Interpretable Categorization of Heterogeneous Time Series Data

Ritchie Lee
Carnegie Mellon University Silicon Valley
NASA Ames Research Park, MS23-11
Moffett Field, California 94089
ritchie.lee@sv.cmu.edu

Mykel J. Kochenderfer
Stanford University
496 Lomita Mall, Durand 250
Stanford, California 94305
mykel@stanford.edu

Ole J. Mengshoel
Carnegie Mellon University Silicon Valley
NASA Ames Research Park, MS23-11
Moffett Field, California 94089
ole.mengshoel@sv.cmu.edu

Joshua Silbermann
Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Road
Laurel, Maryland 20723
joshua.silbermann@jhuapl.edu

ABSTRACT

We analyze data from simulated aircraft encounters to validate and inform the development of a prototype aircraft collision avoidance system. The high-dimensional and heterogeneous time series dataset is analyzed to discover properties of near mid-air collisions (NMACs) and categorize the NMAC encounters. Domain experts use these properties to better organize and understand NMAC occurrences. Existing solutions either are not capable of handling high-dimensional and heterogeneous time series datasets or do not provide explanations that are interpretable by a domain expert. The latter is critical to the acceptance and deployment of safety-critical systems. To address this gap, we propose grammar-based decision trees along with a learning algorithm. Our approach extends decision trees with a grammar framework for classifying heterogeneous time series data. A context-free grammar is used to derive decision expressions that are interpretable, application-specific, and support heterogeneous data types. In addition to classification, we show how grammar-based decision trees can also be used for categorization, which is a combination of clustering and generating interpretable explanations for each cluster. We apply grammar-based decision trees to a simulated aircraft encounter dataset and evaluate the performance of four variants of our learning algorithm. The best algorithm is used to analyze and categorize near mid-air collisions in the aircraft encounter dataset. We describe each discovered category in detail and discuss its relevance to aircraft collision avoidance.

CCS CONCEPTS

•Information systems → Clustering; •Mathematics of computing → Time series analysis; •Computing methodologies → Supervised learning; Classification and regression trees; *Batch learning;*

KEYWORDS

Interpretable, heterogeneous time series, classification, categorization, clustering, grammar

ACM Reference format:
Ritchie Lee, Mykel J. Kochenderfer, Ole J. Mengshoel, and Joshua Silbermann. 2017. Interpretable Categorization of Heterogeneous Time Series Data. In *Proceedings of KDD Conference, Halifax, Nova Scotia Canada, August 2017 (KDD'17),* 9 pages.
DOI: 10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Airborne collision avoidance systems are mandated worldwide on all large transport and cargo aircraft to help prevent mid-air collisions. Their operation have played a crucial role in the high level of safety in the national airspace [14]. To address the growing needs of the national airspace, the Federal Aviation Administration (FAA) has decided to develop a new aircraft collision avoidance system. The next-generation Airborne Collision Avoidance System (ACAS X) is currently being developed and tested and promises a number of potential improvements over current systems including a reduction in collision risk while simultaneously reducing the number of unnecessary alerts [11].

One of the primary safety metrics of airborne collision avoidance systems is the likelihood of near mid-air collision (NMAC), defined as two aircraft coming closer than 500 feet horizontally and 100 feet vertically. Efficient algorithms have been developed to generate large datasets of NMAC and non-NMAC instances in simulation [17]. However, while it is straightforward to observe that an NMAC has occurred, discovering and categorizing relevant properties of NMACs is much more challenging. Understanding how NMAC events occur is important for both validation and informing development of ACAS X.

We face a combination of challenges in this problem. One challenge is the complexity of the dataset since simulated aircraft encounters produce high-dimensional and heterogeneous time series data. A second challenge is the need for interpretability. That is, humans must be able to understand and reason about the information captured by the model. Interpretability is essential for domain experts to validate the model's output and build trust in the system. It is critical to the acceptance and deployment of safety-critical systems like ACAS X. Existing solutions, however, either are not

KDD'17, Halifax, Nova Scotia Canada
2017. 978-x-xxxx-xxxx-x/YY/MM...$15.00
DOI: 10.1145/nnnnnnn.nnnnnnn

capable of handling such datasets or do not provide explanations that are intuitive to a domain expert. Because no suitable solutions currently exist, encounter data is routinely categorized manually by a domain expert. However, as datasets become large, the time and cost of this approach become prohibitive.

In this paper, we present grammar-based decision tree (GBDT), a framework for interpretable classification of high-dimensional and heterogeneous time series data. GBDT combines decision trees and a grammar framework for classification. In contrast to traditional decision trees that use simple rules for partitioning, GBDT partitions the data using expressions derived from a user-supplied context-free grammar (CFG). This approach offers the flexibility to define expressions that support a wide range of data types and are interpretable and tailored to the user. In addition to classification, GBDTs can also be used for categorization, which is the combined task of clustering data into similar groups and providing intuitive labels for them. Categorizations are produced naturally by the decision tree at no additional cost.

We present an induction algorithm for GBDTs that employs grammar-based expression search (GBES) as a subroutine to search for good partitioning expressions. Four existing GBES algorithms are considered resulting in four variations of the GBDT induction algorithm. We evaluate the performance of these algorithms on an aircraft encounter dataset and discuss the properties of each category discovered by the algorithm.

The remainder of the paper is organized as follows. Section 2 reviews related literature on interpretable models. Section 3 reviews notation and terminology, CFGs as well as four GBES algorithms. Section 4 presents GBDT and shows how the model can be used for both classification and categorization. Section 5 presents a general induction algorithm for GBDTs. Finally, Section 6 evaluates four variants of the GBDT induction algorithm on a simulated aircraft encounter dataset and presents results of applying GBDT to study near mid-air collisions.

2 RELATED WORK

A variety of interpretable models for static data have been proposed in the literature. However, we are not aware of any work that simultaneously addresses high-dimensional and heterogeneous time series data and provides a high-degree of interpretability. Regression models [29], generalized additive models [9][19], and Bayesian case models [10] have been recently proposed as models with interpretability. These models improve interpretability by stating decision boundaries in terms of basis functions or representative instances (prototypes). Bayesian networks have also been used for prediction and data understanding [1].

