of each feature separately (i.e. referring to Eq. 8, $f_i(\mathbf{x}) = s_i(x_i)$). Thus, the g-function in our case is given by

$$\begin{aligned} & g(\mathbf{x}) = s_1(x_1) + s_2(x_2)\ldots s_d(x_d) + \beta_0, \\ \text{where:} \quad & s_i(x_i) = \sum_{j=1}^{K_i} B_j^{(q_i)}(x_i)\beta_j = \mathbf{B}_i'(x_i)\boldsymbol{\beta}_i. \end{aligned} \tag{9}$$

Here, $s_i(x_i)$ denotes a spline function corresponding to i-th feature, $B_j^{(q_i)}(x_i)$ indicates the basis function of order q_i, β_j

[1] probability of datapoint belonging to other class (i.e. $y=0$) will be $1-\hat{y}$.

TABLE I: SVM Result for different values of penalty parameter C. For each dataset, the first row represents the testing accuracy and the second row represents complexity (number of support vectors). $C = 1000$ gives overall best performance.

Pen. Param.	DS1	DS2	DS3	DS4	Truss	WeldedBeam
$C = 1$	94.75 ± 1.97	95.24 ± 0.00	96.93 ± 1.87	45.20 ± 4.24	77.31 ± 2.16	98.83 ± 0.70
	191.94 ± 4.38	16.56 ± 0.80	64.68 ± 2.16	138.76 ± 1.99	343.76 ± 9.51	47.26 ± 3.00
$C = 10$	98.42 ± 1.16	95.24 ± 0.00	99.32 ± 0.70	68.77 ± 4.00	81.29 ± 2.19	99.53 ± 0.42
	58.70 ± 2.90	30.46 ± 0.64	26.68 ± 1.88	262.60 ± 4.76	258.52 ± 10.85	17.86 ± 1.73
$C = 1,000$	$\mathbf{99.88 \pm 0.33}$	$\mathbf{99.70 \pm 0.50}$	$\mathbf{99.75 \pm 0.58}$	$\mathbf{96.63 \pm 1.35}$	$\mathbf{88.54 \pm 1.60}$	$\mathbf{99.63 \pm 0.38}$
	$\mathbf{8.36 \pm 0.87}$	$\mathbf{8.56 \pm 0.67}$	$\mathbf{10.60 \pm 0.92}$	$\mathbf{56.70 \pm 3.13}$	$\mathbf{176.22 \pm 7.87}$	$\mathbf{7.88 \pm 0.86}$

Pen. Param.	m-DS1	m-DS2	m-DS3	Cancer-10	Cancer-30
$C = 1$	99.77 ± 0.67	95.24 ± 0.00	99.97 ± 0.23	$\mathbf{97.15 \pm 1.08}$	90.83 ± 1.83
	70.22 ± 2.23	16.18 ± 0.59	36.54 ± 1.72	69.98 ± 6.51	106.88 ± 4.44
$C = 10$	$\mathbf{100.00 \pm 0.00}$	98.89 ± 0.85	$\mathbf{100.00 \pm 0.00}$	95.98 ± 1.13	91.94 ± 1.36
	26.42 ± 1.46	14.40 ± 0.89	12.60 ± 0.98	56.22 ± 6.43	81.66 ± 4.54
$C = 1,000$	99.93 ± 0.33	$\mathbf{99.97 \pm 0.22}$	$\mathbf{100.00 \pm 0.00}$	95.23 ± 1.09	$\mathbf{95.08 \pm 1.65}$
	$\mathbf{7.38 \pm 0.75}$	$\mathbf{5.34 \pm 0.55}$	$\mathbf{8.82 \pm 1.01}$	$\mathbf{52.36 \pm 4.91}$	$\mathbf{58.74 \pm 5.18}$

are scalar coefficients and K_i is the total number of basis functions used to model the spline. The order of spline (i.e. q_i) and the number of basis-functions K_i is user-specified.

Once the structure of link function $g(\mathbf{x})$ is specified, an optimization algorithm is invoked to learn parameters corresponding to basis functions $\beta_j^{(q_i)}(x_i)$ and coefficients β_j with an objective to minimize the error between the estimated value of probability ($\hat{y}(\mathbf{x})$ Eq. 7) and the actual y values across the dataset. To make the resulting model more generalize and simple, a second-order smoothing is employed. Thus, using Eq. 7 and 9, the overall optimization problem translates to minimizing the following function:

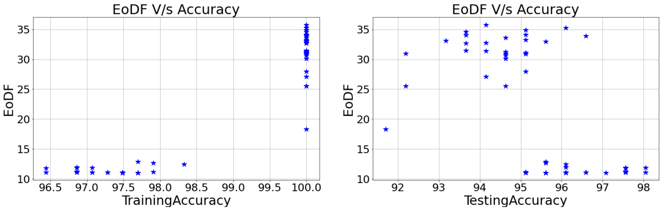
$$\text{Min: } F(\mathbf{B}',\boldsymbol{\beta})=\sum_{i=1}^{N}(y_i-\hat{y}_i(\mathbf{B}'\boldsymbol{\beta}))^2+\sum_{j=1}^{d}\lambda_j\int(s_j''(x_j|_{\mathbf{B}_j'\boldsymbol{\beta}_j}))^2dx_j, \tag{10}$$

where y_i is the actual class of the i-th datapoint (which can have value of either 0 or 1) and $\hat{y}_i$ is the probability of i-th point belonging to class $y = 1$ (i.e. $P(y = 1|\mathbf{x_i})$) as predicted by the GAM classifier using Eq. 7. λ_j are the penalty parameters which are prespecified. In our case, we use $\lambda_j = 0.6$ for all features. The rule complexity of a GAM classifier can be tuned using λ_j, where higher values of λ_j imposes heavy penalty on non-linearities with more than second order. Additionally, the complexity can also be controlled by regulating the *degree* (q_i) and *number of basis-functions* K_i (Eq. 9). In our experimental setup, we conduct series of experiments using different combinations of (K_i, q_i) to model splines for each feature. Values of K and q are picked from the one listed in Table II.

TABLE II: Details regarding parametric study for GAMs.

# Basis Functions (K)	Degree (q)
2, 3, 5, 8, 13, 21	2, 3, 5

Total number of terms arising from the expression of rule $g(\mathbf{x})$ (Eq. 9) is $\sum_{j=1}^{d}(q_j + 1) \times K_j + K_j + 1$. However, due to second-order smoothening effect (Eq. 10), 2nd order non-linearities which are not contributing in minimizing the error $\sum_{i=1}^{N}(y_i - \hat{y}_i(\mathbf{B}'\boldsymbol{\beta}))^2$ will get removed from the rule and thus, the *effective degree of freedom* (EoDF) will be far less than the total length of the rule. Effective degrees of freedom versus accuracy plot for GAM classifiers obtained using various combinations of (K_i, q_i) on Cancer-10 dataset is shown in Figure 4. It is clear that a high training accuracy is achieved with a large EoDF, but makes an over-fitting and produces less testing accuracy. About 500 such experiments are performed and the best combinations of (K_i, q_i) are used to generate results (Table IV) for a given dataset. Note here that generating classifiers using GAM is computationally expensive for high-dimensional datasets and so, we do not run experiments on datasets involving 500 features.



(a) Training Accuracy

(b) Testing Accuracy

Fig. 4: Effective degree of freedom (EoDF) V/s Accuracy for Cancer-10 dataset. The best (K_i, q_i) parameter setting for this dataset is found to be $K^* = [8, 3, 8, 13, 8, 8, 13, 3, 8, 21]$ and $q^* = [2, 2, 5, 5, 2, 3, 3, 2, 2, 2]$.

Advantages:

- Effect of each feature on the output variable can be separately analyzed using partial dependence plots.
- A source code is available [19] for rapid prototyping.

Disadvantages:

- Hyperparameters defining the non-linear functions Eq. 8 needs to be properly identified.
- Slow to train as compared to other methods.
- Becomes computationally expensive to handle high dimensional datasets.

D. Genetic Programming (GP)

Genetic Programming has been extensively used to derive non-linear and interpretable classifiers [2], [4], [5], [9], [20]. A GP algorithm

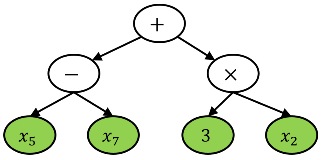


Fig. 5: A sample genetic program (GP) tree. The above GP translates this equation: $f(\mathbf{x}) = (x_5 - x_7) + 3x_2$.

evolves *programs* (or equations of classifier's decision boundary in our case) using genetic operators like crossover and mutation. Programs in GP are usually represented with tree architecture as shown in Figure 5. Internal nodes of this tree can involve mathematical operations, like $+, \times, -, \div, \log, \sin$. Allowable set of mathematical operations are pre-specified by the user. In our case, we use $\{+, \times, -, \div\}$ only. Terminal leaf nodes of a GP program either have one of the input feature x_i or a constant term c. It is to note here that a GP tree ($\mathbf{T}$) represents one non-linear equation and is fundamentally different from the decision tree which involves assembly of split-rule equations which are organized in a hierarchical format (Figure 1). The optimal structure of tree, operators used, features x_i involved and value of constants c are all unknown and are determined through an evolutionary algorithm. The evolution is conducted with an objective to minimize the cross-entropy loss. However, if unchecked, the size of GP trees grows as the evolution progress and the GP algorithm suffers from *bloating* [11]. To counter this effect of bloating and encourage evolution of simpler trees (trees with less number of nodes), a parsimony coefficient P_c is used to penalize the fitness of a GP tree ($\mathbf{T}$) as shown below:

$$\textit{Min:} \quad f_{GP}(\mathbf{T}) = C_{loss} + P_c \times T_{size}, \tag{11}$$

