IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 7 November 2022, accepted 6 December 2022, date of publication 14 December 2022,
date of current version 22 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3229008

RESEARCH ARTICLE

Topological Forest

MURAT ALI BAYIR[1], (Member, IEEE), KIARASH SHAMSI[2], HUSEYINCAN KAYNAK[3], AND CUNEYT GURCAN AKCORA[2], (Member, IEEE)

[1]IEEE Computer Society, Washington, DC 20036, USA
[2]Department of Computer Science, University of Manitoba, Winnipeg, MB R3P 2V3, Canada
[3]MilSOFT Software Technologies, 06490 Ankara, Turkey

Corresponding author: Murat Ali Bayir (muratbayir@ieee.org)

This work of Cuneyt Gurcan Akcora was supported by the Canadian NSERC Discovery Grant RGPIN-2020-05665.

ABSTRACT We propose a new ML model called Topological Forest that contains an ensemble of decision trees. Unlike a vanilla Random Forest, Topological Forest has a special training process that selects a smaller number of decision trees on a topological graph representation that TDA Mapper constructs. Compared to Vanilla Random Forest, Topological Forest significantly improves the computational efficiency of inference time due to the smaller ensemble size and selection of better decision trees while keeping the diversity of decision trees. Our experiments show that Topological Forest can speed up inference time by more than 100x on average while compromising at most 2% reduction in the AUC metric for the prediction quality.

INDEX TERMS Random forest, graph topology, TDA mapper.

I. INTRODUCTION

Random forests [1] are a type of ensemble learning model that use multiple decision trees to make predictions or perform classification tasks. In a random forest, each decision tree is trained on a subset of the features and training data, and the results of all the decision trees are combined to make more robust predictions.

Random forests have gained popularity in recent years due to their simplicity and efficiency [2], [3], [4], [5], [6], [7]. Unlike more complex models like gradient boosted decision trees [8] or deep neural networks [9], random forests can be easily parallelized on a cluster of machines, which makes them faster to train and use for inference. Additionally, the decision rules from the split nodes in a random forest can be easily interpreted by humans, which makes them more transparent than other, more complex models. These advantages make random forests attractive for many classification and regression tasks.

While random forests have many advantages, the qualities and contributions of their individual decision trees are often not considered. Previous attempts to tune the number of decision trees [10], [11] and understand the impact of ensemble size on performance [12] have focused on determining the upper bounds of ensemble size and the optimal number of decision trees, rather than analyzing the quality of individual trees or their contribution to the ensemble. In this paper, we propose a training process that takes into account the similarity and quality of decision trees in a random forest. By carefully selecting decision trees in this way, we achieve similar performance on prediction tasks with a smaller ensemble size. This would reduce the size of the model and improve computational efficiency during inference.

In this paper, we propose the Topological Forest model, which uses a smaller ensemble of decision trees to perform regression and classification tasks. The Topological Forest model has a multi-step training process that begins by training a vanilla random forest. Next, we transform each tree in the random forest into a feature graph and extract key features from the graphs as high-dimensional vectors. We then use TDA Mapper [13] to transform these high-dimensional feature vectors into a topological cluster network. This allows us to identify representative decision trees that can be used to construct a smaller, more efficient random forest. By using this approach, we aim to achieve similar performance on prediction tasks with a smaller ensemble size, reducing the model size and improving computational efficiency.

TDA Mapper is a technique that allows for the soft clustering of high-dimensional data points while constructing a network view where similar clusters are connected by short

The associate editor coordinating the review of this manuscript and approving it for publication was Xujie Li.

This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/

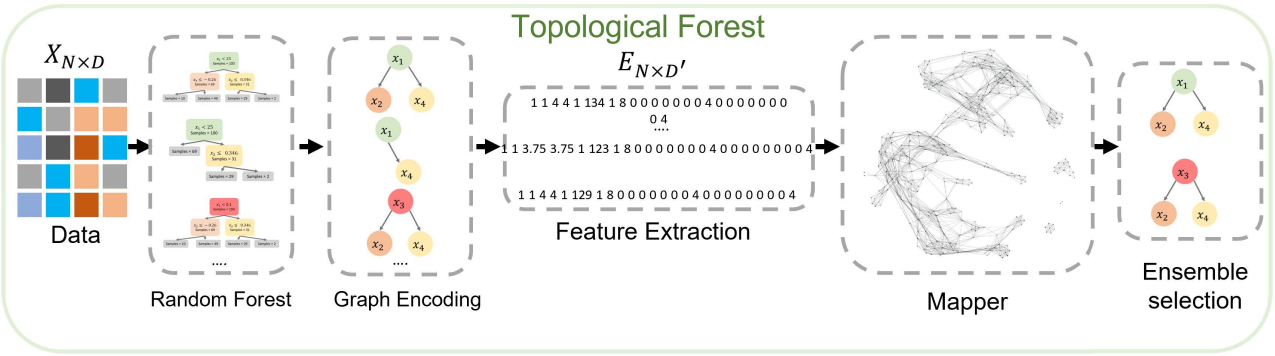

FIGURE 1. Building blocks of the topological forest.

paths. TDA Mapper transforms the high-dimensional input features into a low-dimensional space (in this case, 2D) using a lens or filter function. Common choices for the filter function include projection onto one or more axes using techniques like TSNE [14] or density-based methods. Once the filter function is applied, the data in the original feature space is transformed into 2D space. TDA Mapper then constructs a set cover of the projected space in the form of a set of overlapping intervals with constant length.

Next, TDA Mapper constructs a mapper graph with clustered trees as vertices that represent each set in the cover of the projected space. An edge exists between two vertices if the two sets (due to the overlapping intervals) share some trees in common. Finally, we use different strategies to select decision trees from each cluster to discover a more representative tree ensemble with fewer trees. Our experimental results show that trees in a cluster make similar predictions on out-of-bag samples, which makes TDA Mapper well-suited to our tree selection problem.

In particular, we list the contributions of this paper as follows:

- We propose a new machine learning model called Topological Forest, which uses a significantly smaller number of decision trees than traditional random forest models while achieving similar prediction quality (within 2%). This reduction in the number of decision trees allows for a smaller model size and improved computational efficiency.
- We introduce a novel graph representation of decision trees that can be used in various applications to take advantage of tree features and decision tree similarities. This graph representation allows for more effective analysis and interpretation of decision trees and can be used in a variety of contexts.
- We use TDA Mapper to map similar decision trees into the same clusters, allowing us to construct a more effective random forest by applying various ensemble selection strategies.
- From our experiments on several binary classification tasks, we find that the Topological Forest model significantly improves computational efficiency during inference, with minimal compromise on prediction quality.

The structure of this paper is as follows: The next section provides an overview of related work in the field. In Section III, we introduce the Topological Forest model and its training process. Section IV presents our experimental results. Finally, in Section V, we provide our conclusions and discuss potential future work.

II. RELATED WORK

Random Forest is a popular ensemble-based Machine Learning model that is used for classification or regression tasks. It works by constructing a prediction value from individual decision trees, using different aggregation policies such as majority voting, computing the mean, or median, depending on the type of problem. Because of its simplicity and efficiency, the Random Forest model has been applied successfully to many practical problems, such as patient health prediction, image classification, emotion recognition, malware detection, and user response prediction.

A significant advantage of Random Forest comes from its interpretability [17], [18], and there have been several works [19], [20], and [21] to improve the explainability of Random Forest. For example, decision tree split rules in Random Forest can easily be converted into human-readable rule format [22] to understand the relationship between feature space and the predicted outcome.

Parallelization of training and inference processes [23], [24] is another significant advantage of Random Forest. We can train all decision trees in parallel, and during the inference, we can invoke all decision trees concurrently to compute predicted outcomes. This makes Random Forest very efficient and suitable for large-scale applications that require processing big data [16], [25], [26], [27].