Rule-based models, such as decision trees [3][27], decision lists [28], and decision sets [15], are easy to understand because decision boundaries are stated in terms of input attributes and simple logical operators. Decision trees partition the data using a tree structure and decision lists use an *if-then-else* branching structure. Decision sets use independent decision rules to reduce coupling between rules. Efficient algorithms have been proposed for inducing these models by first using associative rule mining (ARM) to mine a set of interesting rules from the data, then optimizing over combinations of these rules to induce classifiers [2][18][15].

Genetic programming (GP) has been studied extensively for classification problems [6]. GP is particularly well-suited to evolve tree structures [13][32]. A number of studies have used GP to evolve interpretable models for analyzing medical datasets [24][7]. Grammar-guided genetic programming (GGGP) uses a grammar to guide the evolution of genetic programs [34][21]. A variety of classification structures, including decision trees, have been evolved using GGGP [33].

Our work is most similar to the work of Marmelstein et al. [20], where a GP classifier is used as a subroutine to incrementally build a decision tree. However, their work did not use a CFG to guide the search and their work did not consider heterogeneous time series data. We also differ in the choice of fitness function and the consideration of non-evolutionary search techniques.

Inductive Logic Programming (ILP) uses logic programming to find a set of logical implications (Horn clauses) that best explains the data given a set of known facts [22][23]. ILP has been previously used for interpretable classification of static data. Shapelets [35] and subsequences based on the symbolic aggregate approximation (SAX) discretization [30] have been proposed for single-dimensional time series classification. The approach searches for simple patterns that are most correlated with the class label. Interpretability comes from identifying a prototype of the recurring subsequence pattern.

3 PRELIMINARIES

3.1 Notation and Terminology

A multi-dimensional time series dataset D consists of m records, where each record is a two-dimensional matrix of n attributes by T time steps. A trace of an attribute x_i is denoted $\vec{x}_i$ and is a vector of length T that represents the time series of that attribute. A label is associated with each data record. Logical and comparison operators are given broadcast semantics where appropriate. For example, the comparison operator in $\vec{x}_i < c$ compares each element of $\vec{x}_i$ to c and returns a vector of the same size as $\vec{x}_i$. Similarly, the logical operator in $\vec{x}_i \wedge \vec{x}_j$ operates elementwise. The temporal operators F and G are *eventually* and *globally*, respectively. *Eventually* returns true if any value in the input vector is true. *Globally* returns true if all values in the input vector are true.

3.2 Context-Free Grammar

A context-free grammar (CFG) defines a set of rules that govern how to form expressions in a formal language, such as linear temporal logic (LTL) [8]. The grammar defines the syntax of the language as well as provides a direct means to generate valid expressions.

A CFG G is defined by a 4-tuple $(\mathcal{N}, \mathcal{T}, \mathcal{P}, \mathcal{S})$, where

- $\mathcal{N}$ is a set of *nonterminal* symbols, which are symbols that can be replaced by other symbols,
- $\mathcal{T}$ is a set of *terminal* symbols, which are symbols that will appear in the final expression generated by the grammar,
- $\mathcal{P}$ is a set of *productions*, which are rules for replacing a nonterminal symbol with other nonterminal or terminal symbols, and
- $\mathcal{S}$ is the *start* symbol, which is a special nonterminal symbol that serves as the starting expression of a derivation.

To generate an expression from the grammar, we begin with the start symbol, then repeatedly apply production rules to rewrite the expression until no more nonterminals remain. When applying a rule, a nonterminal in the expression is replaced with a substitution defined in a production rule. The rules can be applied any number of times and in any order. The expression is complete when the expression consists of only terminal symbols. The derivation is commonly represented as a tree structure called a derivation tree.

Formally, grammars define only the syntax and not the semantics of expressions. However, for convenience, we shall assume that the semantics of the language are defined and use the term "grammar" loosely to refer to both the grammar and its associated semantics.

3.3 Grammar-Based Expression Search

Grammar-based expression search (GBES) is the problem of finding expressions from a grammar that minimize a given fitness function [21]. The formulation is extremely general due to the flexibility and expressiveness of grammars and the arbitrary choice of fitness function. Owing to this generality, the GBES approach has been applied to a wide variety of applications, including image and signal processing, modeling of medical and economic data, and industrial process control [26]. A number of GBES algorithms have been proposed in the literature. We review four of these algorithms in the following sections.

3.3.1 Monte Carlo. Monte Carlo is a simple algorithm that generates expressions by repeatedly selecting nonterminals in the expression and randomly choosing substitutions to apply. Substitutions are chosen uniformly from all available possibilities. When no nonterminals remain, the fitness of the generated expression is evaluated and the expression with the best fitness is reported.

Since Monte Carlo search is undirected, identical expressions may be evaluated many times, which can be expensive. Furthermore, due to the nature of the sampling, shallower paths will get sampled more frequently than deeper paths. Deep paths may have a vanishingly small probability of being reached [26].

3.3.2 Monte Carlo Tree Search. Monte Carlo tree search (MCTS) is a heuristic search algorithm that is used to optimize certain sequential decision-making problems [12][4]. MCTS is based on reinforcement learning, the idea of improving behavior through experience and interaction with an environment [31].

Expression search is formulated as a sequential decision-making problem by transforming the decisions in the derivation tree so that they occur in sequential order, for example, by assuming depth-first traversal order [5]. The "state" of the system is the expression derived so far and the "action" is the selection of which substitution to apply to the selected nonterminal. MCTS is then applied to optimize the resulting sequential decision-making problem [5].

3.3.3 Grammatical Evolution. Grammatical evolution (GE) [25] is a GGGP algorithm that is based on a sequential representation of the derivation tree. Specifically, GE defines a transformation from a variable-length binary string to a sequence of rule selections in a CFG. Then it uses a standard genetic algorithm (GA) to search over binary strings [26]. Our implementation uses one-point crossover and uniform mutation [26].

3.3.4 Genetic Programming. Genetic programming (GP) is an evolutionary algorithm for optimizing trees [34]. Genetic operators are defined specifically for trees and thus do not require any transformations of the derivation tree. Our implementation uses a crossover operator that exchanges compatible subtrees between two individuals and a mutation operator that replaces entire subtrees with randomly-generated ones. We also use tournament selection for selecting individuals.

