Learning to Explain Voting Rules

Inwon Kang
Rensselaer Polytechnic Institute
Troy, New York, USA
inwon.kang04@gmail.com

Qishen Han
Rensselaer Polytechnic Institute
Troy, New York, USA
hnickc2017@gmail.com

Lirong Xia
Rensselaer Polytechnic Institute
Troy, New York, USA
xialirong@gmail.com

ABSTRACT

Explaining the outcome of a vote is a crucial task to address, especially in the case of complex voting rules. This work proposes a methodology for explaining voting rules using decision-tree-based classifiers, a widely applied tool in XAI domain. Using simple ranking-aware features, the classifiers can be trained to achieve high accuracy while maintaining a human-readable size. We test this framework with well-established voting rules – Copeland, Kemeny-Young, Ranked Pairs, and Schulze – and different decision tree algorithms to generate explanations for each election's outcome. We find that in the setting of three candidates, a decision tree model can perfectly explain these voting rules with simple features. When the number of candidates increases, the tree is not perfect yet remains high accuracy in explaining the outcome. This works aims to improve the public's engagement with election systems by improving interpretability in voting rules with a visual aid which is faithful to the original voting rule.

CCS CONCEPTS

• **Computing methodologies → Classification and regression trees**; **Multi-agent systems**; • **Human-centered computing → Human computer interaction (HCI)**.

KEYWORDS

Social Choice, Artificial Intelligence, Explainable Voting, Machine Learning, Explainable AI, XAI

ACM Reference Format:
Inwon Kang, Qishen Han, and Lirong Xia. 2023. Learning to Explain Voting Rules. In *Proceedings of 6th AAAI/ACM Conference on AI, Ethics, and Society (AIES '23)*. ACM, New York, NY, USA, 11 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

Voting is the most widely applied decision making method with application in every corner of life, ranging from electing the government and congress to making high-stake decisions by multiple supercomputers, and has also become an essential part of many complex machine learning systems. While voting aims to reach a "good" decision, it is also important to *explain* and convince the people on the decision. The requirement and challenges of explaining the vote outcome is especially acute for complex voting rules such as Schulze [23], as it's hard for untrained individuals to understand the technical details and insights of voting rules.

Previous works focuses on logical explanation of the voting rule, which aims to generate interpretation on logical principles and reasoning, including axiomatic approach [6, 19], algorithmic approach, and crowdsourcing [24]. However, it is important to provide a *procedural explanation* in practical scenarios, which allows agents without expertise to examine the votes in detail and verifying the correctness of the outcome. Unlike logical explanation that focus on interpretation, procedural explanation emphasize the trustworthiness of the voting. Trustworthy is an essential guarantee in large elections, where every detail may be questioned and examined by a large population of crowd. Therefore, it is important to provide a human-readable procedural explanation via simple features, for example, the position of a candidate in the rank and the competition between two candidates.

Example 1.1. A local government decides to hold votes on public construction in the neighborhood (e.g., swimming pools, stadium, e.t.c.) with some pre-determined voting rule (e.g., Schulze). To get better support from the public, the government decides to generate an explanation for the voting outcome for examination. The public has no expertise on how the voting rule works, but they can understand simple notions such as "a swimming pool is more welcomed than a stadium" and "a park is a second-favorite choice for half of the voters". The government hopes that their explanation is as simple and correct as possible. How can they generate the explanation?

Motivated by the example, we propose the following question.

Can we generate procedural explanations for voting rules via simple features?

Traditionally, a procedural explanation appears in a form of a pseudo-code or the execution record of the voting rule. This works on simple, intuitive rules such as Plurality and Borda, but is likely to fail on more complex rules. For example, ranked pairs and and Schulze compute the winner on highly abstract features (pair rankings and paths) which might be alien for people not familar with social choice theory. The development of machine learning provides a powerful perspective and tools for analyzing voting rules. Multiple works have demonstrated that deep models such as MLP or GNN can be used to estimate and emulate the behavior of a voting rule [3, 5, 16, 20]. However, deep models can not gurantee a correct representation and are equally or even more complex than the voting rule itself. The methods still need to improve in terms of accuracy and interpretability.

This paper focuses on explaining voting rules via *decision tree models*. Decision tree is s powerful tool to learn and characterize a voting rule via a series of simple features. Moreover, as a white-box

AIES '23, August 9-11th 2023, Woodstock, NY
© 2023 Association for Computing Machinery.
ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/XXXXXXX.XXXXXXX

learning model, a decision tree has a human-readable structure, which allows non-experts to follow the steps in a visual manner. Compared to pseudo codes, a decision tree built upon pairwise comparisons is easier to follow and verify the correctness of the vote. By training a *perfect* decision tree that can accurately predict the winner of an election, one can use the tree's decision paths to justify the outcome of the election.

Our Contribution. We propose a methodology of producing a procedural explanation for a voting rule using decision tree models. In our experiments, we train decision tree models to learn different voting rules. We use *pairwise margin*, i.e. the difference between pairwise comparisons, to characterize a voting rule. A typical pairwise margin looks like "whether the votes of $A > B$ exceeds the votes of $B > C$". In a decision tree, a non-leaf node is a condition of feature, a leaf node is a decision, and a path is procedural explanation of a profile.

We show that decision trees for voting rules that satisfy the Condorcet criterion can be learned for a setting with 3 candidates. In particular, we show that the Generalized Optimal Sparse Decision Tree (GOSDT) [15] and Extreme Gradient Boosting (XGBoost) [7] algorithms output a perfect tree when using the pairwise-margin feature. We also test this framework on a more complex setting with increased candidates, and find that the algorithms were still able to find a near-perfect tree with most of the leaf nodes being correct.

We believe a decision tree serves as a useful tool to explain learning. It is easier to understand than pseudo codes and more transparent than deep models. A decision tree that can provably cover every possible scenario can be relied on as a correct explanation. Because of the transparency and guaranteed accuracy, the tree can then be used by anyone to verify the steps taken to reach the election result. Expanding upon Example 1.1, the local government can release the trained decision tree for the voting process, allowing the public to understand how their votes impacted the result of the outcome, or even decide whether they would like to change the voting rule to a different rule.

2 RELATED WORKS

Explainable Voting. Past literature has approached the problem of explaining a voting rule using the axiomatic approach first proposed by Cailloux and Endriss [6], which decomposes the voting profile into subgroups that fit into some pre-define axioms acknowledged by the public. The voting outcome is then interpreted as a logical chain of these axioms. Proccacia [19] proposed that axiomatic properties should help explain the voting rule to the public in real-life scenarios. Expanding upon this, Peters et al. [18] show that an axiom-based explanation for Borda can be constructed in $O(m^2)$ steps for m alternatives.

Apart from the axiomatic approach, Suryanarayana et al. [24] uses feature-based explanations to explain the outcome of elections. The authors gathered participants through crowdsourcing and asked them to evaluate an election's result based on feature-based explanations.