While Random Forest has the advantages mentioned above, the ensemble size can be unnecessarily large without a significant contribution to the prediction performance. Previous attempts to limit ensemble size focused on tuning the number of decision trees during the training process [10], [12], [28], [29]. Although these approaches yield improvements in the run-time efficiency of Random Forest, they don't analyze the individual quality of decision trees or the similarity between decision trees to boost quality.

We use topological data analysis to analyze our decision trees and make the random forest more efficient in computational costs of inference with a small compromise on prediction accuracy. TDA considers that data has a shape and aims to investigate the underlying manifold structure of the data rather than just using the statistical description [30]. TDA is mainly powered by persistent homology [31], which can capture topological differences across various scales and depict them in persistent diagrams. This novel data analysis approach is rapidly developing for different purposes in machine learning research.

TDA can be used to extract topological features from data to use them as inputs for the machine learning models, or it can be used to improve the model's design and study some aspects of the model [32]. In terms of TDA feature extraction, Harer et al. [33] used total persistence of a persistence diagram, Chen et al. [34] applied p-norm, and Atienza et al. [35] used persistent entropy as a topological descriptor input feature. Rieck et al. [36] used Betti curves [37] to summarize the features. Chevyrev et al. [38] utilized this representation and persistent diagrams to develop a classifier using a random forest and support vector machine. Bubenik et al. presented persistent landscape, a new topological descriptor for mapping the persistent diagram to function space [39]. Zhao et al.used TDA features for graph classification [40]. These studies showed that TDA features could be well-suited as inputs for machine learning models to improve their accuracy.

Some studies applied TDA to design a machine learning model or improve some aspects of a model. Moor et al. presented a topological auto-encoder for low-dimensional representation of input data features [41]. Yuvaraj et al. used TDA to study complex multilayer networks [42] and to cluster them based on topological approaches, and Bulauan et al. [43] clustered complex multilayer networks with topological approaches. Chen et al. [44] introduced an approach for measuring the classification boundary of a classifier by using a topological complexity, and Hofer et al. [45] developed topological constraint to improve the generalization performance of their model. Our method is categorized in the second area of research on topological data analysis. TDA helps improve our Random Forest quality and considerably increases the model's performance.

Our proposed approach is unique because topological random forests can reduce the ensemble size and improve the speed of predictions compared to vanilla Random Forest at the cost of reducing at most 2% in the AUC metric. Topological Forest also keeps the diversity of decision trees by selecting individual decision trees that belong to different similarity classes.

III. TOPOLOGICAL RANDOM FOREST

In this section, we'll explain the details of the Topological Forest: We present a high-level overview of the training process for the Topological Forest. After that, we briefly describe a vanilla Random Forest and introduce the notation we use throughout the paper. Next, we explain the graph encoding of decision trees and discuss how a Random Forest ensemble is constructed from a topological clustering of encoded decision tree representations.

A. OVERVIEW OF THE TRAINING PROCESS

Building a Topological Forest from the training data is a multi-step process, as shown in Figure 1. In the first step, we train a vanilla random forest by using the original dataset with all its features. After that, each decision tree in vanilla random forest is transformed into the graph representation by using relationships between features. In the next step, we extract features from the graph representation of decision trees as N-dimensional vectors. These vectors have topological features like the number of edges or the average degree of vertexes.

Once feature extraction is completed, we use TDA Mapper to transform and cluster N-dimensional feature vectors into a topological network that has a cluster of nodes (each node represents a decision tree). TDA mapper step first invokes a feature embedding task by leveraging TSNE as a filter function to transform N-dimensional feature vectors in 2D. Once compressed features in 2D space are computed, TDA mapper generates a mapper graph with clustered trees as vertices by leveraging a set cover of the projected space in the form of overlapping intervals with fixed lengths.

As Figure 1 shows, we employ two types of graphs to build Topological Forest: a graph representation of a decision tree and a TDA Mapper produced graph (aka mapper graph) to cluster decision trees. The first graph is relatively smaller than the mapper graph and it contains parent and child relationships of features. On the other hand, the mapper graph is a topological network of soft decision tree clusters, and an edge between two clusters exists if they share common decision trees. In the final step of the entire process, we employ an ensemble selection task to build a topological forest from the mapper graph. In the ensemble selection task, we used different selection algorithms to pick decision trees to construct a topological forest from the cluster of decision trees in the mapper graph. In the following sections, we give details of each step.

B. TRAINING VANILLA RANDOM FOREST

We consider a binary classification scheme for ease of exposition, but our approach can be trivially generalized to multi-label classification. Let $\{x_n\}_{n\in Z^+}$ be a set of data points, and let each point x_n be associated with a pair $(\vec{x}_n, y_n)$, where $\vec{x}_n \in \mathcal{R}^D$ is a feature vector and $y_n \in \{0, 1\}$ is a label. There are D features for each data point x_n, and a total of N data points in the training set.

Definition 1 (Random Forest): A Random Forest is a classifier consisting of a collection of decision tree classifiers $\{h(X, Y, \Theta_k), k = 1, \ldots, N\}$ *where the* Θ_k *are independent identically distributed random vectors [1].*

Each tree in Random Forest is trained on a randomly selected subset of data through data bagging. Once the

Random Forest is constructed, each tree casts a unit vote for a class of a given input $\vec{x}_n \in X$.

C. GRAPH ENCODING AND FEATURE EXTRACTION

In this step, we convert each decision tree into a graph structure. Our decision to convert a tree into a graph is motivated by two reasons:

- First, creating a graph from a decision tree allows us to generalize parent-child relations as weighted feature-feature edges, which create summary, interpretable representations of large decision trees (see Figure 5).
- Second, although extracting features of a decision tree is under-studied in the machine learning community, graph feature extraction is a well-studied problem. To illustrate; graph ML offers features for vertices (e.g., eccentricity [46]), edges (e.g., edge centrality [47]), subgraphs (e.g., modularity [48]) and graphs (e.g., clustering coefficient [49]) which can encode the trees of a random forest from various aspect. The encoding brings us closer to measuring the performance of tree ensembles in a principled way.

In the graph representation of each decision tree $T = V \times E$, each vertex $v \in V$ corresponds to a feature $d \in D$, and a directed edge between two vertices $e = \langle v_i, v_j \rangle \in E$ denotes a parent-child split of $d_i \rightarrow d_j$ in the decision tree. A decision tree can split the same feature multiple times into different levels; however, we do not create a duplicate vertex for each new split. Instead, we add duplicate edges from the parent vertex to the child vertex for new parent-child splits. As a result, the encoded graph may contain duplicate edges between vertex pairs. We also ignore feature values in splits during the entire encoding process because traditional graph features do not utilize attributes of vertexes.

Consider example in Figure 3 to understand how the decision tree is transformed into a graph representation. Since our goal here is to show the mapping between the decision tree and graph representation, we omit the example data set and hyperparameters that are used to construct decision tree in Figure 2. In the decision tree, let's assume that the feature x_1 of the root node has a split value of 25 in Figure 2. However, we do not retain this information in the graph representation of this decision tree in Figure 3. The reader should notice that there is only one node for each feature in the graph representation in Figure 3. However, there are two edges between x_1 and x_2 in Figure 3 since there are two child nodes that split on x_2 of the root node, which splits on x_1.

The graph kernel [50] or graph neural network [51] approaches can incorporate the multi-set of split values as vertex features in the learning process. However, such approaches may create high computational costs in the training process. For this reason, we leave the kernel and graph neural network approaches as future work.