4 GRAMMAR-BASED DECISION TREES

We present grammar-based decision tree (GBDT) as a framework that extends decision trees with a grammar framework for the interpretable classification of high-dimensional and heterogeneous time series data. Traditional decision trees partition the input space using simple logical rules, such as $(x_1 < 2)$ [3][27]. However, these rules have limited expressiveness and cannot be used to express more complex logical relationships, such as those between heterogeneous attributes or across time. To address this problem, GBDT allows arbitrary logical expressions as decision rules. Any logical expression can be used so long as the evaluation of it produces a Boolean result. The domain of the logical expressions is constrained using a user-supplied CFG, where the user can introduce domain knowledge and application-specific tailoring. The decision expressions are organized as nodes in a tree where partitions are performed in a top-down fashion as in a traditional decision tree.

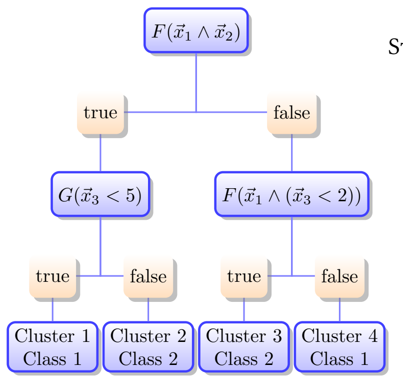
Class label prediction works as in a traditional decision tree. To predict a label for a given record, we recursively evaluate each decision expression on the record starting at the root of the tree and following the corresponding branch conditions down the tree. When a leaf node is reached, the most frequent label seen in the training data at that leaf node is returned. In this manner, predictions can be made for both previously seen and unseen data.

An added benefit of using a decision tree structure is the ability to extract a categorization from the model at no additional cost. Categorization can be helpful in building intuition and explaining data. A GBDT can be used to categorize data by considering each leaf node of the tree to be a separate category. To describe the category, a global expression for a node can be derived by forming the conjunction of all branch expressions along the path that connects the root and the node of interest. The members of the cluster are the records where the cluster's global expression holds. Since partitions are mutually exclusive, the clusters do not overlap.

Figure 1 shows an example of a GBDT with accompanying CFG expressed in Backus-Naur Form (BNF). The CFG in this example describes a simple temporal logic. It assumes that the data has four attributes $\vec{x}_1, \vec{x}_2, \vec{x}_3, \vec{x}_4$, where the attributes $\vec{x}_1$ and $\vec{x}_2$ are Boolean time-series vectors and $\vec{x}_3$ and $\vec{x}_4$ are time-series vectors of real numbers. The grammar contains two types of rules. *Expression rules*, such as $\mathrm{Ev} ::= F(\mathrm{VB})$, contain partially-formed expressions that contain terminal and non-terminal symbols. Non-terminal symbols are substituted further using the appropriate production rules. *Or Rules*, such as $B ::= \mathrm{Ev} \mid \mathrm{Gl}$, contain the symbol |, which is used to delineate the different possible substitutions.

Each non-leaf node of the decision tree contains a logical expression that governs which child branch is followed. Leaf nodes show the predicted class label. For example, a data record where

$F(\vec{x}_1 \wedge \vec{x}_2)$ is true and $G(\vec{x}_3 < 5)$ is false would be predicted to have class label 2. The example also shows the unique cluster numbers for each leaf node. For example, the righmost leaf node in Figure 1a is labeled as cluster 4. The global expression for cluster 4 is $\neg F(\vec{x}_1 \wedge \vec{x}_2) \wedge \neg F(\vec{x}_1 \wedge (\vec{x}_3 < 2))$ and the members of cluster 4 are those records in the data where that global expression holds.



(a) Tree

$$
\begin{aligned}
\textsc{Start} &::= \mathrm{B} \\
\mathrm{B} &::= \textsc{Ev} \mid \textsc{Gl} \\
\textsc{Ev} &::= F(\mathrm{VB}) \\
\textsc{Gl} &::= G(\mathrm{VB}) \\
\mathrm{VB} &::= \vec{x}_1 \mid \vec{x}_2 \mid \textsc{And} \mid \textsc{Lt} \\
\textsc{And} &::= \mathrm{VB} \wedge \mathrm{VB} \\
\textsc{Lt} &::= \mathrm{VR} < C \\
\mathrm{VR} &::= \vec{x}_3 \mid \vec{x}_4 \\
\mathrm{C} &::= 1 \mid 2 \mid 3 \mid 4 \mid 5
\end{aligned}
$$

(b) Grammar

Figure 1: Grammar-Based Decision Tree Example

4.1 Grammars for Heterogeneous Time Series

Grammar design is crucial to the effectiveness of GBDT. The ideal grammar should produce expressions that are interpretable, tailored to the application, and support the attribute types present in the dataset. This section aims to offer design suggestions to the user to achieve these goals. The exact details of the grammar will depend on the specific needs of the user.

In the GBDT framework, expressions are evaluated on a data record and produce a Boolean output. The symbols of the expression can refer to fields of the record, constants, or functions. We adopt a subset of linear temporal logic (LTL), a formal logic with temporal operators often used in the analysis of time-series data [8]. We have found grammars similar to the one presented in Figure 1 to be particularly effective for heterogeneous time series data. The grammar produces expressions that integrate Boolean, categorical, and real-valued data types nicely and the expressions use simple well-known operators, which are very intuitive.

When designing a grammar, it is important to manage data types. For example, the grammar in Figure 1 is organized by the data type of its non-terminals. The production rule B selects amongst functions that return a Boolean, whereas the production rule VB selects amongst functions and expressions that return a vector of Booleans. Similarly, the non-terminal VR represents a vector of real values while C represents a constant. Organization of types is not only important to provide functions with valid inputs but also key to meaningfully nest and combine heterogeneous types.

To keep the example simple, Figure 1 included only a small number of operators. The grammar can be readily extended to include a much wider set of operators including disjunct $\vee$, greater than $>$, equals $=$, where the equality operator is important for datasets with categorical attributes, and even arbitrary user-defined functions.