Machine Learning and Voting Rules. Past works on using machine learning to learn from voting rules have been mostly focused on using models based on neural network architecture. Burka et al. [5] train a Multi Layer Perceptron (MLP) model to learn various voting rules. They show that MLP performs near perfect for the Borda rule, given a sufficient sample size. Anil et al. [3] show Permutation Invariant Neural Nets (PIN) can learn generalized voting rules very accurately compared to models like MLP, as well as demonstrating that new voting rules that consider various (utilitarian, egalitarian) social welfare functions perform better than classical NN models. Machine learning can also be used in conjunction with voting rules. Doucette et al. [9] propose a method using conventional machine learning to complete a preference profile with incomplete ballots to be used with traditional voting rules. In this work, machine learning is used to 'complete' the incomplete ballots based on the patterns extracted from existing ballots. Xia et al. [25] discuss a top-down approach of designing a voting rule using machine learning. The authors discuss the goals of building such a system, and how to design a training workflow to train a machine learning model to output a voting rule with desired properties. Building upon this idea, Grant et al. [10] propose a concept of direct democracy system through *voting avatars*, in which each individual is represented by an automated agent based on a customized machine-learning model.

Explainability of Machine Learning Models. Also known as XAI, past works in explainable AI have proposed different models and methodologies to increase machine learning models' explainability. Gunning et al. [12] discuss the various ML models that can be used to generate explanations, comparing the complexity versus the explainability of various broad types of machine learning. Samek et al. [22] conduct a survey on existing methods of explaining AI models and the scope of explanation. The authors discuss various methods for explaining black-box models, such as using decision trees or logistic regression weights to generate an explanation for the target model. Dat et al. [8] reviews AI explanation methods and categorizes them according to their similarities and differences. The authors focus on explaining deep models and point out that there are still limitations to how well such models can be explained. Islam et al. [13] conduct a systematic literature review of XAI by reviewing 137 recent publications relating to XAI. They find a wide variety of application domains such as entertainment, finance, criminal justice and healthcare. Adadi et al. [1] conduct a survey of the XAI field and identify the different types of motivations and application domains associated with it. In their survey, the authors outline the need for XAI as follows: the need for justification, controllability, and discoverability. The authors also point out a wide range of domains that could benefit from XAI, such as healthcare, finance, and criminal justice. Guidotti et al. [11] define an explanation for AI as an "interface between humans and a decision maker that is at the same time both an accurate proxy of the decision maker and comprehensible to humans". Expanding upon this definition, Arrieta et al. [4] define *explainability* as "details and reasons a model gives to make its functioning clear or easy to understand". In this work, we use a combination of these definitions. For a decision tree to be 'explainable' and thus serve as an explanation, *the model needs to be an accurate representation of the target mechanism while providing a clear insight into its decision paths given a weighted majority graph.*

3 PRELIMINARIES

3.1 Voting Rules

A voting rule chooses a set of winners from a set of candidates according to a voting profile. A set of candidates is defined as C, with $|C| = m$. A vote V is a linear ordering of these candidates, namely $V \in L(C)$ and $|V| = n$, where $L(C)$ denotes all the possible orderings. A voting profile P is composed of n votes, thus $P \in L(C)^n$. Throughout this work, we use m to denote the number of candidates and n to denote the size of the profile. A voting rule $r : L(C)^n \rightarrow 2^C \setminus \{\emptyset\}$ maps a profile to a set of winners from the candidates. A weighted majority graph (WMG) of a profile P is a graph structure that characterizes pairwise preferences between agents. Let $|A \succ B|$ be the number of votes that prefer A over B in the profile P. For each $A \succ B$ in profile P, the WMG has an edge $A \rightarrow B$ with weight $|A \succ B| - |B \succ A|$.

3.1.1 Copeland. The Copeland rule selects the winner by counting pairwise preferences. A candidate gets a Copeland score of 1 from each candidate that he/she is preferred over by a majority of the voters and gets α from each candidate that he/she is tied with. In this paper, we assume $\alpha = 0.5$. The total Copeland score of a candidate is the sum of the scores his/her wins from all other candidates. The final winner is declared by finding the candidate who has the highest Copeland score.

3.1.2 Kemeny-Young. The Kemeny-Young rule [14] selects the ranking that *least* disagrees with the profile, and the winner is the top candidate of the select ranking. Given a ranking and a vote, the score of the ranking from the vote is the number of their opposite pairwise preferences. The score of the ranking in the profile is the sum of the score of this ranking from all the votes. The ranking with the minimum score is then selected.

Example 3.1 (Kemeny-Young). For a voting profile with two votes for $A \succ B \succ C$ and one vote for $B \succ C \succ A$. Ranking $A \succ B \succ C$ has no opposite pairwise preferences with the first two votes and has two opposite preferences ($B \succ A$ and $C \succ A$) with vote $B \succ C \succ A$. Therefore, the score of $A \succ B \succ C$ is 2. $A \succ B \succ C$ is the winning ranking because no other ranking has a lower score.

3.1.3 Ranked pairs. The ranked pairs rule compares each pair of alternatives. Every possible pair of candidates are ranked according to their pairwise victory (the number of times a candidate beats another). The pairwise victory are then sorted in descending order (highest first). Iterating through the list of pairs, each pair is added to the final ranking graph as an edge from the winner to the loser unless it creates a cycle. The final winner is found by selecting candidates with no incoming edges.

3.1.4 Schulze. The Schulze rule finds the winner by constructing a weighted majority graph of the preferences. Given a path in the graph, the *path margin* equals to the minimum weight of any of its arcs. And the Schulze score $S(A, B)$ equals to the maximum path margin of all possible $A \rightarrow B$ paths. The Schulze winner is declared by finding a candidate who is preferred over every other candidate – in other words, a candidate A such that $S(A, B) \geq S(B, A)$ for any other candidates B.

$A \succ B$	$A \succ C$	$B \succ A$	$B \succ C$	$C \succ A$	$C \succ B$
5	7	2	5	0	2

Table 1: Example of Pairwise Victory

$\lvert A \succ B\rvert - \lvert A \succ C\rvert$	-2
$\lvert A \succ B\rvert - \lvert B \succ A\rvert$	3
$\lvert A \succ B\rvert - \lvert B \succ C\rvert$	2
$\lvert A \succ B\rvert - \lvert C \succ A\rvert$	5
...	...

Table 2: Example of Pairwise Margin

Example 3.2 (Ranked pairs and Schulze). Here we give an illustration of how ranked pairs and the Schulze rule work. There are four alternatives $\{A, B, C, D\}$ and a profile P whose weighted majority graph is shown in Figure 1. (Edge $C \rightarrow B$ weighs 4 and $A \rightarrow D$ weighs 2.)

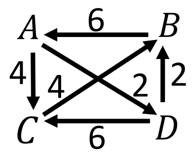

Figure 1: The WMG of the profile in Example 3.2.

For ranked pairs with a lexicographical tie-breaking, the following edges are added to the final ranking graph in order: $B \rightarrow A, D \rightarrow C, A \rightarrow C, A \rightarrow D$. Therefore, B is the winner in ranked pairs. For Schulze, the path margin of $D \rightarrow A$ and $D \rightarrow B$ are 4, $D \rightarrow C$ is 6, while the path margin of $A \rightarrow D$, $B \rightarrow D$, and $C \rightarrow D$ are all 2. Therefore, D is the winner in Schulze.

3.1.5 Pairwise Margins. Our decision trees use *pairwise margins* as features to learn and explain the voting rules. A *pairwise margin* is defined as the difference between every possible pair of pairs of candidates, i.e., the difference between the *pairwise victories of two pairs.* A *pairwise victory* is defined as the number of times candidate A is ranked higher than candidate B. The *pairwise margin* is a powerful feature for WMG-based rules (Copeland, ranked pairs, and Schulze), which calculate the winner on pairwise comparisons and can be used to efficiently represent each step in finding the winner of these rules.