We detail the process for encoding a graph from a decision tree in Algorithm 1 where we employ a breadth-first traversal to discover the parent-child relations in feature splits. The

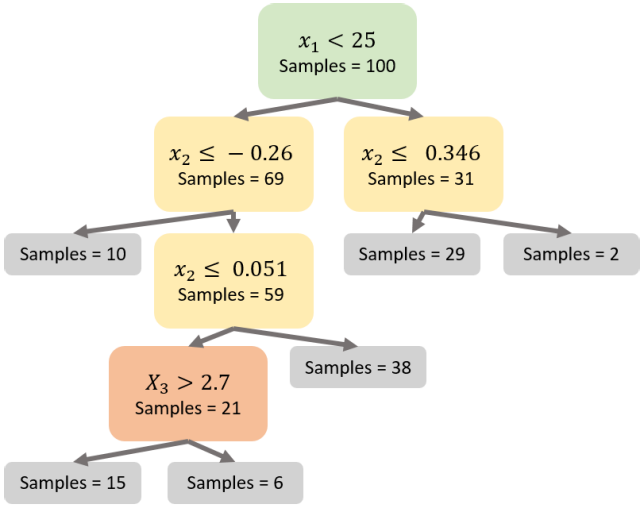

FIGURE 2. Decision tree built on 100 data points. The tree utilizes features x_1, x_2, and x_3 and creates five feature splits (10 children nodes, where 6 of them are leaf nodes). In six of these splits, a further split is not required (we show them as the gray leaf vertices). On the other hand, in four splits, a further feature split is required.

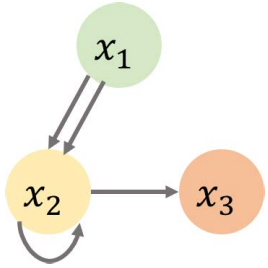

FIGURE 3. Graph representation of the decision tree shown in Figure 2. The four parent-child feature relationships are shown with four directed edges. Feature x_2 has a self-loop because the decision tree uses it in two consecutive splits.

algorithm takes a decision tree as an input and outputs a directed graph structure such as the one shown in Figure 3. In the edge case where the decision tree has only a root node without any split, the algorithm would create a graph that consists of one vertex with special empty feature label without any edges. In all other cases, the algorithm creates at least one feature vertex since there must be at least one split at root node. In cases where tree depth is more than 1, the algorithm creates at least one edge.

We outline three approaches to graph encoding: traditional features, graph kernel [50], or graph neural network [51]. Traditional features and graph kernel approach are considered shallow graph encodings because the output is a low-dimensional feature vector. Graph neural networks create deep encodings that are high-dimensional feature matrices. Deep encoders frequently outperform shallow encoders in ML tasks (see Tables 3 and 4 in [52]); however, the performance comes at a great computational cost which we want to avoid for this task.

We used two main approaches to encode a graph in traditional graph features. We extract individual vertex features, such as vertex degree, in the first approach. Next, we design a

Input: Decision tree T
Output: Directed multi-graph $G(V, E)$
Initialize $\mathcal{G}$ as a directed multi-graph;
Initialize $\mathcal{S}$ as a multi-set;
$r \leftarrow$the root of T;
// Insert root into multi-set S
$S \leftarrow S \cup \{r\}$;
while *S is not empty* **do**
  $r_x \leftarrow -$ pick the first element in S;
  $S \leftarrow S \setminus r_x$;
  if r_x *is not a leaf* **then**
    if $r_x \notin \mathcal{G}.\mathcal{V}$ **then**
      $\mathcal{G}.\mathcal{V} \leftarrow \mathcal{G}.\mathcal{V} \cup \{r_x\}$;
    end
    if $r_x.left$ *is not a leaf* **then**
      if $r_x.left \notin \mathcal{G}.\mathcal{V}$ **then**
        $\mathcal{G}.\mathcal{V} \leftarrow \mathcal{G}.\mathcal{V} \cup \{r_x.left\}$;
        $S \leftarrow S \cup \{r_x.left\}$
      end
      $\mathcal{G}.\mathcal{E} \leftarrow \mathcal{G}.\mathcal{E} \cup \langle r_x, r_x.left\rangle$;
    end
    if $r_x.right$ *is not a leaf* **then**
      if $r_x.right \notin \mathcal{G}.\mathcal{V}$ **then**
        $\mathcal{G}.\mathcal{V} \leftarrow \mathcal{G}.\mathcal{V} \cup \{r_x.right\}$;
        $S \leftarrow S \cup \{r_x.right\}$;
      end
      $\mathcal{G}.\mathcal{E} \leftarrow \mathcal{G}.\mathcal{E} \cup \langle r_x, r_x.right\rangle$;
    end
  end
end
return $\mathcal{G}$

Algorithm 1 Graph Creation From a Decision Tree

readout function, which can be as simple as averaging values to pool vertex features. In the second approach, we ignore vertex features and directly extract graph-level features such as graph diameter and the number of strongly connected components. We follow a combination of both approaches and extract the following features on both vertex and graph levels:

- Vertex:
  - in-degree of a vertex
  - out-degree of a vertex
  - degree of a vertex
  - vertex path distance to all vertices
  - betweenness centrality of a vertex
- Graph:
  - diameter of the graph
  - number of vertices
  - number of edges
  - counts (16) of directed three-vertex motifs [53]
  - clustering coefficient

Additionally, we have tested several vertexes and graph features such as numbers of strongly and weakly connected components and hub/authority scores [54]. For reporting purposes, we only provide features that improve the performance of Topological Forest on out-of-bag test data.

Computational cost: Extracting graph encodings of an individual decision tree is a non-trivial operation. Features like betweenness centrality requires a time complexity of $\mathcal{O}(|V| \times |E|)$ [55] and motif counting requires $\mathcal{O}(|E|)$ [53]. We use the vanilla clustering coefficient implementation of the Jung library [56] whose time complexity is $\mathcal{O}(|V|^3)$. Most datasets on the UCI repository have less than 50 features. As a result, the graphs that we extract from decision trees are quite small, with less than 50 vertices. Furthermore, decision tree depth is bounded by the number of training data points. As a result, there are less than 200 parent-child relationships recorded as edges in these graphs. For these reasons, the graph representation creates quite small graphs (Figure 5) (i.e., $|V| < 50$ and $|E| < 200$ for most datasets). As a result, computational costs of graph representation and encoding in Topological Forest are negligible.

We average vertex features to find the corresponding graph feature (e.g., average vertex degree in the graph). Combining five averaged vertex features and 20 graph features, we create a decision tree representation $\vec{e} \in \mathcal{R}^{D'}$ where $D' = 25$. This representation allows us to compare and contrast decision trees which may use different features, split values, and tree depth.

D. TOPOLOGICAL CLUSTERING

We employ the highly customizable TDA tool Mapper [57] to analyze and cluster decision trees in the original vanilla random forest. TDA Mapper complements traditional clustering, and projection pursuit approaches with a systematic insight into data geometry and topology. It uncovers hidden data patterns that are otherwise inaccessible with conventional data analytic techniques.

The key idea behind TDA Mapper is as follows: Let T be a total number of observed trees and $\{\vec{e}_t\}_{t=1}^{T} \in \mathcal{R}^{D'}$ be a data cloud of tree encodings. For our dataset, $D' = 25$. We employ the t-distributed stochastic neighbor embedding (t-SNE) [14] as a lens to reduce the data into a two-dimensional space. The t-SNE converts similarities between data points to joint probabilities and minimize the Kullback-Leibler divergence between the joint probabilities of the low-dimensional embedding and the high-dimensional data. Next, we select a function $\xi : \{\vec{e}_t\}_{t=1}^{T} \rightarrow \mathbb{R}$ that filters data in one of the two dimensions.

Let I be the range of ξ, that is, $I = [m, M] \in \mathbb{R}$, where $m = \min \xi(\vec{e}_t)$ and $M = \max \xi(\vec{e}_t)$ in the dimension d'. We place data into overlapping bins by dividing the range I into a set S of smaller overlapping intervals of uniform length and let $u_j = \{t : \xi(\vec{e}_t) \in I_j\}$ be trees corresponding to features in the interval $I_j \in S$. For each u_j we perform a k-means clustering to form clusters $\{t_{jk}\}$.