4.2 Natural Language Descriptions

Logical expressions can sometimes be dense and hard to parse. In many cases, we can improve interpretability by additionally providing natural language descriptions of the expressions to the user. One method to automatically translate expressions into English sentences is to map expression rules and terminal symbols in the CFG to corresponding sentence fragments and then use the structure of the expression's derivation tree to assemble the fragments. Figure 2 shows an example of a mapping from expressions and symbols to sentence fragments that could be used with the grammar in Figure 1. As an example, we have given English descriptions to each data attribute as well.

$$
\begin{aligned}
F(\mathrm{VB}) &:= \text{“at some point, VB”} \\
G(\mathrm{VB}) &:= \text{“for all time, VB”} \\
\mathrm{VB} \wedge \mathrm{VB} &:= \text{“VB and VB"} \\
\mathrm{VR} < C &:= \text{“VR is less than } C\text{”} \\
\vec{x}_1 &:= \text{“advisory is active”} \\
\vec{x}_2 &:= \text{“pilot is responding”} \\
\vec{x}_3 &:= \text{“vertical rate”} \\
\vec{x}_4 &:= \text{“airspeed”}
\end{aligned}
$$

Figure 2: Natural Language Map Example

Applying this mapping, the decision rules in Figure 1 can be transformed into the following natural language descriptions: $F(\vec{x}_1 \wedge \vec{x}_2)$ is translated to "at some point, [advisory is active] and [pilot is responding]"; $G(\vec{x}_3 < 5)$ is translated to "for all time, [vertical rate] is less than 5"; and $F(\vec{x}_1 \wedge (\vec{x}_3 < 2))$ is translated to "at some point, [advisory is active] and [[vertical rate] is less than 2]". We include square brackets in the sentence to help the reader disambiguate nested sentence components.

5 INDUCTION OF GBDTS

Induction of a GBDT follows that of a traditional decision tree, except that GBES is used as a subroutine to find the best partition expression. The induction algorithm begins with a single (root) node containing all data records. GBES is then used to search a CFG for the partitioning expression that yields the best fitness. The expression is evaluated on each record and the data is partitioned into two child nodes according to the results of the evaluation. The process is applied recursively to each child until all data records at the node are either correctly classified or a maximum tree depth is reached. The mode of the training labels is used for class label prediction at a leaf node. The GBDT induction algorithm is shown in Algorithm 1.

GBDT is the main entry point to the induction algorithm. It returns a Tree object containing the root node of the induced decision tree. Split attempts to partition the data into two parts. It first tests whether the terminal conditions are met and if so returns a LeafNode object that predicts the mode of the labels. The partitioning terminates if the maximum depth has been reached or if all class labels are the same, which is tested by the IsHomogeneous function. ExpressionSearch uses GBES to search for the expression

Algorithm 1 Grammar-Based Decision Tree Induction

```
1:  function GBDT($G, F, D, L, d$)
2:      $R \leftarrow$ Split($G, F, D, L, d$)
3:      return Tree($R$)
4:  function Split($G, F, D, L, d$)
5:      if IsHomogeneous($L$) or $d = 0$ then
6:          return LeafNode(Mode(L))
7:      $E \leftarrow$ ExpressionSearch($G, F, D, L$)
8:      $(D^+, D^-) \leftarrow$ SplitData($D, E$)
9:      $(L^+, L^-) \leftarrow$ TruthLabels($D^+, D^-$)
10:     $child^+ \leftarrow$ Split($G, F, D^+, L^+, d-1$)
11:     $child^- \leftarrow$ Split($G, F, D^-, L^-, d-1$)
12:     return Node($E, child^+, child^-$)
```

that induces the best split. SplitData evaluates the expression on each data record and partitions the data into two parts according to whether the expression holds. TruthLabels performs a lookup for the class labels. Then, Split is called recursively on each part. Split returns a Node object containing the decision expression and the children of the node.

5.1 Fitness Function

We evaluate the desirability, or *fitness*, of an expression according to two competing objectives. On one hand, we want expressions that split the data so that the resulting clusters have the same ground truth class labels. Splits that induce high homogeneity tend to produce shallower trees and thus shorter global expressions at leaf nodes. They also produce classifiers with better predictive accuracy when the maximum tree depth is limited. To quantify homogeneity, we use the Gini impurity metric following the classification and regression tree (CART) framework [3]. On the other hand, we want to encourage interpretability by minimizing the length and complexity of the expression. Shorter and simpler expressions are easier to interpret and may better describe reality according to the Occam's razor principle. We use the number of nodes in the derivation tree as a proxy for the complexity of an expression. The two objectives are combined linearly into a single (minimizing) fitness function given by

$$F(E) = w_1 I_G(E) + w_2 N_E$$

where F is the fitness function, w_1 and w_2 are weights, and N_E is the number of nodes in the derivation tree of E. The total Gini impurity, $I_G(E)$, is the sum of the Gini impurity of each group that results from splitting the data using expression E. It is given by

$$I_G(E) = \sum_{L \in \{L^+, L^-\}} \sum_{b \in B} f_L^b (1 - f_L^b)$$

where f_L^b is the fraction of elements in L that are equal to b, L^+ are the labels of the records where E evaluates to true, L^- are the labels of the records where E evaluates to false, and $B = \{\text{True, False}\}$.

5.2 Computational Complexity

The most computationally expensive part of GBDT is evaluating the fitness of an expression since it involves visiting each record in the dataset then computing statistics. Furthermore, GBES requires a large number of expression evaluations to optimize the decision expression at each decision node. The deeper the tree, the more nodes that need to be optimized. However, as the tree gets deeper, the nodes operate on increasingly smaller fractions of the dataset. In fact, while the number of decision nodes grows exponentially with tree depth, the number of records that must be evaluated at each level remains constant (the size of the dataset). Overall, the computational complexity of GBDT induction is $O(|D| \cdot N_{GBES} \cdot d)$, where $|D|$ is the number of records in the dataset, N_{GBES} is the number of evaluations used in GBES, and d is the depth of the decision tree.

6 COLLISION AVOIDANCE APPLICATION

ACAS X monitors the airspace of an aircraft and issues alerts to the pilot if a conflict is detected. A resolution advisory (RA) is issued to help the pilot resolve the conflict, for example, instructing the pilot to climb at 1500 feet per minute. The collision avoidance system may revise an RA as the encounter unfolds. The pilot delays for five seconds before responding to an initial RA and three seconds before responding to subsequent RAs [17].

We apply GBDT to analyze simulated aircraft encounters to discover the most predictive properties of NMACs and categorize encounters according to those properties. The results of our study are used to help the ACAS X development team better organize and understand the NMACs and inform development.

6.1 Dataset

We analyze a dataset that contains simulation logs from an aircraft encounter simulator modeling a two-aircraft mid-air encounter [17]. Components in the simulator include sensors, pilot response, aircraft dynamics, and a development prototype of ACAS X. The dataset contains 10,000 encounters with 863 NMACs and 9137 non-NMACs. The class imbalance is due to the rarity of NMACs and the difficulty in generating NMAC encounters. We provide GBDT with the entire dataset so that the algorithm can learn from a larger set of examples. Each encounter has 38 attributes collected at 1 Hz for 50 time steps. The attributes are of mixed type that include numeric, categorical, and Boolean types. The attributes include the state of the aircraft, pilot commands, and the state and output of the collision avoidance system for each aircraft.