Example 3.3. Consider a profile with seven votes: There are three votes of $[A \succ B \succ C]$, two votes of $[A \succ C \succ B]$, and two votes of $[B \succ A \succ C]$. Table 1 and Table 2 show the pairwise victories and (a subset of) pairwise margins of the profile respectively.

3.2 Explainable AI

Explainable AI, also known as XAI, is a subdomain of AI research that studies how AI models can be explained/interpreted. The need for explainability in AI models rises as the models become more complex and are increasingly used in high-risk tasks, such as in the field of medicine or automated driving.

3.2.1 *Explanation methods.*

The Defense Advanced Research Projects Agency's report on XAI defines three broad categories of methods for explaining AI systems [12]. These are deep-model explanation, model induction, and explanation by interpretable model. In this work, we focus on *explanation by interpretable model*, namely decision tree models.

Deep-model explanation and model induction are generated by training a complex deep model. Generally speaking, it is difficult to decode the entire structure of the deep model to be understood by humans. Thus, the resulting explanation is often a *broad* but possibly an *incomplete* explanation of the behavior of the deep model. Rudin et al. [21] argue that if the goal is to output an explainable classifier, training a deep model and training another interpretable proxy of that model is unnecessary if the performance is equivalent. We also find in our experiments that decision trees are able to achieve a perfect score under some settings, which further justifies our choice of using interpretable models as the classifiers.

Linear models can also work well as interpretable classifiers for generating explanations. However, preference profiles are difficult to encode into a set of features that linear models can use. Linear models are also unable to learn the correlation between groups of features. For this reason, we only use decision trees as the interpretable classifier in this work. We also experiment using different varieties of decision tree mining algorithms to test their efficacy.

3.2.2 *Assessing Explanation Quality.*

We define a good explanation as being simple enough for anyone to understand without domain-specific knowledge. This means the features should be easily understandable, and the structure should be simple. A complex explanation may be able to deliver more information about the decision process, but it will also have reduced explainability. On the other hand, an explanation that is too simple may be easier to process but will fail to deliver some intricacies of the decision process. Thus, an effective explanation model should capture the essence of the decision process while maintaining a reasonable degree of simplicity – a subjective measure. In our work, we constrain all our decision trees to have a maximum depth of 6 for $m = 3$. The maximum depth of 6 was a value that was chosen based on our assumption that a tree of depth less than 6 is simple enough for interpretation. We also explore using additional constraining algorithms such as GOSDT and hierarchical shrinkage to constrain our trees further while maintaining the accuracy performance.

3.3 Decision Tree models

A decision tree classifier is a machine learning algorithm that finds the best *split* for a feature at each depth and keeps building nodes until no further splits can be made. The resulting classifier is represented as a binary tree, with each leaf node denoting the predicted label value. The tree structure allows one to traverse the exact decision path inside the estimator. This makes decision tree classifiers especially useful for tasks in which an explicit decision path of the model is needed.

One problem the classic decision tree mining algorithm, Classification And Regression Tree (CART), faces is the size of the resulting tree. While the decision tree classifier's interpretability is useful, the algorithm is prone to overfitting without proper constraints, outputting a rather complex tree. This problem has been tackled with several different algorithms [2, 15]. In our experiments, we consider two optimizations to the decision tree algorithm, namely Generalized Optimized Sparse Decision Tree (GODST) [15], and Hierarchical Shrinkage [2].

3.3.1 *Classification and Regularization Trees.*

The CART algorithm splits the nodes based on the number of samples per class. In this work, we use two implementations of the CART algorithm to compare their efficacy. We consider python implementations of DecisionTreeClassifier from scikit-learn [17] and XGBClassifier from the xgboost [7] package. To distinguish these two different implementations, we refer to each as *Scikit-Learn* and *XGBoost*.

XGBoost is an ensemble classifier that uses gradient boosting to improve its trees. The classifier usually consists of multiple iterative trees, each improving upon the previous tree. However, XGBoost can also be used to train a single decision tree, the structure of which is equivalent to a CART tree. Because XGBoost's implementation of the CART algorithm sufficiently differs from that of scikit-learn's, we consider both models in the experiments. In order to differentiate the models, we will refer to scikit-learn's implementation as Scikit-Learn, and XGBoost as XGBoost.

3.3.2 *Generalized Optimal Sparse Decision Tree.*

Generalized Optimal Sparse Decision Tree (GOSDT) is an algorithm that uses gradient boosting to generate optimal sparse decision trees. Proposed by Lin et al.[15], this algorithm turns the input data into a simpler form by calculating a *threshold* for each continuous feature before mining the decision tree. The GOSDT algorithm reduces the feature search space by utilizing a gradient-boosted model to bin continuous features into categorical bins before the main algorithm. In this work, we skip this step because the features are binned before they are inputted into the model, the steps of which are explained in detail in Section 4.2.

3.3.3 *Hierarchical Shrinkage.*

Hierarchical Shrinkage is another method of pruning a decision tree to reduce its dimensions. Agarwal et al. [2] propose this method to reduce the tree's dimension to increase its interpretability while maintaining its performance. Given a trained decision tree, this algorithm regularizes the tree from the leaf nodes to merge unnecessarily.

4 EXPERIMENTS

We conduct our experiment by generating synthetic voting data and training and testing each decision tree algorithm to the dataset. Each training data comprises k randomly generated preference profiles with m candidates and n votes. Because the models are meant to explain the voting rule used to generate the training data, we evaluate them by testing against another randomly generated dataset which includes scenarios the models may not have faced before. In order to test the flexibility of our models, we generate and experiment with multiple datasets with different m and n.

We conduct experiments to determine the effect of the number of voters n and the number of alternatives m on learning performance. To test this, we use $k = 10,000$, $n \in [10, 100]$ and $m \in [3, 5]$. The value of k was chosen after experimenting with several different values between $[100, 500000]$ and finding a value that does not place an unnecessary computational burden while outputting good models.

Although the task of winner prediction is a multi-class problem, we reduce it into a binary task by training individual models for each candidate's victory. For example, if $m = 5$, we train five different classifiers to learn the victory condition of each candidate and average the accuracy score to evaluate the overall performance. This allows us to obtain five smaller decision trees that can be used to explain each scenario instead of one large decision tree that can be harder to interpret. In order to assess the accuracy of these resulting trees – as one tree alone does not explain the whole election – we average the independent accuracy scores of the m trees.

4.1 Synthetic voting profile

Each dataset comprises k preference profiles, which are composed of n votes which rank m candidates. We chose random generation to create our dataset due to the exponential increase in the number of possible profiles as m grows.

In order to ensure that there are no repeated profiles in the dataset, we randomly generate each profile until we get a profile that has yet to be added to the dataset. Although we are able to generate a complete dataset for $m = 3$ scenario, we found that the number of possible combinations for scenarios with $m > 3$ is too high. Because of this reason, we use random generation for $m = 3$ as well for the sake of consistency. A more detailed discussion about generating a complete dataset can be found in Appendix A.

To ensure the model has learned a correct rule, we generate two separate datasets for training and testing. In the training and testing data, we ensure that each tree has enough samples for both victories and losses to learn a correct tree. To achieve this, we set up the data generation procedure such that the training data contains an equal number of instances of victories for all candidates. First, we generate k sets of preference profiles, randomly iterating until each of m candidates wins in $\frac{k}{m}$ of the profiles, including ties. These profiles are then merged to complete the entire dataset. Once the profiles and the election outcome are computed, the profile is converted into vector space using the pairwise margin feature. We use this method to test our classifiers with $m = 3, 4$ and 5 to test the performance for more complex voting profiles.