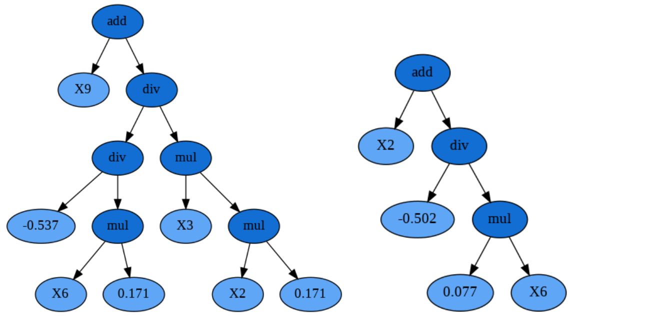
where,

$$C_{loss} = -\frac{1}{N}\sum_{i=1}^{N} y(\mathbf{x}_i)\log(\hat{y}(\mathbf{x}_i)) - (1-y(\mathbf{x}_i))\log(1-\hat{y}(\mathbf{x}_i)),$$
$$\hat{y}(\mathbf{x}) = \text{Sigmoid}(f(\mathbf{x})). \tag{12}$$

In Eq. 11, T_{size} represents size of the tree and is computed by counting total number of nodes in the tree. In Eq. 12, $f(\mathbf{x})$ is the value the GP tree outputs for a given feature vector $\mathbf{x}$ (see Figure 5).

It is important to choose a suitable parsimony coefficient P_c for a problem. Smaller value of P_c will encourage bloating and will evolve complex equations while the higher value of P_c will evolve simpler equations at an expense of reduced classification accuracy. In our case, we perform experiments using three values P_c: 0.01, 0.005 and 0.001, and conduct 50 runs on each dataset shown in Table III (discussed in Section III) after randomly splitting the dataset into 70% training and 30% testing for each run. Statistics regarding testing accuracy and *complexity* (measured as the total number of internal nodes) is reported in the table. It is clear from the table that while a small P_c produces a better accuracy, a large P_c produces smaller sized GPs. To demonstrate, we present two GP classifiers for $P_c = 0.005$ and 0.01 obtained for the breast cancer Wisconsin dataset (involving total 10 features) in Figure 6. Training (T_r) and testing (T_e) accuracy are better for $P_c = 0.005$.

Table III indicates that GP does not perform well on certain problems even in small-sized problems, such as DS1 and DS4. In a mathematical classifier search, there are two hierarchical aspects which must be learnt: (i) structure of the classifier, and (ii) coefficient of each term in the structure. GP attempts to learn both aspects in a single optimization task. We argue that while a "good" structure may have evolved at a generation, if its associated coefficients are not proper, the whole classifier will be judged as "bad". We attempt to alleviate this aspect in the next procedure by using a bilevel optimization framework.



(a) $P_c = 0.005$, $T_r = 96.44$, $T_e = 99.02$, Complexity = 6.
(b) $P_c = 0.01$, $T_r = 95.60$, $T_e = 98.05$, Complexity = 3.

Fig. 6: Classifiers for Cancer data: $P_c = 0.005$: $f(\mathbf{x}) = x_9 + \frac{-0.537}{(0.171x_6)(0.171x_3x_2)}$ and $P_c = 0.01$: $f(\mathbf{x}) = x_2 + \frac{-0.502}{(0.077x_6)}$.

Advantages:

- Non-linearity gets automatically determined during evolution.
- Open Source Code is available https://gplearn.readthedocs.io/en/stable/index.html.

Disadvantages:

- Correct set of operators needs to be specified to derive optimal interpretable classifier.
- Training is slow as compared to SVM and CART.
- Parsimony coefficient P_c severely impacts the performance of GP and so it needs to be tuned properly.

E. Nonlinear Decision Tree (NLDT) Approach

Recently, an evolutionary algorithm based non-linear decision tree classifier was proposed in [8]. The classifier is represented in the form of a *non-linear* decision tree as shown in Figure 7. Unlike in regular CART based decision tree where the split-functions are constrained to have axis-parallel

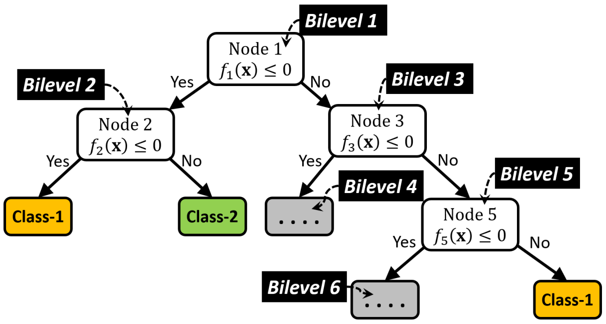


Fig. 7: NLDT Schematic.