Since the NMAC condition (horizontal separation < 500 feet and vertical separation < 100 feet) is directly observable in the data, GBDT will split on this rule because it has the best fitness. While correct, the result does not provide useful information about the NMACs. We address the issue by filtering the NMAC event from the encounters. For each encounter, we compute the closest point of approach (CPA), which is the point where the separation of the two aircraft reaches a minimum, then trim the encounter to retain only data from the start of the encounter to five seconds prior to that time. Five seconds strike a good balance between removing CPA information and leaving sufficient encounter data for prediction.

6.2 Grammar

We crafted a custom CFG for the ACAS X dataset building on the grammar presented in Figure 1. We include temporal logic operators *eventually F* and *globally G*; elementwise logical operators *conjunct*

$\wedge$, *disjunct* $\vee$, *negation* $\neg$, and *implies* $\implies$; comparison operators *less than* $<$, *less than or equal to* $\leq$, *greater than* $>$, *greater than or equal to* $\geq$, and *equal* $=$; mathematical functions *absolute value* $|x|$, *difference* $-$, and *sign* SIGN; and computing function *count* COUNT (which returns the number of true values in a vector of Booleans).

In addition to dividing the attributes by data type, the ACAS X grammar further subdivides the attributes by their physical representations. The reason is to be able to compare attributes with constant values that have appropriate scale and resolution. For example, even though aircraft heading and vertical rate are both real-valued attributes, aircraft heading should be compared to values in the range of $-180°$ to $180°$, whereas vertical rate should be compared to values in the range of -80 feet per second to 80 feet per second.

6.3 Comparison of Induction Algorithms

We study the performance of GBDT when used together with Monte Carlo, MCTS, GE, and GP as subroutines. We evaluate the algorithms based on classification performance and interpretability of the produced models.

6.3.1 Classification Performance. We evaluate the classification performance of the models on the ACAS X dataset and report accuracy, precision, recall, and F1-score. A decision tree can always achieve perfect classification performance on a training set given a sufficient number of splits. However, large and deep trees are harder to interpret than smaller ones. These experiments limit the maximum tree depth to four.

6.3.2 Interpretability Metrics. We consider various metrics for quantifying the interpretability of the models. These metrics aim to capture the size and complexity of various parts of the model. Intuitively, large and complex models tend to be less interpretable than smaller ones.

- **Average rule length.** The average length of a rule measured in number of characters. In general, shorter rules are easier to interpret than longer ones.
- **Average rule nodes.** The average number of nodes in the derivation trees of the decision expressions. This metric aims to capture the complexity of a rule rather than only its representation length. In general, derivations with smaller number of nodes in its tree are less complex and thus easier to interpret.

We performed 20 random trials for each algorithm and report the mean of the results in Table 1. The best performing entry for each metric is highlighted.

Table 1: Comparison of GBDT Induction Algorithms

	MC	MCTS	GE	GP
Accuracy	96.58	96.35	96.57	**96.71**
F1-Score	0.8148	0.8086	0.8170	**0.8231**
Precision	0.7658	0.7384	0.7576	**0.7688**
Recall	0.8720	**0.8949**	0.8886	0.8875
Avg. Length	12.11	**11.38**	12.20	12.09
Avg. Nodes	12.78	**12.08**	12.79	12.73

In our experiments, GBDT-GP produced decision trees with the highest classification accuracy and F1-score, while GBDT-MCTS produced decision trees that were shorter and less complex. Overall, we selected the GBDT-GP induction algorithm for its better classification performance yet still competitive interpretability performance.

6.4 Categorizing Aircraft Encounters

We apply GBDT-GP to learn a categorization for the ACAS X dataset. A maximum tree depth of four is used. Figure 3 shows an example of the categorization results extracted from the leaf nodes of the tree. The figure shows plots of altitude versus time, which, while cannot fully capture the high-dimensionality of the encounter data, is generally most visually informative in this ACAS X application since ACAS X issues RAs only in the vertical direction. Since we are primarily interested in categorizing NMACs, we consider only NMAC categories. Figure 3 shows the first five encounters for each of the six NMAC categories identified, where each row is a separate category. The categories are labeled in ascending order starting at cluster 1 at the top row. The first row only shows two plots because category 1 only contained two encounters.

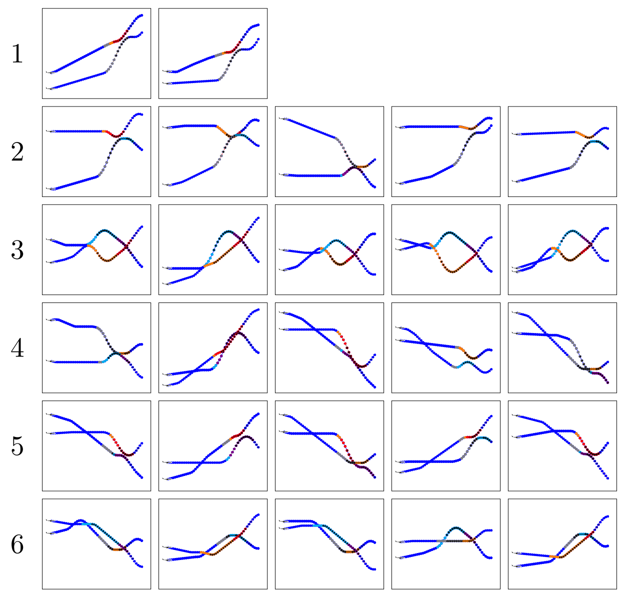


Figure 3: Visual Overview of Categorized ACAS X Encounter Data. Each Row is a Category.

The following sections summarize the properties that distinguish each category and discuss their relevance. Because the categories are derived from leaf nodes at depth four, each category is described by the conjunction of four expressions. The hierarchical nature of GBDT means that some decision expressions are shared between categories while others are not. For each category, we first present the natural language output of the decision expressions, and then provide an analysis of the encounters.