We analyze the empirical distribution of edge lengths where each cluster is merged to find the number of clusters. The merging criteria are based on the rationale that internal

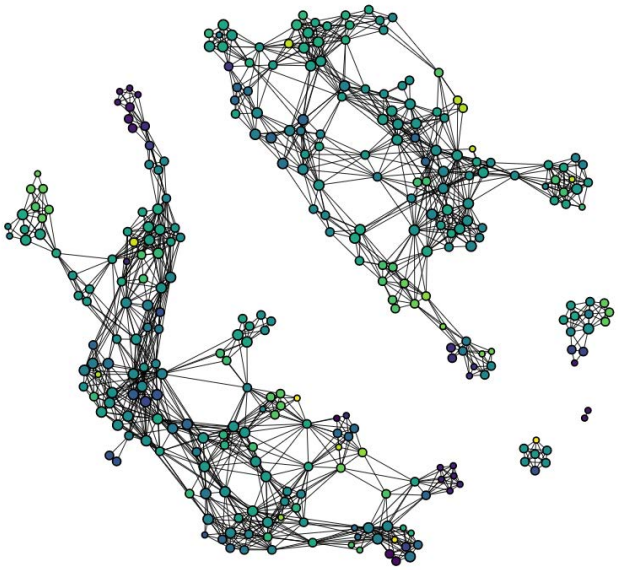

FIGURE 4. Mapper network of the Adult dataset with $cls = 5$, $n_{cubes} = 10$ and $overlap = 0.6$. Each node is a soft cluster of decision trees from the Vanilla Forest (i.e., a tree can belong to multiple clusters). Edges indicate shared decision trees between two clusters.

distances (i.e., within a cluster) are expected to be lower than external distances (i.e., in-between clusters), and distributions of internal and external distances are disjoint. Let $\{t_{jk}\}$ denote the k-th cluster of the j-th interval. We construct a cluster graph by transforming each cluster into a node and adding an edge between two nodes k and p if clusters $\{t_{jk}\}$ and $\{t_{lp}\}$ contain overlapping data points, i.e., $\{t_{jk}\} \cap \{t_{lp}\} \neq \emptyset$. Formally, the graph is called a TDA Mapper graph or a topological network.

After the graph transformation of decision tree clusters, TDA Mapper produces a low dimensional representation of the underlying data structure in the form of ''cluster tree'' graph $\mathcal{CT}$ where each ''cluster'' is a branch of some single connected component rather than a disconnected component on its own as in conventional clustering analysis. In Fig. 4, we show an example of ''cluster tree'' graph that is constructed from Adult Dataset [58].

In the underlying mapper library, we can control the bin count with the n_{cubes}, and interval overlap with the *overlap* parameters. A high *overlap* value creates more edges between vertices, whereas higher n_{cubes} and k values create more vertices in the graph. As explained in Section III-E, Topological Forest allows fine-tuning the inference process on the mapper network with various selection strategies that consider vertex sizes and edges. We consistently report good performance results across various datasets in our experiments with appropriate mapper parameters.

E. ENSEMBLE SELECTION

As the previous section III-D explained, the TDA mapper network contains vertices that are clusters of decision trees and edges that show the relationship between neighbor clusters. This type of mapper network encodes topological insights

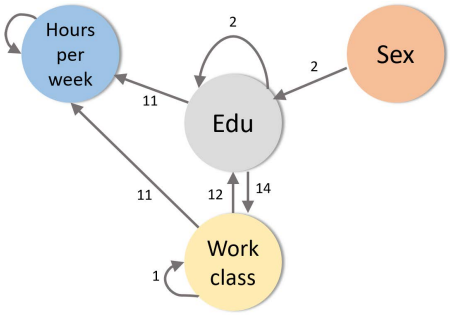

FIGURE 5. Sample graph representation of a single decision tree from the Adult dataset. Edge labels are the number of parent/child splits in a single decision tree. Higher numbers on edges show more parent/child split from the source feature to the destination feature on the tree.

into the trained decision trees. In particular, the network shape and the positions of vertices on the network convey helpful information about the diversity and quality of decision trees. For example, consider the disconnected components of the mapper graph in Figure 4. Although we use a high overlap value of 0.6, which increases the probability of establishing an edge between two clusters, three groups of vertices (i.e., clusters) at the bottom right have no edges to the rest of the network; they form disconnected components. This phenomenon is inherently due to the dissimilarity of some decision trees and their encodings with respect to the rest of the decision trees. However, this observation of dissimilarity does not tell us anything about the utility of such isolated clusters and their decision trees. In one (and most likely) scenario, trees of the isolated vertices may have been built on useless features or noisy data points that add no predictive power to the forest. In a second scenario, the isolated trees may have been built on the most predictive features and data points. We design topological cluster selection strategies to test such hypotheses and evaluate the predictive power of clusters.

We will use Figure 6 to explain three selection strategies: random, greedy mapper, and quality. In all three strategies, we first compute a cluster graph $\mathcal{CT}$ with a set of clusters defined as vertices on the graph. For simplicity, we will use a graph notation and refer to a cluster k as $v_k \in \mathcal{CT}$.

1) RANDOM

We randomly select n clusters and build a Random Forest from the union of the trees of the chosen clusters.

2) GREEDY MAPPER

We select n clusters that yield the highest AUC individually and create an ensemble from the union of the decision trees. We build an ensemble from the trees of each cluster and test the predictive power of the ensemble on validation data. Next, we test the ensemble on out-of-bag test data.

3) QUALITY

The Quality Strategy uses a homogeneity metric-based selection which we define as the average tree agreements over

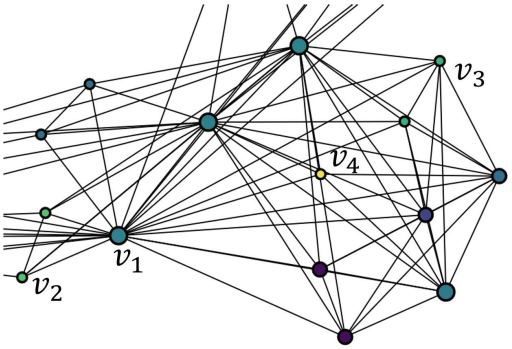

FIGURE 6. Inset of the mapper network shown in Figure 4 (bottom right). Vertex size is proportional to the number of decision trees in the cluster. Vertices v_2, v_3 and v_4 each contain a single decision tree. v_1 contains 13 decision trees. Vertex colors are trivial artifacts of the Mapper library (i.e., binned mean ids of the decision trees).

correct and incorrect predictions for all validation data points. If most trees agree on a label, homogeneity will be high. However, the trees may agree on true (correctly predicted) and false (incorrectly predicted) labels. Clusters whose trees are homogeneous on true labels are preferable to those of false labels. In both cases, we hypothesize that if decision trees of a cluster contradict each other in classification (i.e., low homogeneity), the cluster must have been formed out of "bad trees".

We split the set of data points in $X = \{X_i \cup X_f \cup X_t\}$ w.r.t. the random forest classification. X_t comprises data points classified correctly by the random forest by majority voting, whereas X_f comprises data points that are misclassified by random forest by majority voting. Lastly, X_i includes points where the decision trees vote equally for true and false labels. Formally, we define true homogeneity as follows:

Definition 2: (True homogeneity of cluster v_k where h is the number of decisions trees in cluster v_k that correctly classify data point x)

$$H_{True} = \frac{1}{|X_t|} \sum_{x \in X_t} \frac{|h \in v_k \text{ s.t } h(x, \Theta) = y|}{|v_k|}$$

Similarly, we define false homogeneity as follows:

Definition 3: (False homogeneity of cluster v_k where h is the number of decisions trees in cluster v_k that miss classified data point x)

$$H_{False} = \frac{1}{|X_f|} \sum_{x \in X_f} \frac{|h \in v_k \text{ s.t } h(x, \Theta) \neq y|}{|v_k|}$$

We calculate a cluster quality score for all existing clusters based on the true and false homogeneity scores as follows:

Definition 4 (Cluster Quality Index):

$$CQI = w_{True} \times H_{True} - w_{False} \times H_{False},$$

where

$$w_{True} = \frac{|X_t|}{|X|}, \quad w_{False} = \frac{|X_f|}{|X|}$$

Example 1 (Homogeneity): Consider the two topological clusters shown in Figure 7. Cluster 1 contains six decision trees, where four of them correctly classify data point 1. Cluster 2 contains four decision trees, and only one of them misclassifies data point 1. Similarly, only one decision tree of cluster 1 misclassifies data point 2, whereas three decision trees of cluster 2 misclassify data point 2.