Some voting rules considered in this work can result in ties. Instead of breaking the ties, we consider both of the tied candidates to be winners of that election. This is because adding a random or alphabetical tie-breaking mechanism will add to the rule's complexity, which may add an unnecessary burden to the models. We are interested in explaining the mechanism of the voting rule, so we consider each tied candidate to be the winner when training the models.

4.2 Input features

The generated data must first be encoded into a set of features for the decision trees to understand the election data. We use the *pairwise margins* feature discussed in Section 3.1.5. This feature represents the ranking properties of the profile, which is well-suited for WMG-based voting rules. We use only one type of feature so that the decision tree is easy to understand. In order to reduce the search space for the decision tree's thresholds, we pre-determine a threshold for each feature by binning them into categories and marking $False$ as -1 and $True$ as 1 to make the tree splitting algorithm always learns a threshold of 0 for a node.

4.3 Performance Metrics

We use the accuracy score to evaluate the resulting trees. Because each tree is trained on a binary classification task to predict the victory of a particular candidate, we can calculate their individual accuracy scores. To evaluate the entire set of trees that are trained on a voting rule, we calculate the average of the individual accuracy scores.

4.4 Experiment Environment

All of our experiments were conducted on a system with 2 AMD EPYC 7313 16-Core Processors with 256G RAM running Ubuntu 20.04.6 LTS.

4.5 Results

4.5.1 Decision Tree Structure. An instance of a decision tree is in Figure 2.

A decision tree has a binary-tree structure. Each non-leaf node represents a condition in the form of "if a certain pairwise margin is positive". Each leaf node is a judgment on whether the candidate is a winner. Given a voting profile, the decision path can be generated by finding the next node that the outgoing edge points to, depending on whether the condition is met or not. See Appendix C for more diagrams of decision trees.

4.5.2 Effect of n on Accuracy. We conduct our experiment with varying values of m and n to determine how different candidates and voters influence the learning process. Because odd values of n produce an incomplete tree that cannot handle even values of n, we only consider even values for n. This is due to the fact that there can be ties present with even values of n that do not appear in the odd cases. We train the trees for different values of $n \in [10, 100]$ incremented by 10. When the dataset is randomly generated, we find that increasing the value of n has a positive correlation with the models' performance, except for Copeland for Scikit-Learn and HSTree. The fact that Scikit-Learn and HSTree show a similar trend is not surprising, given the fact that HSTree is an optimization of Scikit-Learn. Upon checking the resulting decision trees, we find that the Scikit-Learn and HSTree resulted in decision trees that tend to underfit and perform poorly on the Copeland rule testing data. The general increase in performance may be attributed to the fact that the selected features are the ratio of votes, so the values converge more as the profile size increases. We also observe that XGBoost and GOSDT consistently score near 100% accuracy on all four rules. A plot of the accuracy scores of the models for each voting rule with varying values of n can be seen in Figure 3. Finally, when training/testing the trees for the $m = 3$ scenario by generating the complete dataset with every possible preference profile, we found that all of the algorithms were able to produce a perfect tree regardless of the value of n. This suggests that XGBoost and GOSDT can generate a perfect tree even with an imperfect dataset, while Scikit-learn and HSTree require a complete dataset to produce a perfect tree.

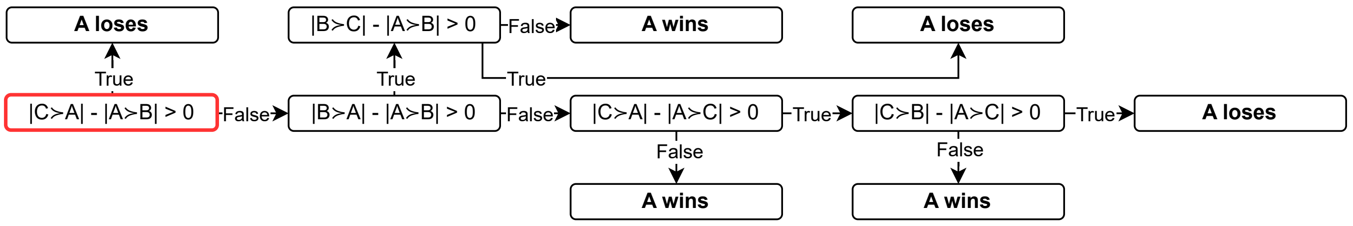

Figure 2: GOSDT tree predicting A's victory for the Schulze rule, $m = 3$.

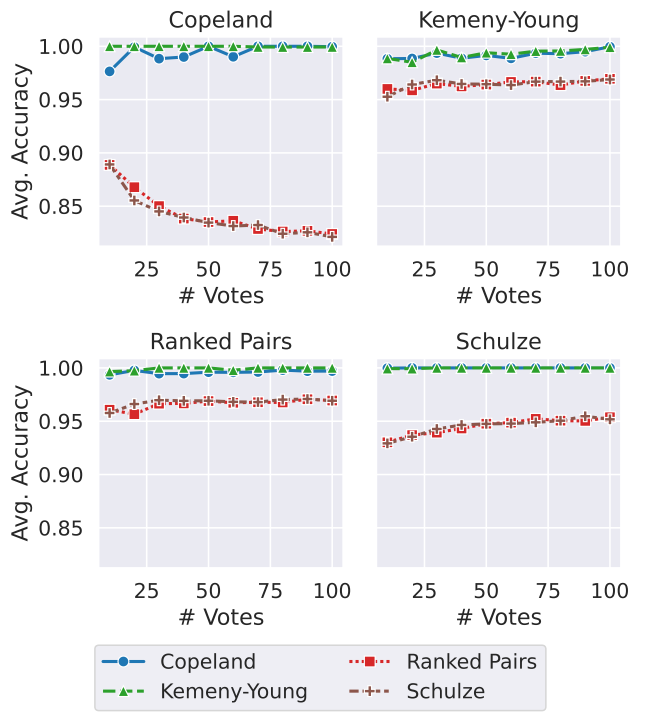

Figure 3: Averaged performance of models trained using randomly generated data over $n \in [10, 100]$