6.4.1 *Category 1.*

- **False** "For all time, [the absolute difference between [pilot 1's commanded vertical rate] and [pilot 2's commanded vertical rate] is less than 50 ft/s]"
- **False** "At some point, [[absolute altitude difference] is less than or equal to 100 ft]"
- **False** "Whenever [the sign of [aircraft 2's vertical rate] is equal to the sign of [aircraft 1's vertical rate]], it is also true that [pilot 1 is flying intended trajectory]"
- **False** "At some point, [[pilot 2's commanded vertical rate] is less than or equal to 1 ft/s]"

The encounters in this category are characterized by both aircraft climbing, maintaining vertical separation, then the lower aircraft drastically increasing climb rate toward the upper aircraft near CPA. The aggressive maneuvering causes vertical separation to be lost rapidly and results in an NMAC. Figure 4 shows the aircraft vertical rates of an encounter where the large difference in vertical rate just before NMAC is clearly visible at 34 seconds and NMAC occurs at 39 seconds. Text labels in figure indicate the issued RAs. An asterisk at a time step indicates that the pilot is following active RA and a dash indicates that the pilot is following the previous RA.

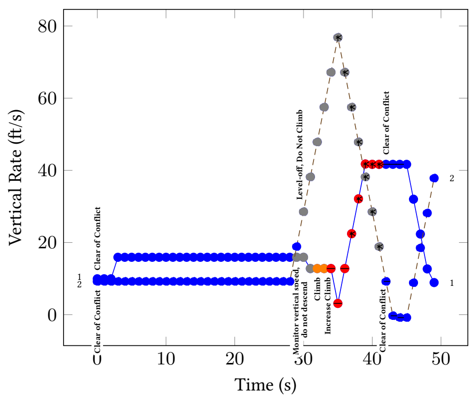


Figure 4: Vertical Maneuvering Just Before CPA

Aggressive maneuvering close to CPA is known to be problematic. With the pilot's five-second response time, there is insufficient time remaining for the collision avoidance system to resolve the conflict. The encounter is exacerbated by having a climbing aircraft being approached from below, which greatly reduces maneuvering options for the upper aircraft. Fortunately, these large sudden maneuvers, and thus these encounters, are extremely rare in practice.

6.4.2 *Category 2.*

- **False** "For all time, [the absolute difference between [pilot 1's commanded vertical rate] and [pilot 2's commanded vertical rate] is less than 50 ft/s"
- **False** "At some point, [[absolute altitude difference] is less than or equal to 100 ft]"
- **True** "Whenever [the sign of [aircraft 2's vertical rate] is equal to the sign of [aircraft 1's vertical rate]], it is also true that [pilot 1 is flying intended trajectory]"
- **False** "Whenever [it is not true that [the absolute value of [pilot 1's commanded vertical rate] is less than 10 ft/s]], it is also true that [the absolute difference between [pilot 2's commanded vertical rate] and [aircraft 2's RA target rate] is less than 30 ft/s]"

The encounters in this category are also characterized by a maintenance of vertical separation in the first part of the encounter before aggressive maneuvering close to CPA. The relative vertical rate of the aircraft exceeds 50 feet per second at some point in the encounter. However, in these encounters, aircraft 1 does not receive an advisory until close to CPA where the large maneuvering occurs. As a result, pilot 1 has not yet started to respond to the RA at 5 seconds to CPA. The aggressiveness of pilot 2's maneuvering is reiterated in the property that the commanded vertical rate differs from the advisory's target rate by more than 30 feet per second. That is, the pilot is not complying with the RA. These encounters differ from those in category 1 in that the aircraft maintains a greater altitude separation in the first part of the encounter and both aircraft simultaneously maneuver toward each other near CPA.

6.4.3 *Category 3.*

- **False** "For all time, [the absolute difference between [pilot 1's commanded vertical rate] and [pilot 2's commanded vertical rate] is less than 50 ft/s]"
- **True** "At some point, [[absolute altitude difference] is less than or equal to 100 ft]"
- **False** "[The number of times [the sign of [aircraft 2's RA target rate] is equal to the sign of [aircraft 1's RA target rate]] is greater than 23]"
- **True** "[The number of times [[pilot 1 is responding to previous RA] or [RA alarm occurs on aircraft 2]] is less than or equal to 1]"

Similar to categories 1 and 2, the encounters in this category also contain aggressive vertical maneuvering at some point in the encounter. However, the aggressive maneuvering occurs earlier in the encounter causing the aircraft to become co-altitude prior to CPA. Generally the maneuvering leads to one or more altitude crossings early on and results in active RAs throughout most of the encounter. Figure 5 shows the vertical rates of an encounter where the large difference in vertical rate occurs well prior to CPA in the encounter. The large difference in vertical rates is seen at 25 seconds and NMAC occurs at 37 seconds.

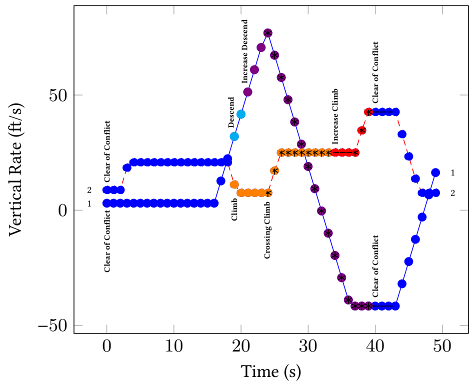


Figure 5: Vertical Maneuvering Early in Encounter

6.4.4 *Category 4.*

- **False** "For all time, [the absolute difference between [pilot 1's commanded vertical rate] and [pilot 2's commanded vertical rate] is less than 50 ft/s]"
- **True** "At some point, [[absolute altitude difference] is less than or equal to 100 ft]"
- **True** "[The number of times [the sign of [aircraft 2's RA target rate] is equal to the sign of [aircraft 1's RA target rate]] is greater than 23]"
- **False** "Whenever [the sign of [aircraft 2's RA target rate] is equal to the sign of [pilot 1's commanded vertical rate]], it is also true that [the absolute value of [aircraft 2's vertical rate] is less than 10 ft/s]"

The encounters in this category also contain aggressive vertical maneuvering close to CPA. Loss of vertical separation before 5 seconds prior to CPA is manifested either early in the encounter or as an extended period where the aircraft are co-altitude near CPA. Compared to category 3, the RAs occur later in the encounter, which is reflected in the property that the RA target rates are of the same sign for a large part of the encounter. Target rates are zero and thus have the same sign when no RA is active. Together, these properties suggest that the aircraft are co-altitude without active advisories in the early part of the encounter. Additionally, there is a time in the encounter where aircraft 2's advisory and aircraft 1's commanded vertical rate are in the same direction and aircraft 2's vertical rate exceeds 10 feet per second. This property is largely manifested as pilot 2 aggressively maneuvering against the advisory with a high vertical rate leading to NMAC.