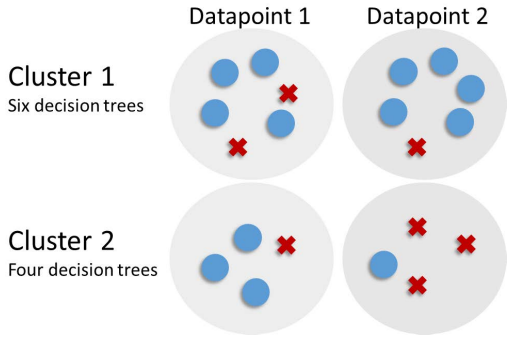

FIGURE 7. Decision tree votes of two clusters for two data points. Green circles are correct, and red crosses are incorrect classifications.

In true homogeneity computations, we will use data points 1 and 2 for cluster 1, but only datapoint 1 for cluster 2. Datapoint 2 is misclassified by the majority of cluster 2 decision trees. As a result, datapoint 2 will be used to compute the false homogeneity of cluster 2.

For cluster 1, $H_{False} = 0$ *and* $H_{True} = \frac{1}{2}(4/6+5/6) = 3/4$.

For cluster 2, $H_{False} = \frac{1}{1}(3/4) = 3/4$ *and* $H_{True} = \frac{1}{1}(3/4) = 3/4$.

We compute the cluster quality index for cluster 1 as $CQI_1 = (2/2) \times 3/4 - (0/2) \times 0 = 0.75$ *whereas* $CQI_2 = (1/2) \times 3/4 - (1/2) \times 3/4 = 0.0$.

As a result, our quality metric favors cluster 1 in its ensemble selection.

Homogeneity does not punish nor reward Tie cases where decision trees vote equally for correct and incorrect labels. In Tie cases the number of miss-classifying decision trees is equal to number of correctly classifying decision trees in a cluster. After indexing all clusters, we select the highest quality clusters and create an ensemble out of their trees.

Quality Top-x: Differing from the Quality approach, after selecting Top-K clusters based on the index score, we limit tree selection to Top-1, Top-2, or Top-5 best trees based on the tree score index in each cluster. For calculating the tree score index inside each cluster, we count the number of positive and negative collaborations of each tree. If a tree contributes to true homogeneity, it is rewarded with $+1$, and if it contributes to false homogeneity, it is penalized with -1. The top 1, 2, and 5 trees with the highest rewards are selected for an ensemble.

IV. EXPERIMENTAL RESULTS

We have released our source code at https://github.com/cakcora/MultiverseJ where we have developed a complete classification Random Forest in Java.

A. DATASETS

For the experiments, we selected six classification datasets from the UCI Machine Learning Repository with two

TABLE 1. Dataset characteristics.

Dataset	Instances	Attributes	Majority Class
Diabetes [59]	100K	21	78%
Binary Connect-4 [59]	68K	42	65%
Adult [58]	33K	14	76%
Binary Poker [60]	25K	10	99%
Binary Letter Recognition [61]	20K	16	80%
Nursery [62]	13K	6	50%

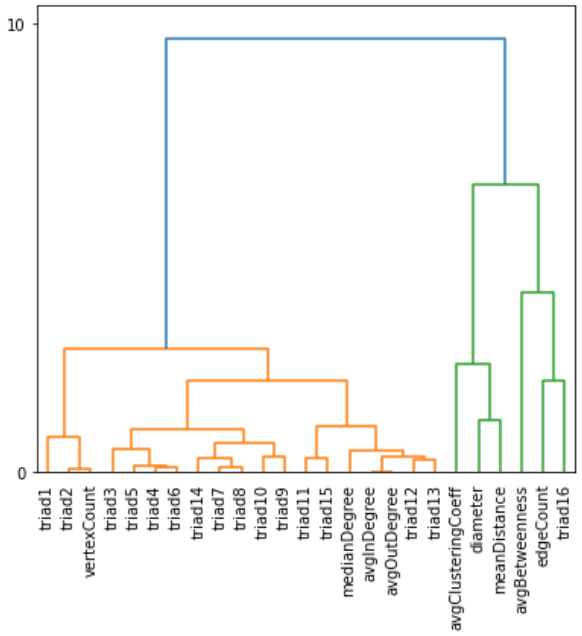

FIGURE 8. Hierarchical clustering of the 25 features used in graph encodings on the Adult dataset. The y-axis values are distances between features. Triad 16 co-clusters with edge count, but other triads mostly cluster together. Furthermore, the distances are small (e.g., between triads 12 and 13) compared to the distance between edgeCount and triad 16.

selection criteria to ensure robust classification results: the number of data points in a dataset must be more than 10K, and the dataset should have six or more features. As Table 1 shows, the selected datasets have diversified population sizes, attributes, and majority-class percentages. Diabetes, Adult, and Nursery data sets have binary labels while other datasets have more than two classes.

Since our focus in this paper is binary classification tasks, we reduce the number of classes for non-binary datasets in the following way. For the Poker dataset *nothing in hand*, *one pair*, *two pair* and *three of a kind* classes were re-labeled as *low probability to win*. Other classes were classified as *high probability to win*. In the Connect-4 dataset, *draw* and *loss* were merged in *not-win class* along with the existing *win* class. In the Letter Recognition dataset, the first 13 letters were merged in a 0 class, and the last 13 letters were merged in a 1 class.

B. FEATURES AND GRAPH ENCODINGS

During the training phase (See Section III-C), we extract a graph from each decision tree of the vanilla Random Forest

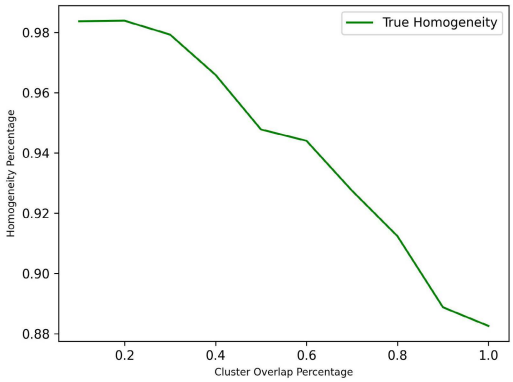

FIGURE 9. Homogeneity in the adult dataset with a changing cluster overlap percentage (*perc_overlap*).

for all data sets. Next, we extract features from each graph and use the TDA Mapper to create a topological network.

We use hierarchical clustering to show the similarity of features over trees. Although hierarchical clustering is not used by proposed approach in the paper, we use graph features extracted from each decision tree and pass them to the TDA mapper. If these features are very correlated and clustered together in the feature space, this will reduce the efficiency of TDA mapper step as it process these features as an input. Therefore, visualization of this clustering information is very critical from experimental results perspective.

In Figure 8 we show the relationship between the features where we hierarchically cluster the extracted features from graphs. Triads are the directed 3-vertex motifs [53], and they co-cluster well, except for triad 16, which is the strongest (closed) triangle motif. Unsurprisingly the median, average in (avgIn), and average out-degree (avgOut) co-cluster with the triads 12 and 13, which are closed triangle motifs. However, these degree-statistics-based features do not co-cluster with betweenness or edge count. This behavior arises because we one-hot encode categorical features of the data, creating many vertices (i.e., new features) in the graph. Such graphs have many edges, but they exhibit low clustering coefficients and few connected triads.

C. TUNING FOR TDA MAPPER

We have experimented with a set of TDA Mapper parameters for each dataset and reached the best overall AUC performance for $n\_cubes = 10$, $perc\_overlap = 0.6$ and $number\_of\_clusters = 5$. The performance is not sensitive to the *number_of_clusters* within a cube or *num_cubes*. However, *perc_overlap* determines how connected the topological network will be. A more connected network implies more shared decision trees between clusters. As a result, clusters include less similar decision trees that create lower classification homogeneity (i.e., trees vote differently on data points).