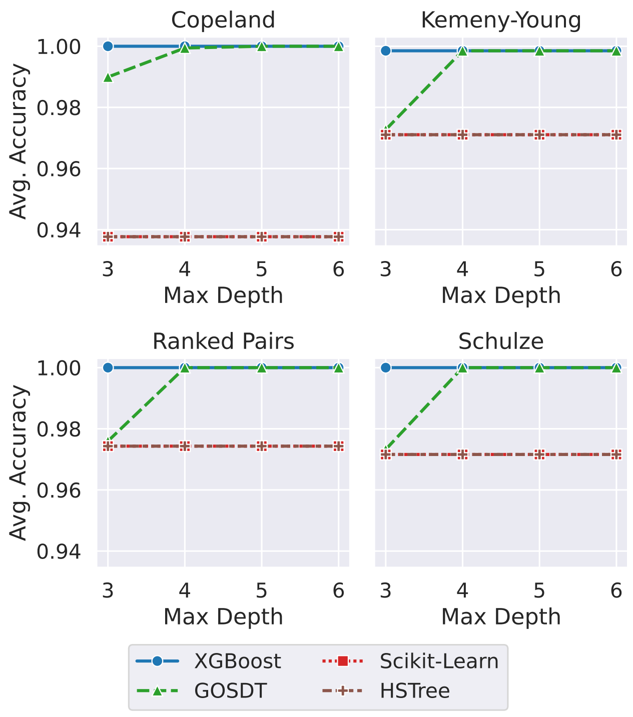

Figure 4: Averaged performance of models trained with $max\_depth \in [3, 6]$

4.5.3 Model Hyperparameters. In order to produce a tree that learns the voting rule while minimizing the size of the tree, we experiment with different hyperparameters for each algorithm. To find the correlation between the maximum depth of the trees and the accuracy, we test the algorithms with maximum depth values between $[3, 6]$ with $m = 3, n = 100$ setting. The results of this experiment can be seen in Figure 4. We find that the different algorithms require different depth settings to produce a perfect tree. The XGBoost algorithm was able to learn a perfect tree with a maximum depth set to 3. However, GOSDT required an additional layer of depth to produce a perfect tree. This behavior may be attributed to the fact that GOSDT applies a heavier penalty towards 'complex' decision trees, likely trying to avoid a tree with full leaves. We also find that limiting the criterion for leaf node splits to avoid redundant leaves is needed, which we set to 0.2 in our final experiments. The GOSDT and HSTree algorithm also offers a set of parameters relevant to each algorithm, which are again set after experimenting with different values. The hyperparameters for each model can be seen in Table 4 of Appendix B. In order to keep the results reproducible, the corresponding random seed of each algorithm is set to 0. Although XGBoost and GODST could output a perfect tree, the Scikit-Learn algorithm and HSTree algorithm could not learn a perfect tree in any of these settings. Since HSTree is an algorithm that runs on top of the Scikit-Learn algorithm, this result is not surprising.

4.5.4 Learning from $m > 3$ Scenarios. We conduct another set of experiments by varying the number of candidates, m. We train and test on profiles with $m \in [3, 5]$ to determine how well the same trees perform in a larger candidate setting. As expected, the average accuracy of the models decreases as the number of candidates grows, as seen in Figure 5. With these larger settings, we find that the GOSDT algorithm cannot converge, running out of memory

	Copeland	**Kemeny-Young**	**Ranked Pairs**	**Schulze**
XGBoost	1.0	0.99	1.0	1.0
GOSDT	1.0	0.99	1.0	1.0
Scikit-Learn	0.82	0.96	0.95	0.95
HSTree	0.81	0.96	0.95	0.95

Table 3: Final averaged accuracy score of best trees learned by model for $m = 3, n = 100$. KY refers to Kemeny-Young, and RP refers to Ranked Pairs

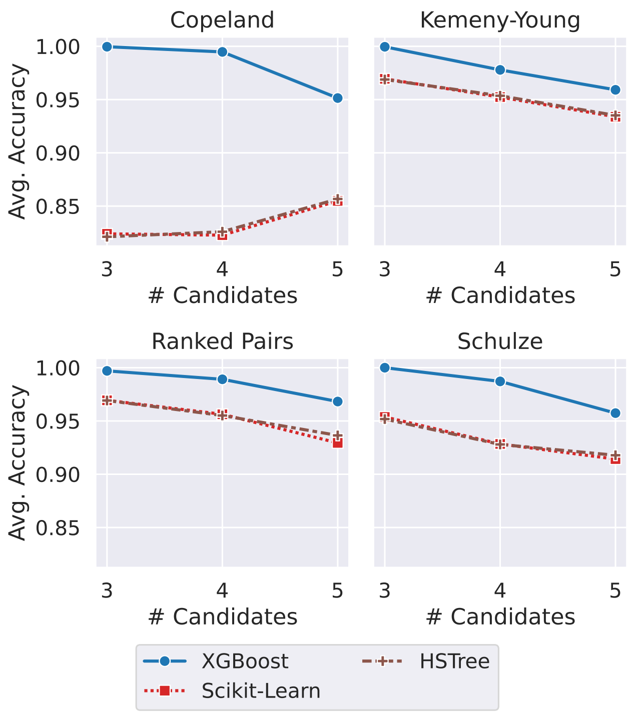

Figure 5: Averaged performance of models over $m \in [3, 5]$

on a 256G RAM system. This is likely because the size of the feature set grows exponentially with larger values of m and the fact that the random dataset needs to be much larger to consider more cases. We also face the problem of not having a complete dataset. Interestingly, Scikit-Learn and HSTree algorithms can be seen with increased performance for the Copeland rule. The unpacked decision trees show that the Scikit-Learn trees start to display increased performance with less overfitting compared to XGBoost, with 9 leaf nodes compared to the 42 leaf nodes of the XGBoost tree. However, this behavior was limited to the Copeland rule.

Looking deeper into the data generation process, we find the number of possible *pairwise margin* features for $m = 3$ was 147, which can be covered in a random generation of 100,000 samples. However, we were not able to confirm such a bound for $m > 3$ due to the fact that the number of profiles became too large to generate and check. Thus, we rely on a random generation process that is possibly incomplete to train the trees for $m > 3$ and find that it is computationally difficult to generate training data that covers every case. More details about the computation related to data generation can be found in Appendix A. Although the resulting decision is not *perfect*, we found that the trees are still able to score with a high accuracy score. For example, upon examining the Scikit-Learn tree for Copeland with $m = 4, n = 50$, which scored 99% accuracy, we find that 81 out of 125 of the leaf nodes are correct with most of the incorrect leaf nodes being correct around 10% of the time on the test dataset.

4.5.5 *Learning the Perfect Tree.*

The results show that XGBoost and GOSDT can often learn perfect trees in the $m = 3$ setting. For the Scikit-Learn decision tree and HSTree optimization, the models never perform above 90% accuracy with the same depth and leaf node constraints. In the Kemeny-Young rule, the accuracy scores can be seen improving as the size of the profiles grows larger. This suggests that the models, especially GOSDT and XGBoost, may be able to learn a near-perfect model given a dataset that contains every possible feature that can take place. The Ranked Pairs and Schulze rule display a very similar trend. Again GOSDT and XGBoost can be seen outperforming the other classifiers, producing a near-perfect tree for most cases. For Schulze, the models are able to find a perfect tree for $m = 3$. The Scikit-learn decision tree and HSTree can slightly improve their performance as the size of the profile grows, although the performance is still far from perfect. The final averaged accuracy score of the trees trained with $m = 3, n = 100$ can be seen in Table 3.

While most models had a high accuracy score (>90%) for the 4 voting rules that satisfy the Condorcet criterion, only XGBoost and GOSDT were able to learn a *perfect* rule that scored 100% accuracy. Upon examining the tree structure, we find that GOSDT produces a sparser tree than XGBoost. Compared to the 24 nodes present in the XGBoost tree in Figure 6 of Appendix C for the Copeland rule, the GOSDT tree, as can be seen in Appendix C Figure 7, outputs a tree with only 16 nodes while achieving the same perfect performance. The final trees trained on profile with $m = 3, n = 100$ can be seen in Figure 2 and Figures 6 and 7 of Appendix C.