6.4.5 *Category 5.*

- **True** "For all time, [the absolute difference between [pilot 1's commanded vertical rate] and [pilot 2's commanded vertical rate] is less than 50 ft/s]"
- **False** "At some point, [pilot 2 is responding to current RA]"
- **False** "At some point, [the absolute difference between [aircraft 1's vertical rate] and [aircraft 2's vertical rate] is less than 1 ft/s]"
- **False** "Whenever [[number of seconds remaining in pilot 2's response delay] equals 5], it is also true that [the absolute difference between [aircraft 1's RA target rate] and [aircraft 2's vertical rate] is less than 20 ft/s]"

In these encounters, the aircraft cross in altitude early in the encounter without active advisories, then maneuver back toward each other to cause an NMAC. Since the aircraft are estimated to cross safely and appear to vertically diverge following the crossing, the collision avoidance system witholds an RA to reduce the number of unnecessary alerts. However, after crossing, the aircraft make a sudden maneuver toward each other that results in an NMAC. Because the aircraft are already close in altitude, the vertical maneuvering in these encounters is less aggressive than those in previous categories. Due to the late maneuvering and the pilot response delay, pilot 2 does not start to comply with the issued RA until within five seconds of CPA. Figure 6 shows the vertical profile of an encounter in this category where the aircraft cross in altitude at 19 seconds and NMAC at 38 seconds.

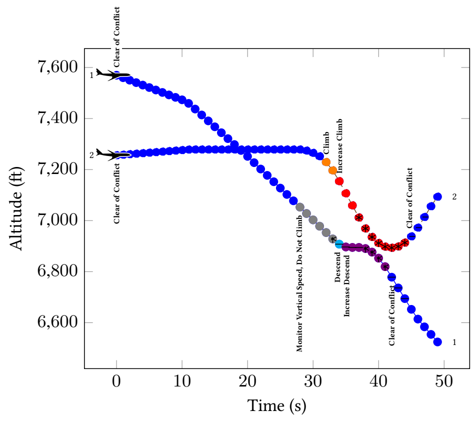


Figure 6: Maneuvering Following Altitude Crossing

6.4.6 *Category 6.*

- **True** "For all time, [the absolute difference between [pilot 1's commanded vertical rate] and [pilot 2's commanded vertical rate] is less than 50 ft/s]"
- **True** "At some point, [pilot 2 is responding to current RA]"
- **True** "At some point, [crossing RA is issued to aircraft 1]"
- **False** "[The number of times [[altitude difference relative to aircraft 1] is less than or equal to 50 ft] is greater than 21]"

In these encounters, an initial RA is issued early in the encounter and the aircraft cross in altitude during the pilot's initial response delay. As the pilots respond to the RA, their maneuvers actually result in a reduction in vertical separation rather than increasing it. The encounter eventually leads to a crossing RA to be issued late in the encounter before resulting in an NMAC. A crossing RA is one where the aircraft are expected to cross in altitude following the RA. Issuing advisories can be problematic in cases where aircraft are approximately co-altitude due to the uncertainty in the aircraft's estimated positions and future intentions. In these encounters, the problem arises as one aircraft maneuver and crosses altitude immediately after an initial RA is issued to the other aircraft. Figure 7 shows the vertical profile of an encounter with these properties. The aircraft cross in altitude during pilot 2's response delay period at 21 seconds and NMAC occurs at 39 seconds.

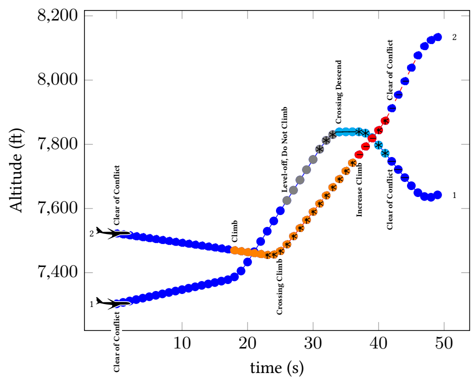


Figure 7: Altitude Crossing During Response Delay

7 CONCLUSIONS

This paper introduced GBDT, a framework that combines decision trees and grammar-based expression search for interpretable classification and categorization. GBDT was developed specifically to address the need for interpretable models that can support high-dimensional and heterogeneous time series data. To further improve interpretability, we showed a method to automatically generate English sentences by providing a map of subexpressions to sentence fragments. We applied GBDT to categorize an aircraft encounter dataset and showed that the method produces interpretable and insightful categories. Our approach not only partitions a dataset into similar groups, but also explains the relevant properties of each group. The source code for GBDT and the experiments in this paper are available online [16].

ACKNOWLEDGEMENTS

We thank Neal Suchy at the FAA, Michael Owen, Robert Klaus, and Cynthia McLain at MIT Lincoln Laboratory as well as others on the ACAS X team for their feedback and continued support of the project. We gratefully thank Max Egorov for sharing code with us. This work was supported by the Safe and Autonomous Systems Operations (SASO) Project under NASA ARMD's Airspace Operations and Safety Program (AOSP).

REFERENCES

[1] Aniruddha Basak, Ole Mengshoel, Stefan Hosein, and Rodney Martin. 2016. Scalable Causal Learning for Predicting Adverse Events in Smart Buildings. In *Workshops at the Thirtieth AAAI Conference on Artificial Intelligence.*

[2] Dimitris Bertsimas, Allison Chang, and Cynthia Rudin. 2011. ORC: Ordered Rules for Classification A Discrete Optimization Approach to Associative Classification. (2011).

[3] Leo Breiman, Jerome Friedman, Charles J Stone, and Richard A Olshen. 1984. *Classification and Regression Trees.* CRC Press.

[4] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton. 2012. A Survey of Monte Carlo Tree Search Methods. *IEEE Transactions on Computational Intelligence and AI in Games* 4, 1 (2012), 1–43.

[5] Tristan Cazenave. 2013. Monte-Carlo Expression Discovery. *International Journal on Artificial Intelligence Tools* 22, 01 (2013), 1250035.