Figure 9 shows that our methods are robust against the overlap percentage. An increasing overlap percentage lowers homogeneity, but the decrease is not drastic (from 0.99 to 0.89), which shows that features of decision trees

TABLE 2. AUC of different methods using 10% of trees. Greedy Mapper AUC is, in average, 98% of the Vanilla Forest AUC.

Dataset	Greedy Mapper	Quality	Q-Top1	Q-Top2	Q-Top5	Random	Vanilla Forest
Diabetes	**0.63 ± 0.008**	0.56 ± 0.01	0.56 ± 0.01	0.56 ± 0.01	0.56 ± 0.01	0.59 ± 0.02	0.65 ± 0.006
Binary Connect-4	**0.81 ± 0.01**	0.79 ± 0.01	0.78 ± 0.01	0.80 ± 0.01	0.79 ± 0.01	0.77 ± 0.02	0.85 ± 0.005
Adult	**0.89 ± 0.005**	0.88 ± 0.005	0.88 ± 0.005	0.88 ± 0.006	0.88 ± 0.005	0.87 ± 0.01	0.90 ± 0.004
Binary Poker	**0.76 ± 0.1**	0.54 ± 0.15	0.53 ± 0.13	0.53 ± 0.16	0.52 ± 0.15	0.41 ± 0.17	0.77 ± 0.099
Binary Letter Recognition	0.94 ± 0.008	0.94 ± 0.009	0.95 ± 0.009	**0.95 ± 0.008**	0.94 ± 0.01	0.92 ± 0.02	0.95 ± 0.006
Nursery	0.53 ± 0.01	0.52 ± 0.01	**0.54 ± 0.01**	0.53 ± 0.01	0.52 ± 0.01	0.47 ± 0.01	0.48 ± 0.015

are diversified enough to create separate clusters even when we allow for a higher overlap. As a result, trees in a cluster are similar and vote similarly. The high homogeneity is a significant result because, as we show in Section IV-D it will enable us to use fewer decision trees from a cluster but reach similar performance in classification.

D. CLASSIFICATION RESULTS

Our experiments partition each dataset into 80% training, 10% validation, and 10% out-of-bag test subsets. We ran each experiment 30 times, where we used random seeds to select the partitions in each replica. A vanilla forest has been created on the training subset in each replica. The vanilla forest has been tested for any possible overfitting by comparing the overall accuracy of the train and test set, which were close to each other.

In the next step, we employ the Kepler Mapper [63], a Python implementation of TDA Mapper, with built-in visualization, dimensionality reduction, and clustering options on the vanilla forest decision trees to create and visualize our TDA results.

We report our results in terms of ROC AUC and run-time performance. We compared six ensemble selection strategies with vanilla random forest. The definition of each strategy is explained in Section III-E. Table 2 shows the mean and standard deviation of AUC results over the 30 replicas. As Table 2 shows, we find that Greedy Mapper has the best AUC on four datasets among all ensemble selection strategies. Greedy Mapper also loses only 2% of AUC with significantly smaller (10x, 30 trees) Topological Forest size compared to the vanilla Random Forest, which has 300 decision trees. In the Poker and Letter Recognition datasets, where we have substantial class imbalances (99% and 80%, respectively), Greedy Mapper and Q Top5 AUC are close to the Vanilla Forest in terms of AUC, which offers evidence that selecting the best clusters can also help with class imbalance.

We also reported the prediction quality of three random forest policies that have 5%, 10% and 100% of decision trees as proxy for all approaches that tune the number of decision trees. Table 3 shows the results for randomly picking up decision trees. Since topological forest has 10% of decision trees, comparing these three policies gives an idea of the upper and lower bound on prediction quality when number of trees is purely tuned. From these new experimental results, we find that the quality and diversity of decision trees are very important factors in further improving the prediction quality of random forest. As shown in Table 3, the Greedy Mapper approach produces random forest with 10% of decision trees, which is better than random policy that uses 10% of decision trees.

TABLE 3. AUC for random tree selection policies.

Dataset	5% of trees	10% of trees	100% of trees
Diabetes	0.58 ± 0.007	0.60 ± 0.005	0.65 ± 0.006
Binary Connect-4	0.76 ± 0.005	0.79 ± 0.007	0.85 ± 0.005
Adult	0.87 ± 0.005	0.88 ± 0.006	0.90 ± 0.004
Binary Poker	0.40 ± 0.144	0.55 ± 0.165	0.77 ± 0.099
Binary Letter Recog.	0.91 ± 0.006	0.93 ± 0.009	0.95 ± 0.006
Nursery	0.45 ± 0.014	0.46 ± 0.015	0.48 ± 0.015

Table 4 shows the computational cost of the inference task from our best approach, Greedy Mapper, and the Vanilla Random Forest when deployed on the test data. Here, the computational cost during the inference task is defined as the total CPU time spent for all computations, including the cost of inference from individual decision trees. In the best case, Greedy Mapper improves computational cost 217 times compared to Vanilla Forest in the Adult dataset, whereas the improvement is around 22 times for the Binary Poker dataset. On average, Greedy Mapper reduces the inferring time on the test data with such high values for two reasons. First, the number of decision trees in the ensemble is reduced by 90% or more after applying ensemble selection strategies. Second, trees that are built on *low quality* features grow too complex to fit the training data better. However, such deeper trees are unlikely to appear in the best-performing clusters. As a result, such trees are excluded in Greedy Mapper, contributing to better run-time results.

We also compared the computational cost of all approaches for the training task in Table 5. Here, the computational cost during the training task is defined as the total CPU time spent for all computations, including training individual decision trees and any other downstream steps like graph encoding of decisions trees or running the TDA Mapper. While, on average, Greedy Mapper is 4.1 times slower than Vanilla Forest for the training task, it is 113 times faster than Vanilla Forest in the inference task. Thus, the trade-off between the cost of training and the computational performance in the inference task significantly justifies the use of Topological Forest.

The efficiency of the Topological Forest is mainly related to the computational time savings in inferral. Topological Forest uses 10% of the trees and yields comparable performance to Vanilla Forest. Furthermore, Topological Forest performs better than the Vanilla forest in Nursery and Letter

TABLE 4. Computational cost for inference (millisecond).

Dataset	Vanilla Forest	Greedy Mapper	Speed Up Ratio
Diabetes	842.16 ± 146.34	4.76 ± 3.85	177X
Binary Connect-4	382.33 ± 63.60	2.53 ± 1.07	112X
Adult	294.53 ± 56.38	1.36 ± 2.98	217X
Binary Poker	36.76 ± 2.16	1.68 ± 0.47	22X
Binary Letter Recog.	145.10 ± 7.00	1.14 ± 0.69	127X
Nursery	44.03 ± 9.27	1.73 ± 0.63	25X

TABLE 5. Computational cost for training (second).

Dataset	Vanilla Forest	Greedy Mapper	Slowness Ratio
Diabetes	7.51 ± 0.41	18.49 ± 0.39	2.46X
Binary Connect-4	1.61 ± 0.08	7.37 ± 0.22	4.57X
Adult	5.04 ± 0.53	14.01 ± 0.45	2.77X
Binary Poker	1.38 ± 0.13	6.11 ± 0.22	4.42X
Binary Letter Recog.	4.40 ± 1.11	11.73 ± 0.88	2.66X
Nursery	0.58 ± 0.09	4.48 ± 0.11	7.72X

Recognition datasets (Table 2). In this sense, our method has better AUC performance for some datasets as well.

V. CONCLUSION AND FUTURE WORK

We have developed an open-source implementation of our novel ML method Topological Forest. Our approach builds on a Vanilla Random Forest implementation but uses topological methods to create a refined ensemble that has a smaller number of decision trees and better trees in the forest. On average, Topological Forest speeds up inference time by more than 100x for a cost of at most 2% reduction in AUC. The results of our experiments suggest that the topological forest is considerably faster than random forest. Moreover it needs less resources and efforts compared to neural networks.