The following theorem provides a correctness guarantee of the Schulze decision tree in Figure 2

THEOREM 4.1. *For three alternatives A, B, and C and any profile P, the GOSDT tree for Schulze in Figure 2 correctly decides whether A is a winner.*

For Schulze with three candidates, A becomes a loser when there is another candidate (for example, B) such that the path weight of A to B is strictly smaller than the path weight of B to A. This happens when the weight of $B \succ A$ in the weighted majority graph is positive and larger than either $A \succ C$ or $C \succ B$. Otherwise, A will be the winner. The full proof is in Appendix D.1.

5 CONCLUSION AND FUTURE WORK

We propose a framework for training decision trees to learn the outcome of a voting rule and serve as an explanation. We train different decision tree models on synthetic voting profile data for different voting rules. Our experiments show that voting rules that incorporate rankings into their mechanism (i.e., satisfy the Condorcet criterion) can be well estimated by decision tree models and produce a human-readable diagram. We use the*pairwise margin* features to train our dataset, which allows our decision trees to be agnostic to the number of voters in a voting profile. We find that XGBoost and GOSDT can learn a perfect tree even with an incomplete dataset, while Scikit-Learn and HSTree algorithm requires a dataset containing every possible case to learn a perfect tree. We also find that increasing the number of candidates requires a tree with a greater depth.

We also compare the performance and efficiency of different decision tree mining algorithms in generating explanations. Using an optimization algorithm, specifically, GOSDT, can help with the performance of the baseline decision tree and reduce its size at the cost of computational efficiency. We find that Copeland and Schulze can be perfectly learned with three candidates, and ranked pairs and Kemeny-Young achieve a near-perfect score experimentally.

Our framework can be used by an organization that needs to conduct an election with maximum transparency to the voters. The correct decision tree can be used to verify that the election was indeed conducted correctly and to audit the entire auditing process. For example, suppose a new (and complex) voting rule was implemented, and the public is not familiar with it. In that case, the decision tree can be referred to in order to understand why the final outcome was chosen and whether the public agrees with the mechanisms of the new voting rule. By having a visual explanation of the voting rule that captures every possible scenario, this framework can be used to increase the public's engagement in voting systems.

One direct extension of this work is generating simple decision trees for votes with multiple alternatives. Another future direction is to create new explainable voting rules based on decision trees. We have demonstrated that decision trees can act as proxies for voting rules. If we reverse this process, we can design new voting mechanisms that are both explainable and axiomatically desirable.

6 ETHICAL STATEMENT

This work is aimed to improve the public's engagement with voting theory. All of our experiments were conducted with simulated data, and no real-life data was collected or used for this work. All of the code used in our experiments and analysis are uploaded on a github repository, and will be made public if the work is accepted.

ACKNOWLEDGMENTS

Acknowledgements

REFERENCES

[1] Amina Adadi and Mohammed Berrada. 2018. Peeking inside the black-box: a survey on explainable artificial intelligence (XAI). *IEEE access* 6 (2018), 52138–52160.

[2] Abhineet Agarwal, Yan Shuo Tan, Omer Ronen, Chandan Singh, and Bin Yu. 2022. Hierarchical Shrinkage: Improving the accuracy and interpretability of tree-based models. In *Proceedings of the 39th International Conference on Machine Learning*. 111–135.

[3] Cem Anil and Xuchan Bao. 2021. Learning to Elect. In *Advances in Neural Information Processing Systems*. 8006–8017.

[4] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. 2020. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information fusion* 58 (2020), 82–115.

[5] Dávid Burka, Clemens Puppe, László Szepesváry, and Attila Tasnádi. 2021. Voting: a machine learning approach. *European Journal of Operational Research* (2021).

[6] Olivier Cailloux and Ulle Endriss. 2016. Arguing about Voting Rules. In *Proceedings of the 2016 International Conference on Autonomous Agents Multiagent Systems*. 287–295.

[7] Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*. 785–794.

[8] Arun Das and Paul Rad. 2020. Opportunities and challenges in explainable artificial intelligence (xai): A survey. *arXiv preprint arXiv:2006.11371* (2020).

[9] John A Doucette, Kate Larson, and Robin Cohen. 2015. Conventional machine learning for social choice. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*.

[10] Umberto Grandi. 2019. Agent-mediated social choice. In *The Future of Economic Design*. Springer, 89–96.

[11] Riccardo Guidotti, Anna Monreale, Salvatore Ruggieri, Franco Turini, Fosca Giannotti, and Dino Pedreschi. 2018. A survey of methods for explaining black box models. *ACM computing surveys (CSUR)* 51, 5 (2018), 1–42.

[12] David Gunning and David Aha. 2019. DARPA's explainable artificial intelligence (XAI) program. *AI magazine* 40, 2 (2019), 44–58.

[13] Mir Riyanul Islam, Mobyen Uddin Ahmed, Shaibal Barua, and Shahina Begum. 2022. A systematic review of explainable artificial intelligence in terms of different application domains and tasks. *Applied Sciences* 12, 3 (2022), 1353.

[14] John G Kemeny. 1959. Mathematics without numbers. *Daedalus* 88, 4 (1959), 577–591.

[15] Jimmy Lin, Chudi Zhong, Diane Hu, Cynthia Rudin, and Margo Seltzer. 2020. Generalized and Scalable Optimal Sparse Decision Trees. In *Proceedings of the 37th International Conference on Machine Learning*. 6150–6160.

[16] Scott M Lundberg and Su-In Lee. 2017. A unified approach to interpreting model predictions. *Advances in neural information processing systems* 30 (2017).

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

[18] Dominik Peters, Ariel D Procaccia, Alexandros Psomas, and Zixin Zhou. 2020. Explainable Voting. In *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (Eds.). 1525–1534.

[19] Ariel D. Procaccia. 2019. *Axioms Should Explain Solutions*. Springer International Publishing, 195–199.

[20] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. " Why should i trust you?" Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*. 1135–1144.

[21] Cynthia Rudin. 2019. Stop explaining black box machine learning models for high stakes decisions and use interpretable models instead. *Nature machine intelligence* 1, 5 (2019), 206–215.

[22] Wojciech Samek, Grégoire Montavon, Sebastian Lapuschkin, Christopher J Anders, and Klaus-Robert Müller. 2021. Explaining deep neural networks and beyond: A review of methods and applications. *Proc. IEEE* 109, 3 (2021), 247–278.

[23] Markus Schulze. 2011. A new monotonic, clone-independent, reversal symmetric, and condorcet-consistent single-winner election method. *Social choice and Welfare* 36, 2 (2011), 267–303.

[24] Sharadhi Alape Suryanarayana, David Sarne, and Sarit Kraus. 2022. Justifying Social-Choice Mechanism Outcome for Improving Participant Satisfaction. In *Proceedings of the 21st International Conference on Autonomous Agents and Multiagent Systems*. 1246–1255.

[25] Lirong Xia. 2013. Designing social choice mechanisms using machine learning. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. 471–474.

A DISCUSSION ABOUT DATA GENERATION

For a voting scenario with m candidates, there are $m!$ unique votes. Since there can be duplicate votes in a preference profile, the number of possible profiles with n votes can be expressed as a subset of these $m!$ votes with repeats. For n voters, the possible number of subsets given $m!$ can be expressed as $(m! + n - 1)!/(n! * (m! - 1)!)$. Then, the number of possible profiles given $m = 3, n = 100$ is 96560646. This value grows exponentially with larger values of m, with the number of possible profiles for $m = 4, n = 100$ resulting around $5 * 10^{24}$.