[6] Pedro G Espejo, Sebastián Ventura, and Francisco Herrera. 2010. A Survey on the Application of Genetic Programming to Classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part C* 40, 2 (2010), 121–144.

[7] Jesús K Estrada-Gil, Juan C Fernández-López, Enrique Hernández-Lemus, Irma Silva-Zolezzi, Alfredo Hidalgo-Miranda, Gerardo Jiménez-Sánchez, and Edgar E Vallejo-Clemente. 2007. GPDTI: A Genetic Programming Decision Tree Induction Method to Find Epistatic Effects in Common Complex Diseases. *Bioinformatics* 23, 13 (2007), i167–i174.

[8] Dov M. Gabbay, Amir Pnueli, Saharon Shelah, and Jonathan Stavi. 1980. On the Temporal Analysis of Fairness. In *Conference Record of the Seventh Annual ACM Symposium on Principles of Programming Languages, Las Vegas, Nevada, USA, January 1980.* 163–173.

[9] Trevor J Hastie and Robert J Tibshirani. 1990. *Generalized Additive Models.* Vol. 43. CRC Press.

[10] Been Kim, Cynthia Rudin, and Julie A Shah. 2014. The Bayesian Case Model: A Generative Approach for Case-Based Reasoning and Prototype Classification. In *Advances in Neural Information Processing Systems.* 1952–1960.

[11] Mykel J Kochenderfer, Jessica E Holland, and James P Chryssanthacopoulos. 2012. Next-Generation Airborne Collision Avoidance System. *Lincoln Laboratory Journal* 19, 1 (2012), 17–33.

[12] Levente Kocsis and Csaba Szepesvári. 2006. Bandit Based Monte-Carlo Planning. In *European Conference on Machine Learning (ECML).* 282–293.

[13] John R. Koza. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA.

[14] James K Kuchar and Ann C Drumm. 2007. The Traffic Alert and Collision Avoidance System. *Lincoln Laboratory Journal* 16, 2 (2007), 277–296.

[15] H. Lakkaraju, S. Bach, and J. Leskovec. 2016. Interpretable Decision Sets: A Joint Framework for Description and Prediction. In *ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD).*

[16] Ritchie Lee. 2017. Grammar-Based Decision Trees (GBDTs) Julia Package. (2017). https://github.com/sisl/GrammarExpts.jl.git

[17] Ritchie Lee, Mykel J. Kochenderfer, Ole J. Mengshoel, Guillaume P. Brat, and Michael P. Owen. 2015. Adaptive Stress Testing of Airborne Collision Avoidance Systems. In *Digital Avionics Systems Conference (DASC).* Prague, Czech Republic.

[18] Benjamin Letham, Cynthia Rudin, Tyler H McCormick, David Madigan, and others. 2015. Interpretable Classifiers using Rules and Bayesian Analysis: Building a Better Stroke Prediction Model. *The Annals of Applied Statistics* 9, 3 (2015), 1350–1371.

[19] Yin Lou, Rich Caruana, and Johannes Gehrke. 2012. Intelligible Models for Classification and Regression. In *Proceedings of the 18th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.* ACM, 150–158.

[20] Robert E Marmelstein, Gary B Lamont, John R Koza, and Wolfgang Banzhaf. 1998. Pattern Classification using a Hybrid Genetic Program. In *Genetic Programming 1998: Proceedings of the Third.* Morgan Kaufmann, 223–231.

[21] Robert I Mckay, Nguyen Xuan Hoai, Peter Alexander Whigham, Yin Shan, and Michael O'Neill. 2010. Grammar-Based Genetic Programming: A Survey. *Genetic Programming and Evolvable Machines* 11, 3-4 (2010), 365–396.

[22] Stephen Muggleton. 1995. Inverse Entailment and Progol. *New Generation Computing* 13, 3-4 (1995), 245–286.

[23] Stephen Muggleton, Luc De Raedt, David Poole, Ivan Bratko, Peter Flach, Katsumi Inoue, and Ashwin Srinivasan. 2012. ILP Turns 20. *Machine Learning* 86, 1 (2012), 3–23.

[24] Po Shun Ngan, Man Leung Wong, Kwong Sak Leung, and Jack CY Cheng. 1998. Using Grammar Based Genetic Programming for Data Mining of Medical Knowledge. *Genetic Programming* (1998), 254–259.

[25] Michael O'Neil and Conor Ryan. 2003. Grammatical Evolution. In *Grammatical Evolution.* Springer, 33–47.

[26] Riccardo Poli, William B Langdon, Nicholas F McPhee, and John R Koza. 2008. *A Field Guide to Genetic Programming.*

[27] J. Ross Quinlan. 1986. Induction of Decision Trees. *Machine Learning* 1, 1 (1986), 81–106.

[28] Ronald L Rivest. 1987. Learning Decision Lists. *Machine Learning* 2, 3 (1987), 229–246.

[29] Holger Schielzeth. 2010. Simple Means to Improve the Interpretability of Regression Coefficients. *Methods in Ecology and Evolution* 1, 2 (2010), 103–113.

[30] Pavel Senin and Sergey Malinchik. 2013. SAX-VSM: Interpretable Time Series Classification using SAX and Vector Space Model. In *Data Mining (ICDM), 2013 IEEE 13th International Conference on.* IEEE, 1175–1180.

[31] Richard S. Sutton and Andrew G. Barto. 1998. *Reinforcement Learning: An Introduction.* MIT Press, Cambridge, MA.

[32] Toru Tanigawa and Qiangfu Zhao. 2000. A Study on Efficient Generation of Decision Trees using Genetic Programming. In *Proceedings of the 2nd Annual Conference on Genetic and Evolutionary Computation.* Morgan Kaufmann Publishers Inc., 1047–1052.

[33] Athanasios Tsakonas. 2006. A Comparison of Classification Accuracy of Four Genetic Programming-Evolved Intelligent Structures. *Information Sciences* 176, 6 (2006), 691–724.

[34] Peter A Whigham and others. 1995. Grammatically-Based Genetic Programming. In *Proceedings of the Workshop on Genetic Programming: From Theory to Real-World Applications*, Vol. 16. 33–41.

[35] Lexiang Ye and Eamonn Keogh. 2011. Time Series Shapelets: A Novel Technique that Allows Accurate, Interpretable and Fast Classification. *Data Mining and Knowledge Discovery* 22, 1 (2011), 149–182.