TABLE III: GP Result for different values of parsimony coefficient P_c. For each dataset, the first row represents the testing accuracy and the second row represents complexity (number of internal nodes). $P_c = 0.001$ produces better results.

Pars. coeff.	**DS1**	**DS2**	**DS3**	**DS4**	**Truss**	**WeldedBeam**
$P_C = 0.01$	61.07 ± 9.91 $\mathbf{3.40 \pm 3.70}$	95.24 ± 0.00 $\mathbf{1.98 \pm 0.14}$	65.37 ± 11.57 $\mathbf{4.44 \pm 2.37}$	49.93 ± 1.43 $\mathbf{1.12 \pm 3.40}$	82.78 ± 11.28 $\mathbf{5.20 \pm 3.30}$	84.88 ± 13.08 $\mathbf{9.32 \pm 5.15}$
$P_C = 0.005$	77.3 ± 11.29 16.18 ± 9.99	95.24 ± 0.00 3.86 ± 1.23	86.27 ± 11.41 19.86 ± 11.45	50.37 ± 2.96 2.06 ± 3.88	90.03 ± 8.50 11.98 ± 7.12	92.35 ± 6.06 14.08 ± 5.35
$P_C = 0.001$	$\mathbf{91.70 \pm 6.91}$ 67.72 ± 26.72	$\mathbf{95.37 \pm 0.63}$ 15.14 ± 13.55	$\mathbf{96.50 \pm 3.3}$ 76.74 ± 33.36	$\mathbf{58.00 \pm 11.22}$ 18.76 ± 23.94	$\mathbf{97.36 \pm 3.81}$ 36.02 ± 16.99	$\mathbf{96.46 \pm 4.14}$ 35.90 ± 18.28

Pars. coeff.	**m-DS1**	**m-DS2**	**m-DS3**	**Cancer-10**	**Cancer-30**
$P_C = 0.01$	89.53 ± 3.27 $\mathbf{8.34 \pm 1.98}$	95.65 ± 0.70 $\mathbf{3.58 \pm 1.07}$	96.33 ± 4.68 $\mathbf{15.04 \pm 6.36}$	94.03 ± 4.59 $\mathbf{5.56 \pm 2.06}$	90.47 ± 4.54 $\mathbf{4.78 \pm 2.30}$
$P_C = 0.005$	93.37 ± 4.57 16.32 ± 9.55	95.65 ± 0.70 3.76 ± 1.22	98.4 ± 1.99 19.88 ± 9.94	95.04 ± 1.76 7.88 ± 3.07	90.96 ± 6.29 5.74 ± 1.84
$P_C = 0.001$	$\mathbf{98.83 \pm 1.88}$ 55.38 ± 22.39	$\mathbf{96.67 \pm 1.93}$ 14.08 ± 9.11	$\mathbf{99.27 \pm 1.22}$ 49.80 ± 21.69	$\mathbf{96.13 \pm 1.29}$ 15.80 ± 5.66	$\mathbf{92.40 \pm 4.98}$ 14.58 ± 7.14

structure (Figure 1), split-functions $f_i(\mathbf{x})$ in NLDT are nonlinear to the features and are represented as weighted sum of p power-laws as shown below:

$$f(\mathbf{x}) = \begin{cases} \displaystyle\sum_{i=1}^{p} w_i B_i + \theta_1, & \text{if } m = 0, \\ \left| \displaystyle\sum_{i=1}^{p} w_i B_i + \theta_1 \right| - \theta_2, & \text{if } m = 1, \end{cases} \tag{13}$$

where B_i are the power-laws ($B_i = \prod_{j=1}^{d} x_j^{b_{ij}}$), w_i are coefficients, θ_i are biases, and d is the number of features in the dataset. The exponents b_{ij} of the j-th feature in the i-th power-law can assume a value from a pre-specified discrete set E. In our case, we choose $E = -1, -2, \ldots, 3$. The number of power-laws p is set to 3 in all the experiments. At each conditional node in NLDT, the expression for split-rule $f(\mathbf{x})$ is derived by optimizing exponents b_{ij}, coefficients w_i, biases θ_i and the modulus-flag m using a dedicated bilevel algorithm as shown in Figure 7. The upper level of the bilevel algorithm operates in the discrete space of exponents b_{ij} (which are encoded using a matrix $\mathbf{B}$) and the modulus flag m while for each upper level solution S_U, the lower level algorithm searches for the optimal values of weights $\mathbf{w}$ and biases $\mathbf{\Theta}$. The upper level is modeled as a single-objective constrained optimization problem with an objective to minimize the complexity F_U of the split-rule $f(\mathbf{x})$ while ensuring that child nodes resulting from split have their net impurity F_L less than a user specified threshold value τ_I (set to 0.05 in our experiments). The bilevel optimization formulation to derive a split-rule $f(\mathbf{x})$ in NLDT can then be written as shown below:

$$\begin{aligned} \text{Min.} \quad & F_U(\mathbf{B}, m, \mathbf{w}^*, \mathbf{\Theta}^*), \\ \text{s.t.} \quad & (\mathbf{w}^*, \mathbf{\Theta}^*) \in \operatorname{argmin}\big\{F_L(\mathbf{w}, \mathbf{\Theta})|_{(\mathbf{B},m)} \big| F_L(\mathbf{w}, \mathbf{\Theta})|_{(\mathbf{B},m)} \\ & \quad \leq \tau_I, -1 \leq w_i \leq 1,\ \forall i,\ \mathbf{\Theta} \in [-1, 1]^{m+1}\big\}, \\ & m \in \{0, 1\},\ b_{ij} \in \{-3, -2, -1, 0, 1, 2, 3\}. \end{aligned} \tag{14}$$

The upper level objective F_U, which quantifies the complexity is computed by counting all non-zero exponents b_{ij} in the expression of $f(\mathbf{x})$ (Eq. 13). The lower level objective function F_L which quantifies the quality of split is obtained using the weighed sum of impurities of child nodes as shown in Eq. 2. Evolutionary algorithms for both upper and lower level are employed to conduct an efficient search on upper level variables ($\mathbf{B}$, m) and lower level variables ($\mathbf{w}$, $\mathbf{\Theta}$). Splits in NLDT are recursively derived until a certain termination criteria is met. The bilevel optimization serves as a very efficient search technique to derive simple split rules, an example of which is shown in Figure 8 for Wisconsin breast cancer dataset involving total 10 features. Besides being an interpretable classifier, it also reveals that only five (x_2-x_4, x_7 and x_{10}) of ten features are important in making the classification.

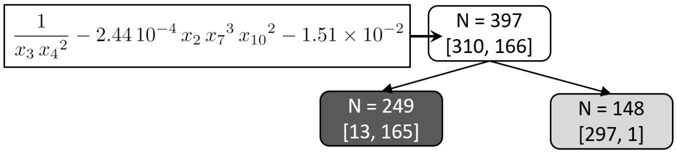

$\frac{1}{x_3 x_4{}^2} - 2.44\,10^{-4}\,x_2\,x_7{}^3\,x_{10}{}^2 - 1.51 \times 10^{-2}$
N = 397
[310, 166]
N = 249
[13, 165]
N = 148
[297, 1]

Fig. 8: NLDT with a single nonlinear rule obtained for Breast Cancer Wisconsin dataset (total 10 features) is shown. If the function value on the root node is less-than-equal-to zero, the data point is classified as Class 2 point with 165/178 or 92.7% accuracy and if it is positive, then the data point is classified as Class 1 point with 297/298 or 99.7% accuracy.

Advantages:

- Due to the use of nonlinear structure, the NLDT will have a fewer rules,
- The structure of the rules can be controlled easily, so interpretable rules can be obtained.
- Recent advancements in nonlinear optimization methods enable NLDTs to be evolved efficiently.

Disadvantages:

- Maximum depth, total number of power laws per rule, exponent set, impurity threshold τ_I and minimum number of data points to conduct split, need to be set.
- Training is slower as compared to CART and SVM.

Details regarding the bilevel optimization algorithm and parameter settings can be found from [8].

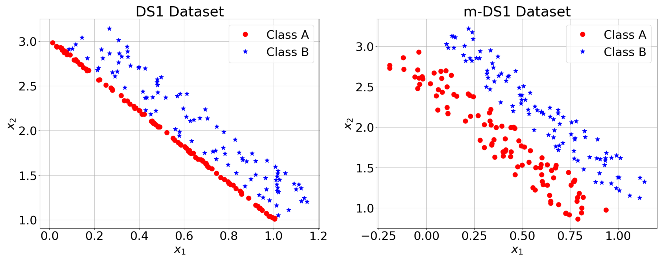
III. Datasets Considered

In our study, we conduct experiments on total 19 datasets to explore and investigate behaviour of various classification algorithms discussed above on varieties of features spaces and data distributions.