However, it should be noted that in many of these profiles can also result in the same *pair_margin* features, since the pattern of WMG may be similar. For example, in the case of $m = 3, n = 100$, we find that the number of unique pairwise margine features possible is 147, which can be easily covered when randomly generating 100,000 non-repeating samples. In fact, we find that the number of unique features in a voting profile with $m = 3$ and $n \geq 12$ are 147. This suggests that there is likely a threshold for the number of possible features in $m > 3$ scenarios as well, although we were not able to check it due to computing constraints.

B HYPERPARAMETERS USED IN EXPERIMENTS

Model	Parameters
XGBoost	{**max_depth:** 3, min_samples_split: 0.2}
Scikit-Learn	{**max_depth:** 6, min_samples_split: 0.2}
GOSDT	{regularization: 0.001, **depth_budget:** 4}
HSTree	{**max_depth**: 6, max_leaf_nodes: 7}

Table 4: Hyperparameters used for final tree

C MORE EXAMPLES OF GENERATED TREE DIAGRAMS

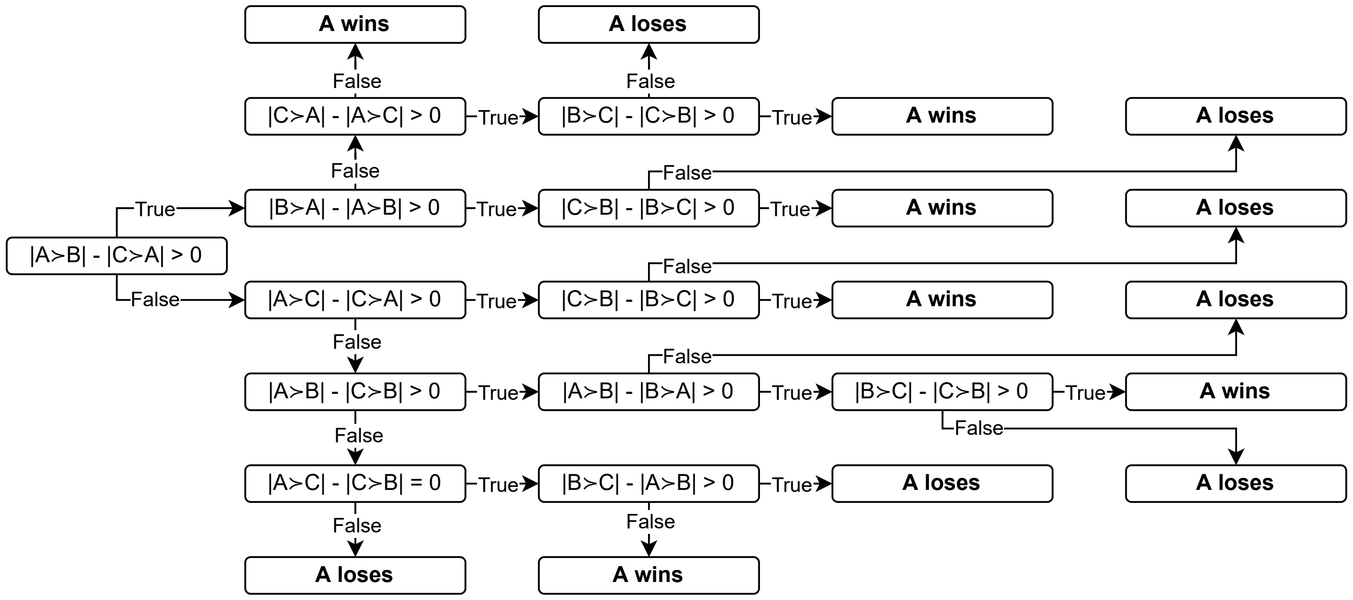

Figure 6: XGBoost tree predicting A's victory for the Copeland rule, $m = 3$.

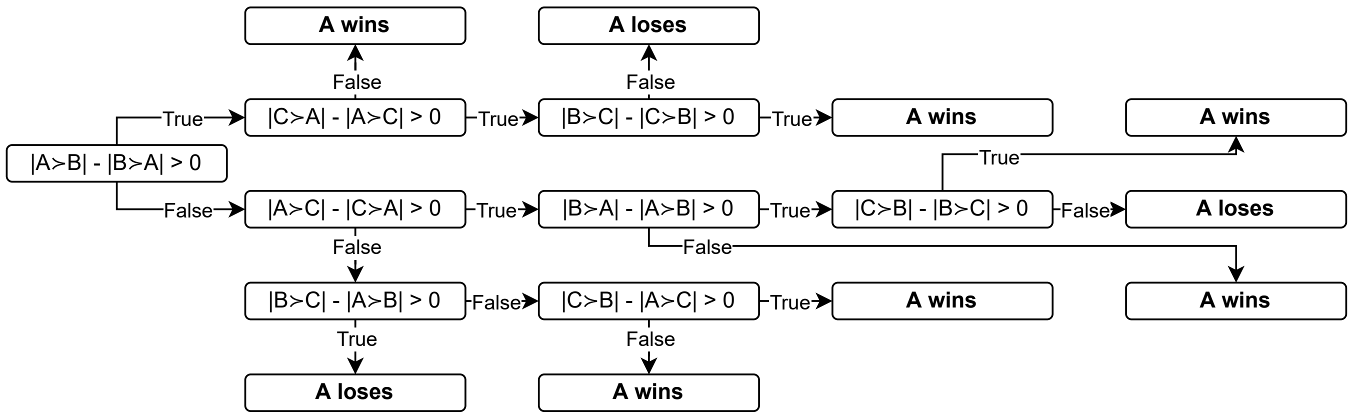

Figure 7: GOSDT tree predicting A's victory for the Copeland rule, $m = 3$.

D CORRECTNESS OF THE TREE

Throughout the section, we prove the correctness by enumerating every leaf node in the tree. Note that a leaf node can be uniquely represented by a path from the root to the leaf, and each branch choice on the path can be represented by "TRUE" or "FALSE", which is whether the condition at the node is satisfied. We use $|A \succ B|$ as the number of votes that prefers A to B in the profile P, and $D[A \succ B] = |A \succ B| - |B \succ A|$ be the marginal difference between A and B.

D.1 Schulze

Theorem 4.1. *For three alternatives A, B, and C and any profile P, the GOSDT tree in Figure 2 correctly decides whether A is a winner.*

Proof. First note that for any alternatives W, X, Y, and Z (that can be the same alternative), $|W \succ X| \geq |Y \succ Z|$ directly implies $D[W \succ X] \geq D[Y \succ Z]$, and $|W \succ X| > |Y \succ Z|$ directly implies $D[W \succ X] > D[Y \succ Z]$. We will use this property of D throughout the proof.

[TRUE]. In this case at least one of $C \succ A$ and $B \succ A$ is true, and $D[C \succ A] > D[A \succ B]$. If $C \succ A$ is true, the path weight of C to A is at least $D[C \succ A]$, while the path weight of A to C will not exceed $\max(D[A \succ B], D[A \succ C] < 0)$. If $C \succ A$ is true, A is dominated by B similarly. Therefore, A is a loser.

[FALSE, TRUE, TRUE]. The path weight of B to A is at least $D[B \succ A]$, while the path weight of A to B is at most $\max(D[C \succ B], D[A \succ B]) < D[B \succ A]$. Therefore, A is dominated by B and is a loser.