As a future work, the capabilities of topological forest in different machine learning tasks will be a good area of research. We will use the topological forest in our future research to address the distribution shift problem by developing more diverse random forest.

REFERENCES

[1] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[2] C. Iwendi, A. K. Bashir, A. Peshkar, R. Sujatha, J. M. Chatterjee, S. Pasupuleti, R. Mishra, S. Pillai, and O. Jo, "COVID-19 patient health prediction using boosted random forest algorithm," *Frontiers Public Health*, vol. 8, p. 357, Jul. 2020.
[3] M. Sheykhmousa, M. Mahdianpari, H. Ghanbari, F. Mohammadimanesh, P. Ghamisi, and S. Homayouni, "Support vector machine versus random forest for remote sensing image classification: A meta-analysis and systematic review," *IEEE J. Sel. Topics Appl. Earth Observ. Remote Sens.*, vol. 13, pp. 6308–6325, 2020.
[4] S. Hamsa, I. Shahin, Y. Iraqi, and N. Werghi, "Emotion recognition from speech using wavelet packet transform cochlear filter bank and random forest classifier," *IEEE Access*, vol. 8, pp. 96994–97006, 2020.
[5] S. A. Roseline, S. Geetha, S. Kadry, and Y. Nam, "Intelligent vision-based malware detection and classification using deep random forest paradigm," *IEEE Access*, vol. 8, pp. 206303–206324, 2020.
[6] Y. Chen, W. Zheng, W. Li, and Y. Huang, "Large group activity security risk assessment and risk early warning based on random forest algorithm," *Pattern Recognit. Lett.*, vol. 144, pp. 1–5, Apr. 2021.
[7] L. Cheng, X. Chen, J. D. Vos, X. Lai, and F. Witlox, "Applying a random forest method approach to model travel mode choice behavior," *Travel Behav. Soc.*, vol. 14, pp. 1–10, Jan. 2019.
[8] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu, "LightGBM: A highly efficient gradient boosting decision tree," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–9.
[9] G. Montavon, W. Samek, and K.-R. Müller, "Methods for interpreting and understanding deep neural networks," *Digit. Signal Process.*, vol. 73, pp. 1–15, Feb. 2018.
[10] T. M. Oshiro, P. S. Perez, and J. A. Baranauskas, "How many trees in a random forest?" in *Proc. Int. Workshop Mach. Learn. Data Mining Pattern Recognit.* Cham, Switzerland: Springer, 2012, pp. 154–168.
[11] M. Liu, R. Lang, and Y. Cao, "Number of trees in random forest," *Comput. Eng. Appl.*, vol. 51, no. 5, pp. 126–131, 2015.
[12] P. Probst and A.-L. Boulesteix, "To tune or not to tune the number of trees in random forest," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 6673–6690, Jan. 2017.
[13] G. Carlsson, "Topology and data," *Bull. Amer. Math. Soc.*, vol. 46, no. 2, pp. 255–308, 2009.
[14] L. V. D. Maaten and G. Hinton, "Visualizing data using t-SNE," *J. Mach. Learn. Res.*, vol. 9, no. 11, pp. 1–27, 2008.
[15] M. S. Alam and S. T. Vuong, "Random forest classification for detecting Android malware," in *Proc. IEEE Int. Conf. Green Comput. Commun. IEEE Internet Things IEEE Cyber, Phys. Social Comput.*, Beijing, China, Aug. 2013, pp. 663–669.
[16] S. Zeng, M. A. Bayir, J. J. Pfeiffer, D. Charles, and E. Kiciman, "Causal transfer random forest: Combining logged data and randomized experiments for robust prediction," in *Proc. 14th ACM Int. Conf. Web Search Data Mining*, Mar. 2021, pp. 211–219.
[17] D. Gunning, M. Stefik, J. Choi, T. Miller, S. Stumpf, and A.-Z. Yang, "XAI—Explainable artificial intelligence," *Sci. Robot.*, vol. 4, no. 37, 2019, Art. no. eaay7120.
[18] F. K. Došilović, M. Brčić, and N. Hlupić, "Explainable artificial intelligence: A survey," in *Proc. 41st Int. Conv. Inf. Commun. Technol., Electron. Microelectron. (MIPRO)*, May 2018, pp. 210–215.
[19] O. Sagi and L. Rokach, "Explainable decision forest: Transforming a decision forest into an interpretable tree," *Inf. Fusion*, vol. 61, pp. 124–138, Sep. 2020.
[20] S. M. Lundberg, G. Erion, H. Chen, A. DeGrave, J. M. Prutkin, B. Nair, R. Katz, J. Himmelfarb, N. Bansal, and S.-I. Lee, "From local explanations to global understanding with explainable AI for trees," *Nature Mach. Intell.*, vol. 2, no. 1, pp. 56–67, Jan. 2020.
[21] D. Petkovic, R. Altman, M. Wong, and A. Vigil, "Improving the explainability of random forest classifier—User centered approach," in *Proc. Pacific Symp.* Singapore: World Scientific, 2018, pp. 204–215.
[22] G. Bakirli and D. Birant, "DTreeSim: A new approach to compute decision tree similarity using re-mining," *TURKISH J. Electr. Eng. Comput. Sci.*, vol. 25, pp. 108–125, Jan. 2017.
[23] J. Chen, K. Li, Z. Tang, K. Bilal, S. Yu, C. Weng, and A. Li, "A parallel random forest algorithm for big data in a spark cloud computing environment," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 4, pp. 919–933, Aug. 2017.
[24] Y. Xu, "Research and implementation of improved random forest algorithm based on spark," in *Proc. IEEE 2nd Int. Conf. Big Data Anal. (ICBDA)*, Mar. 2017, pp. 499–503.
[25] W. Lin, Z. Wu, L. Lin, A. Wen, and J. Li, "An ensemble random forest algorithm for insurance big data analysis," *IEEE Access*, vol. 5, pp. 16568–16575, 2017.
[26] S. D. Río, V. López, J. M. Benítez, and F. Herrera, "On the use of MapReduce for imbalanced big data using random forest," *Inf. Sci.*, vol. 285, pp. 112–137, Nov. 2014.
[27] M. A. Bayir, M. Xu, Y. Zhu, and Y. Shi, "Genie: An open box counterfactual policy estimator for optimizing sponsored search marketplace," in *Proc. 12th ACM Int. Conf. Web Search Data Mining*, Melbourne, VIC, Australia, Jan. 2019, pp. 465–473.
[28] P. Probst, M. N. Wright, and A. Boulesteix, "Hyperparameters and tuning strategies for random forest," *WIREs Data Mining Knowl. Discovery*, vol. 9, no. 3, p. e1301, May 2019.
[29] P. Latinne, O. Debeir, and C. Decaestecker, "Limiting the number of trees in random forests," in *Proc. Int. Workshop Multiple Classifier Syst.* Cham, Switzerland: Springer, 2001, pp. 178–187.
[30] C. Fefferman, S. Mitter, and H. Narayanan, "Testing the manifold hypothesis," *J. Amer. Math. Soc.*, vol. 29, no. 4, pp. 983–1049, Feb. 2016.
[31] H. Edelsbrunner, D. Letscher, and A. Zomorodian, "Topological persistence and simplification," in *Proc. 41st Annu. Symp. Found. Comput. Sci.*, 2000, pp. 454–463.