A. Customized Data: DS1-4 and modified DS1-3

Four synthetic two dimensional datasets DS1-DS4 and their variants m-DS1, m-DS2 and m-DS3 are generated using the procedure provied in [8] to investigate behavior of classification algorithms across following properties:

- **Data Distribution:** For DS1-DS4 datasets, degree of scatter in data varies across classes. For m-DS1, m-DS2 and m-DS3 the scattering of data for each class is more similar than that in original DS datasets. A visualisation of feature spaces for DS1 and m-DS1 dataset is provided in Figure 9a and 9b, respectively.
- **Geometry of Decision Boundary:** Here, the effect of the nature of the simplest possible decision boundary is considered. Decision boundary corresponding to DS1-DS2 and modified DS1-DS2 is linear, DS3 and m-DS3 have decision boundary involving nonlinearity of order 2 and DS4 have two disjoint linear decision boundaries.
- **Data Bias:** Here, effect of bias in class representation is considered. All datasets except DS2 and m-DS2 are balanced. For DS2 and m-DS2, minority class has 5 times less number of data points as the majority class.



(a) DS1 Dataset.

(b) m-DS1 Dataset.

Fig. 9: Original DS1 and its modified version.

B. Cancer Datasets

We use breast cancer Wisconsin data involving 10 features and Wisconsin Diagnostics dataset having 30 features.

C. Pareto versus Non-Pareto Classification

In multi-objective optimization, there are two types of solutions: (i) Pareto-optimal set and (ii) Dominated set. Users are interested in knowing what feature relationships (decision variables interactions) make a solution Pareto-optimal, thereby making the task a binary classification problem.

1) Test problems: We use modified versions of ZDT [23] and DTLZ [7] problems with two and three objectives, respectively to generate datasets involving 30 and 500 features (details in [8]). These two problem sizes also allow us to perform a scale-up study of the classification methods.

2) Real-world Problems: Next, we consider two real-world problems – welded beam and truss design problems [6].

IV. Results and Discussions

Table IV presents the testing accuracy and complexity of five classification methods on 19 problems. For each method, a parametric study is performed on each problem and the setting which obtained the best testing accuracy is used to generate the final results. Statistics of 50 runs (with random data split of 70% training and 30% testing in each) on each dataset for two performance metrics is presented in Table IV.

For CART, the complexity metric is defined as total number of nodes, for SVM, it is defined as total number of feature vectors, for GAM, it is defined as the effective degrees of freedom (EoDF); for GP, it is defined as the total number of internal nodes; and for NLDT, it is defined as total number of variables present in the entire tree. It is clear that a method with high testing accuracy and low complexity is better.

The table clearly indicates that NLDT performs well in terms of both metrics. Also, the performance of NLDT scales well with an increase in feature size. CART produces a good compromise on accuracy and complexity, but performs worse than NLDT on both metrics. While SVM achieves a high accuracy, in general, the complexity of its classifiers is large, thereby making them not easy to interpret for any explainability purposes. The performance of GP is poor for achieving a high accuracy. GAM is clearly not suitable for problems with a large number of features and cannot be run due to impractical computational time requirement for some problems (marked with a dash). GP cannot match both accuracy and complexity obtained by NLDT. In most problems, NLDT classifiers require fewer conditional rules (albeit with restricted nonlinearities) and still achieve near 100% correct testing accuracy.

V. Conclusions

In this paper, we have presented three popular binary classification methods – CART, SVM and GAM. We have also included a genetic programming approach and a recently proposed nonlinear decision tree (NLDT) approach for a comparison with three existing methods on 19 different problems involving two to 500 features. The advantages and disadvantages of each method are described by highlighting one or more problem parameters which control the potential trade-off between complexity of the obtained classifier and its testing accuracy. The extensive comparative results have indicated that the NLDT approach makes an excellent compromise between the testing accuracy and complexity of the classifier. While the former is always important for a classifier, the latter allows an user to look for an explanation involving features and their interactions for the classifier's working principles.

The study raises a number of interesting future studies: (i) extension to multi-class classification problems, (ii) extension to regression problems, (iii) use of the bilevel approach similar to that used in NLDT search with GP to improve GP's performance, (iv) extension to nonlinear forest (NLF) search

TABLE IV: Summary of results obtained using various methods. For each dataset, the first row indicates testing accuracy and the second row indicates complexity. Italicized entries are statistically insignificant (according to 95% confidence in Wilcoxon rank-sum test) compared to the best entry in the same row.