[FALSE, TRUE, FALSE]. In this case, $D[B \succ A], D[C \succ B]$, and $D[A \succ C]$ are positive. Among the three marginal differences, $D[B \succ A]$ is the smallest. The path weight of A to B and A to C will no larger than $\min(D[A \succ C], D[C \succ B])$, while the path weight of B to A and C to A will not larger than $D[B \succ A]$. Therefore, A is a winner.

[FALSE, FALSE, TRUE, TRUE]. In this case, $D[A \succ B] \geq D[C \succ A] > D[B \succ C]$, and $D[C \succ A] \geq 0$. Therefore, the path weight of A to C will not exceed $\max(D[B \succ C], D[A \succ C])$, while the path weight of C to A is at least $D[C \succ A]$. Therefore, A is dominated by C and cannot be a winner.

[FALSE, FALSE, TRUE, FALSE]. In this case, $D[A \succ B] \geq D[C \succ A] \geq 0$, and $D[B \succ C] \geq D[C \succ A]$. Similar to the [FALSE, TRUE, FALSE] case, A becomes a winner.

[FALSE, FALSE, FALSE]. In this case, $D[A \succ B] \geq 0$, and $D[A \succ C] \geq 0$. Therefore, the path weight of $A$to B and A to C are non-negative, while the reverse is non-positive. Therefore, A is a winner. □

D.2 Copeland

Theorem 1. *For three alternatives A, B, and C and any profile P, the XGBoost tree for Copeland in Figure 6 correctly decides whether A is a winner.*

Proof. **[TRUE, TRUE, TRUE]**. In this case, we know that B beats A and C beats B in the head-to-head competition. Since $|A \succ B|$ is greater than $|C \succ A|$, we know that $|A \succ C|$ is greater than $|B \succ A|$ and exceeds half of the voters. Therefore, A beats C, and three alternatives form a cycle in the WMG. Therefore, all the alternatives including A are winners.

[TRUE, TRUE, FALSE]. B still beats A, and C loses or is tied with B. In this case, B has a score of at least 1.5, while A beaten by B has at most 1. Therefore, A is a loser.

[TRUE, FALSE, TRUE, TRUE]. In this case, C beats A, and B beats C. And since that $|A \succ B|$ is greater than $|C \succ A|$, A beats B. Therefore, all the alternatives are co-winners.

[TRUE, FALSE, TRUE, FALSE]. Similar to the [TRUE, TRUE, FALSE] case, C wins A and does not lose B and becomes the unique winner.
[TRUE, FALSE, FALSE]. In this case, A does not lose either B or C. Moreover, since $|A \succ B|$ is greater than $|C \succ A|$, A wins at least one of B and C. Therefore, A gets a score of at least 1.5 and becomes the unique winner.
[FALSE, TRUE, TRUE]. In this case, A beats C and C beats B. Since both B and C cannot be a Condorcet winner, A will always be a winner regardless of the relationship between B and C.
[FALSE, TRUE, FALSE]. In this case, A beats C, and B is not beaten by C. Since $|A \succ B|$ is not greater than $|C \succ A|$, A is beaten by B. Therefore, B gets a score of at least 1.5, and A cannot be the winner.
[FALSE, FALSE, TRUE, TRUE, TRUE]. Similar to the [FALSE, TRUE, TRUE] case, A beats B, and B beats C. Therefore, A is a winner.
[FALSE, FALSE, TRUE, TRUE, FALSE]. Similar to the [FALSE, TRUE, FALSE] case. A beats B, and C is not beaten by anyone. Moreover, since $|A \succ B|$ is not greater than $|C \succ A|$, A is beaten by C. Therefore, C gets a score of at least 1.5, and A cannot be the winner.
[FALSE, FALSE, TRUE, FALSE]. In this case, A does not beat either B or C. Moreover, since $|A \succ B|$ is greater than $|C \succ B|$, B beats C and becomes the unique winner. Therefore, A is not a winner.
[FALSE, FALSE, FALSE, TRUE, TRUE]. A does not beat C. Since $|A \succ C|$ is equal to $|C \succ B|$, C does not beat B. Then from $|A \succ B|$ is less than $|B \succ C|$ and not greater than $|C \succ B|$, we know that B beats A. Therefore, A cannot be a winner.
[FALSE, FALSE, FALSE, TRUE, FALSE]. Similarly, we have A does not beat C and C does not beat B. On the other hand, we know that $|A \succ B|$ is not less than $|B \succ C|$ and not greater than $|C \succ B|$. Therefore B does not beat C either, and B and C form a tie. Since $|A \succ C|$ is equal to $|C \succ B|$, A and C also form a tie. And as $|A \succ B|$ is between $|B \succ C|$ and $|C \succ B|$, A and B form a tie, too. Therefore, there is a three-way tie between all the alternatives, and everyone is a co-winner.
[FALSE, FALSE, FALSE, FALSE]. In this case, A does not beat C. If $|A \succ C|$ is greater than $|C \succ B|$, as $|A \succ C|$ is smaller than the half, B must beat C. And since $|A \succ B|$ is not greater than $|C \succ B|$, B must also beat A. Therefore, B is the Condorcet winner, and A cannot be a winner. If $|A \succ C|$ is smaller than $|C \succ B|$, we consider the following subcases. (1) A is tied with C. In this case, since $|A \succ C|$ is smaller than $|C \succ B|$, C beats B. Other the other hand, A cannot beat B since $|A \succ B|$ is smaller than $|C \succ A|$. Therefore, A cannot be a winner. (2) C beats A. In this case, A can be a winner if and only if there is a cycle in WMG. However, this is not true because $|A \succ B|$ is not greater than $|C \succ B|$. Therefore, A cannot be a winner.

□

THEOREM 2. *For three alternatives $A, B,$ and C and any profile P, the GODST tree for Copeland in Figure 7 correctly decides whether A is a winner.*

PROOF. **[TRUE, TRUE, TRUE]**. In this case, A beats B, C beats A, and B beats C. Therefore, there is a cycle in the WMG, and everyone is a winner.
[TRUE, TRUE, FALSE]. In this case, A beats B and C beats A, but B does not beat C. Therefore, C has a score of at least 1.5 while A's score is 1. Therefore, A is not a winner.
[TRUE, FALSE]. In this case, A beats B and is not beaten by C. Therefore, A has a score of at least 1.5 and is the winner.
[FALSE, TRUE, TRUE, TRUE]. This case is a cycle of $A \succ C \succ B \succ A$. Therefore, everyone is a winner.
[FALSE, TRUE, TRUE, FALSE]. In this case, B wins A and is not beaten by C. Therefore, A cannot be the winner.
[FALSE, TRUE, FALSE]. In this case, A wins C and is tied with B. Therefore, A is a winner.
[FALSE, FALSE, TRUE]. In this case, A does not beat either B or C. If A is tied with B, then B wins C and gets a score of 1.5, while A's score is at most 1. If A loses B, A's score will always be lower than B's. Therefore, A cannot be a winner.
[FALSE, FALSE, FALSE, TRUE]. This is similar to the [FALSE, FALSE, TRUE] case. A does not beat either B or C, and A's score is strictly lower than C's. Therefore, A cannot be a winner.
[FALSE, FALSE, FALSE, FALSE]. In this case, both $|A \succ B|$ and $|A \succ C|$ are not larger than half. This means that both $|B \succ C|$ and $|C \succ B|$ are not larger than half. Therefore, the situation must be a three-way tie in the head-to-head competition. Therefore, everyone is a winner.

□