[32] F. Hensel, M. Moor, and B. Rieck, "A survey of topological machine learning methods," *Frontiers Artif. Intell.*, vol. 4, p. 52, May 2021.
[33] D. Cohen-Steiner, H. Edelsbrunner, J. Harer, and Y. Mileyko, "Lipschitz functions have L_p-stable persistence," *Found. Comput. Math.*, vol. 10, no. 2, pp. 127–139, Apr. 2010.
[34] C. Chen and H. Edelsbrunner, "Diffusion runs low on persistence fast," in *Proc. Int. Conf. Comput. Vis.*, Barcelona, Spain, Nov. 2011, pp. 423–430.
[35] N. Atienza, R. Gonzalez-Diaz, and M. Rucco, "Persistent entropy for separating topological features from noise in Vietoris-Rips complexes," *J. Intell. Inf. Syst.*, vol. 52, no. 3, pp. 637–655, Jun. 2019.
[36] B. Rieck, F. Sadlo, and H. Leitte, "Topological machine learning with persistence indicator functions," in *Topological Methods in Data Analysis and Visualization*. Cham, Switzerland: Springer, 2017, pp. 87–101.
[37] Y. Umeda, "Time series classification via topological data analysis," *Inf. Media Technol.*, vol. 12, pp. 228–239, Jan. 2017.
[38] I. Chevyrev, V. Nanda, and H. Oberhauser, "Persistence paths and signature features in topological data analysis," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 42, no. 1, pp. 192–202, Jan. 2020.
[39] P. Bubenik, "Statistical topological data analysis using persistence landscapes," *J. Mach. Learn. Res.*, vol. 16, no. 1, pp. 77–102, 2015.
[40] Q. Zhao, Z. Ye, C. Chen, and Y. Wang, "Persistence enhanced graph neural network," in *Proc. Int. Conf. Artif. Intell. Statist.*, Palermo, Italy, Aug. 2020, pp. 2896–2906.
[41] M. Moor, M. Horn, B. Rieck, and K. Borgwardt, "Topological autoencoders," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Jul. 2020, pp. 7045–7054.
[42] M. Yuvaraj, A. K. Dey, V. Lyubchich, Y. R. Gel, and H. V. Poor, "Topological clustering of multilayer networks," *Proc. Nat. Acad. Sci. USA*, vol. 118, no. 21, May 2021, Art. no. e2019994118.
[43] P. S. Ignacio, J.-A. Bulauan, and J. R. Manzanares, "A topology informed random forest classifier for ECG classification," in *Proc. Comput. Cardiol. Conf. (CinC)*, Dec. 2020, pp. 1–4.
[44] C. Chen, X. Ni, Q. Bai, and Y. Wang, "A topological regularizer for classifiers via persistent homology," in *Proc. 22nd Int. Conf. Artif. Intell. Statist.*, Naha, Okinawa, Japan, 2019, pp. 2573–2582.
[45] C. Hofer, F. Graf, M. Niethammer, and R. Kwitt, "Topologically densified distributions," in *Proc. Int. Conf. Mach. Learn. (ICML)*, Jul. 2020, pp. 4304–4313.
[46] P. Hage and F. Harary, "Eccentricity and centrality in networks," *Social Netw.*, vol. 17, no. 1, pp. 57–63, Jan. 1995.
[47] P. D. Meo, E. Ferrara, G. Fiumara, and A. Ricciardello, "A novel measure of edge centrality in social networks," *Knowl.-Based Syst.*, vol. 30, pp. 136–150, Jun. 2012.
[48] U. Brandes, D. Delling, M. Gaertler, R. Görke, M. Hoefer, Z. Nikoloski, and D. Wagner, "On finding graph clusterings with maximum modularity," in *Proc. Int. Workshop Graph-Theoretic Concepts Comput. Sci.* Cham, Switzerland: Springer, 2007, pp. 121–132.
[49] T. Schank and D. Wagner, "Approximating clustering coefficient and transitivity," *J. Graph Algorithms Appl.*, vol. 9, no. 2, pp. 265–275, 2005.
[50] N. M. Kriege, F. D. Johansson, and C. Morris, "A survey on graph kernels," *Appl. Netw. Sci.*, vol. 5, no. 1, pp. 1–42, Dec. 2020.
[51] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," 2017, *arXiv:1709.05584*.
[52] F. Errica, M. Podda, D. Bacciu, and A. Micheli, "A fair comparison of graph neural networks for graph classification," 2019, *arXiv:1912.09893*.
[53] V. Batagelj and A. Mrvar, "A subquadratic triad census algorithm for large sparse networks with small maximum degree," *Social Netw.*, vol. 23, no. 3, pp. 237–243, Jul. 2001.
[54] J. M. Kleinberg, "Authoritative sources in a hyperlinked environment," in *Proc. SODA*, vol. 98, 1998, pp. 668–677.
[55] U. Brandes, "A faster algorithm for betweenness centrality," *J. Math. Sociol.*, vol. 25, no. 2, pp. 163–177, 2001.
[56] J. O'Madadhain, D. Fisher, P. Smyth, S. White, and Y.-B. Boey, "Analysis and visualization of network data using JUNG," *J. Stat. Softw.*, vol. 10, no. 2, pp. 1–35, 2005.
[57] G. Singh, F. Memoli, and G. Carlsson, "Topological methods for the analysis of high dimensional data sets and 3D object recognition," in *Proc. Eurograp. Symp. Point-Based Graph.*, M. Botsch, R. Pajarola, B. Chen, and M. Zwicker, 2007, pp. 1–10.
[58] R. Kohavi, "Scaling up the accuracy of Naive–Bayes classifiers: A decision-tree hybrid," in *Proc. 2nd Int. Conf. Knowl. Discovery Data Mining*, vol. 96, 1996, pp. 202–207.
[59] D. Dua and C. Graff, "UCI machine learning repository," Univ. California, Irvine, CA, USA, Tech. Rep., 2017.
[60] R. Cattral, F. Oppacher, and D. Deugo, "Evolutionary data mining with automatic rule generalization," *Recent Adv. Comput., Comput. Commun.*, vol. 1, no. 1, pp. 296–300, 2002.
[61] P. W. Frey and D. J. Slate, "Letter recognition using Holland-style adaptive classifiers," *Mach. Learn.*, vol. 6, no. 2, pp. 161–182, 1991.
[62] M. Olave, V. Rajkovic, and M. Bohanec, "An application for admission in public school systems," *Expert Syst. Public Admin.*, vol. 1, pp. 145–160, Jan. 1989.
[63] H. van Veen, N. Saul, D. Eargle, and S. Mangham, "Kepler mapper: A flexible Python implementation of the mapper algorithm," *J. Open Source Softw.*, vol. 4, no. 42, p. 1315, Oct. 2019.

MURAT ALI BAYIR (Member, IEEE) received the Ph.D. degree from the Department of Computer Science, University at Buffalo, in 2010. He is currently leading the Counterfactual and Offline Evaluation Team, Ads Division, Meta Platforms Inc., Before Meta Ads, he spent seven years at Bing Ads, as a Principal Researcher, and three years at Google Inc., as a Software Engineer. He has published more than 30 papers in international conferences and journals, including WWW, WSDM, WISE, *WWW* journal, and *Data & Knowledge Engineering*. His research interests include machine learning, data mining, and optimization algorithms.

KIARASH SHAMSI received the B.S. degree from the Iran University of Science and Technology, Tehran, Iran, and the M.S. degree from the University of Science and Culture, Tehran. He is currently pursuing the Ph.D. degree in computer science with the University of Manitoba, Canada. His research interests include data science, machine learning, and blockchain technologies.

HUSEYINCAN KAYNAK received the B.S. degree from Hacettepe University, Ankara, Turkey, in 2020. He is currently a Software Engineer with MilSOFT Software Technologies, Ankara. His research interests include machine learning and optimization algorithms. He also worked as a Student Representative of ACM Student Chapter, Hacettepe University, during his B.S. study.

CUNEYT GURCAN AKCORA (Member, IEEE) received the M.S. degree from University at Buffalo, Buffalo, NY, USA, and the Ph.D. degree from the University of Insubria, Italy. He is currently an Assistant Professor of computer science and statistics with the University of Manitoba, Canada. He has published in leading conferences and journals, including IEEE Transactions, KDD, *The VLDB Journal*, ICDM, SDM, IJCAI, and ICDE. His research interests include data science on complex networks and large-scale graph analysis, with applications in social, biological, the IoT, and blockchain networks. He was a recipient of Fulbright Scholarship.