Sr.	Problem	NLDT	CART	SVM	GAM	GP
1	DS1	99.55 ± 1.08 $\mathbf{2.3 \pm 0.6}$	90.32 ± 4.06 14.5 ± 1.7	99.87 ± 0.45 8.16 ± 0.88	$\mathbf{100.0 \pm 0.00}$ 2.89 ± 0.00	91.70 ± 6.91 67.72 ± 26.72
2	DS2	99.44 ± 0.87 $\mathbf{2.3 \pm 0.7}$	95.43 ± 1.50 11.0 ± 1.4	99.33 ± 1.10 7.64 ± 0.87	$\mathbf{100.0 \pm 0.00}$ 2.89 ± 0.00	95.37 ± 0.63 15.14 ± 13.55
3	DS3	$\mathbf{99.77 \pm 0.67}$ $\mathbf{2.2 \pm 0.5}$	95.00 ± 2.35 11.5 ± 1.3	$\mathit{99.63 \pm 0.69}$ 10.22 ± 1.42	$\mathit{99.47 \pm 1.03}$ 4.98 ± 0.14	96.50 ± 3.30 76.74 ± 33.36
4	DS4	$\mathbf{98.88 \pm 1.65}$ $\mathbf{3.1 \pm 1.4}$	88.68 ± 3.60 31.3 ± 4.2	93.97 ± 2.35 43.70 ± 2.69	48.63 ± 6.50 3.80 ± 0.99	59.63 ± 10.81 24.70 ± 26.40
5	m-DS1	99.10 ± 1.54 $\mathbf{2.00 \pm 0.00}$	89.73 ± 4.53 7.90 ± 1.22	99.90 ± 0.40 7.50 ± 0.75	$\mathbf{100.0 \pm 0.00}$ 2.90 ± 0.00	98.83 ± 1.88 55.38 ± 22.39
6	m-DS2	99.46 ± 1.08 $\mathbf{2.10 \pm 0.30}$	96.25 ± 1.92 5.96 ± 0.81	99.94 ± 0.44 5.44 ± 0.67	$\mathbf{99.94 \pm 0.31}$ 2.90 ± 0.00	96.67 ± 1.93 14.08 ± 9.11
7	m-DS3	99.20 ± 1.30 $\mathbf{2.02 \pm 0.14}$	92.87 ± 4.35 5.78 ± 1.11	$\mathbf{100.00 \pm 0.00}$ 8.82 ± 0.89	99.17 ± 1.48 3.24 ± 0.22	99.27 ± 1.22 49.8 ± 21.69
8	Cancer-10	$\mathbf{96.50 \pm 1.16}$ $\mathbf{6.4 \pm 1.7}$	94.34 ± 1.92 11.6 ± 2.4	95.07 ± 1.23 51.26 ± 5.02	95.32 ± 1.49 22.14 ± 10.36	96.13 ± 1.29 15.80 ± 5.66
9	Cancer-30	$\mathbf{96.20 \pm 1.49}$ $\mathbf{9.2 \pm 4.1}$	92.11 ± 2.07 10.8 ± 2.1	95.24 ± 1.29 58.88 ± 4.46	93.74 ± 5.83 32.47 ± 12.41	92.40 ± 4.98 14.58 ± 7.14
10	Welded Beam	98.58 ± 1.13 $\mathbf{3.9 \pm 1.0}$	97.72 ± 1.04 8.42 ± 1.42	$\mathbf{99.58 \pm 0.45}$ 7.86 ± 1.27	$\mathit{99.53 \pm 0.48}$ 11.06 ± 0.81	96.46 ± 4.14 35.90 ± 18.28
11	Truss	$\mathbf{99.54 \pm 0.75}$ $\mathbf{3.30 \pm 0.90}$	98.33 ± 1.10 11.06 ± 3.15	88.21 ± 1.62 174.28 ± 8.49	96.18 ± 1.20 19.19 ± 1.06	97.36 ± 3.81 36.02 ± 16.99
12	m-ZDT1-30	98.97 ± 0.57 $\mathbf{7.60 \pm 3.50}$	97.77 ± 0.58 30.26 ± 4.65	$\mathbf{99.39 \pm 0.35}$ 82.08 ± 4.19	85.31 ± 1.35 220.20 ± 11.73	93.58 ± 10.21 45.34 ± 26.09
13	m-ZDT1-500	98.93 ± 0.60 $\mathbf{9.34 \pm 4.15}$	95.96 ± 0.80 21.02 ± 1.55	$\mathbf{100.00 \pm 0.00}$ 140.58 ± 4.25	— —	83.21 ± 18.42 52.14 ± 24.47
14	m-ZDT2-30	98.96 ± 0.57 $\mathbf{8.10 \pm 3.35}$	97.88 ± 0.70 28.22 ± 2.35	$\mathbf{99.51 \pm 0.33}$ 80.98 ± 3.50	84.97 ± 1.18 233.69 ± 6.56	91.57 ± 11.92 48.44 ± 23.69
15	m-ZDT2-500	98.87 ± 0.72 $\mathbf{8.84 \pm 3.95}$	95.96 ± 0.80 21.02 ± 1.55	$\mathbf{100.00 \pm 0.00}$ 140.56 ± 4.00	— —	85.06 ± 16.82 50.64 ± 21.24
16	m-DTLZ1-30	$\mathbf{98.77 \pm 0.87}$ $\mathbf{11.98 \pm 5.85}$	78.52 ± 7.94 128.40 ± 22.39	94.22 ± 0.95 615.54 ± 9.24	55.96 ± 3.19 33.89 ± 0.01	81.59 ± 17.32 16.08 ± 21.89
17	m-DTLZ1-500	$\mathbf{93.76 \pm 4.24}$ 22.72 ± 7.50	78.31 ± 7.21 126.94 ± 20.07	64.32 ± 1.76 1236.82 ± 13.26	— —	80.49 ± 11.54 $\mathbf{8.66 \pm 16.19}$
18	m-DTLZ2-30	$\mathbf{97.22 \pm 2.25}$ 17.48 ± 6.75	69.83 ± 6.16 156.00 ± 16.09	94.25 ± 1.04 615.44 ± 10.67	54.52 ± 2.96 35.84 ± 0.02	79.81 ± 19.46 $\mathbf{12.32 \pm 14.79}$
19	m-DTLZ2-500	$\mathbf{95.32 \pm 4.45}$ 22.02 ± 10.62	76.68 ± 5.44 133.22 ± 15.95	64.22 ± 1.48 1245.92 ± 11.45	— —	78.46 ± 14.08 $\mathbf{8.08 \pm 15.21}$

involving multiple NLDTs for generating more compact, but slightly more complex and more accurate rules.

References

[1] C. M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[2] M. C. J. Bot and W. B Langdon. Application of genetic programming to induction of linear classification trees. In *European Conference on Genetic Programming*, pages 247–258. Springer, 2000.

[3] L. Breiman. *Classification and regression trees*. Routledge, 2017.

[4] A. Cano, A. Zafra, and S. Ventura. An interpretable classification rule mining algorithm. *Information Sciences*, 240:1–20, 2013.

[5] I. De Falco, A. D. Cioppa, and E. Tarantino. Discovering interesting classification rules with genetic programming. *Applied Soft Computing*, 1(4):257–269, 2002.

[6] K. Deb. *Multi-objective optimization using evolutionary algorithms*. Wiley, Chichester, UK, 2001.

[7] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable test problems for evolutionary multi-objective optimization. In A. Abraham, L. Jain, and R. Goldberg, editors, *Evolutionary Multiobjective Optimization*, pages 105–145. London: Springer-Verlag, 2005.

[8] Y. Dhebar and K. Deb. Interpretable rule discovery through bilevel optimization of split-rules of nonlinear decision trees for classification problems, 2020.

[9] J. Eggermont, J. N. Kok, and W. A. Kosters. Genetic programming for data classification: Partitioning the search space. In *Proceedings of the 2004 ACM symposium on Applied computing*, pages 1001–1005, 2004.

[10] T. J. Hastie and R. J. Tibshirani. *Generalized additive models*, volume 43. CRC press, 1990.

[11] H. Iba. Bagging, boosting, and bloating in genetic programming. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation*, pages 1053–1060, 1999.

[12] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, Q. Ma, W.and Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Adv. in neural information processing systems*, pages 3146–3154, 2017.

[13] M. Kearns and Y. Mansour. On the boosting ability of top–down decision tree learning algorithms. *Journal of Computer and System Sciences*, 58(1):109–128, 1999.

[14] K. Larsen. Gam: the predictive modeling silver bullet. *Multithreaded Stitch Fix*, 30:1–27, 2015.

[15] J. A. Nelder and R. W. Wedderburn. Generalized linear models. *Journal of the Royal Statistical Society: Series A*, 135(3):370–384, 1972.

[16] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[17] J. Quinlan. *C4.5: Programs for machine learning*. Elsevier, 2014.

[18] J. R. Quinlan. Induction of decision trees. *Machine learning*, 1(1):81–106, 1986.

[19] D. Servén and C. Brummitt. pygam: generalized additive models in python. *Zenodo. DOI*, 10, 2018.

[20] K. C. Tan, A. Tay, T. H. Lee, and C. M. Heng. Mining multiple compre-

hensible classification rules using genetic programming. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02*, volume 2, pages 1302–1307. IEEE, 2002.

[21] S. N. Wood. *Generalized additive models: An introduction with R*. CRC press, 2017.

[22] D. Zhang, X. Zhou, S. C. H. Leung, and J. Zheng. Vertical bagging decision trees model for credit scoring. *Expert Systems with Applications*, 37(12):7838–7843, 2010.

[23] E. Zitzler, K. Deb, and L. Thiele. Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary Computation Journal*, 8(2):125–148, 2000.