Learning Voting Trees

Ariel D. Procaccia, Aviv Zohar, Yoni Peleg, and Jeffrey S. Rosenschein
(2007 - PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE)

Raissa Chut Steinberg - 341241297
Aviv Kapitulnik - 214096745

Topic/Main claims

The paper discusses the procedure of learning voting trees given tournaments and their winners according to a specific voting rule which is unknown to us. It outlines how, for some voting rules, the assumption that it is possible to learn their behavior and construct a voting tree close to our target is made, and we can find a compact representation for a voting rule that the designer has in mind.

The main claim is that the class of rules represented by binary voting trees is PAC-learnable and that given that certain conditions take place, only a polynomial number of observations (tournament with its winner) are required to achieve the desired result/

A binary voting tree is a tree structure that is used to determine a winner among a given set of alternatives through pairwise comparisons in each inner node - where the leaves are a certain assignment of the alternatives, the pairwise comparisons start from the bottom of the tree (the leaves) and the winner in each pairwise comparison rises up to its parent node until we are left with a single alternative - the winner. A voting tree is compactly represented if it is described efficiently - there are no redundant leaves in the tree. A voting tree is consistent with given data (tournaments and their winners) if, for each tournament (pairwise comparison results from voter preferences), the tree's predicted winner matches the actual election outcome according to what the designer suggested.

Important notations and definitions

The paper relies on several theorems that support the main claims. We'll focus on the two main ones, before doing that, let's define the relevant notations and additional definitions that will serve as a foundation in understanding the theorems the paper talks about:

- A - set of alternatives, $|A| = m$.
- Z - a sample space, in our setting, the sample space is a set of preferences that define a tournament (lists of preferences of voters).
- f^* - a function from Z to A which defines the voting rule (the winner according to the designers idea, given a tournament).

- $\hat{Z}$ - a set $\{(z_1, f^*(z_1)), (z_2, f^*(z_2)), , ..., (z_t, f^*(z_t)), \}$ where $\{z_1, z_2, ...z_t\} \subseteq Z$ and $\{f^*(z_1), f^*(z_2), ...f^*(z_t)\} \subseteq A$ are sampled *i.i.d.* according to an unknown distribution D over Z (our training set).

- A few classes of functions that will help us in understanding further ideas:

  1. $\mathcal{F}$ - A class of functions $f : Z \to A$, in our setting it is a class of voting trees
  2. $\mathcal{F}_m$ - A class of functions $f : Z \to A$, in our setting it is a class of voting trees over m alternatives
  3. $\mathcal{F}_m^n$ - A class of functions $f : Z \to A$, in our setting it is a class of voting trees over m alternatives with at most n leaves

- PAC-learnable - a function class $\mathcal{F}$ is PAC-learnable if there exists a learning algorithm that can learn it with high probability for any distribution D over the sample space Z (the sample space defined above).

  Given $\epsilon, \delta \in (0, 1)$, there exists a sample complexity $s(\epsilon, \delta)$ such that if the sample size S is at least s and the samples are drawn i.i.d. from D, then with probability at least $1 - \delta$, the learned function $\hat{f}$ satisfies:

  $$\Pr_{z \sim D}[\hat{f}(x) \neq f^*(x)] \leq \epsilon,$$

  where f^* is the target function.

  The features are the preferences of voters that define a tournament and the labels are the winners according to f^*.

- A class F shatters a set $S \subseteq Z$ if there exist two functions $f, g \in F$ such that:

  1. $\forall z \in S,\ f(z) = g(z)$.

  2. For every subset $S_1 \subseteq S$, there exists a function $h \in F$ such that:

  $$h(z) = \begin{cases} f(z) & \text{if } z \in S_1, \\ g(z) & \text{if } z \in S \setminus S_1. \end{cases}$$

  This definition is relevant to us as it helps us define the next idea which sees abundant use throughout the paper.

- $D_G(\mathcal{F})$ - The generalized dimension of $\mathcal{F}$ is the greatest integer $\mathbf{d}$ such that there exists a set $\mathbf{S}$ where $|\mathbf{S}| = \mathbf{d}$ that is shattered by $\mathcal{F}$.
- $\mathcal{T} = \mathcal{T}(A)$ - A set of all tournaments over the alternatives set A.
- A voting rule is a function $F : \mathcal{T} \rightarrow A$

  An additional idea used in Theorem five:
- TREE-SAT - The TREE-SAT problem is a problem in which we are given a binary tree where some of the leaves are already labeled by alternatives and a training set that consists of pairs $(\succ_j, x_{i_j})$ where $\succ_j \in \mathcal{T}$ and $x_{i_j} \in A$ and we're being asked whether there exists an assignment of alternatives to the rest of the unlabeled leaves that is consistent with the training set - that is, when using the tree with said assignment we will get the winners that would be provided by the target function given the training set.

The TREE-SAT we defined above is used extensively in theorem 5 and the following empirical part.

Additionally, we assume that a target function $f^*(z)$ exists, and the given training examples are, in fact labeled by it. (The rule that the designer provided, f^*)

After these definitions, we can show the main results of the paper:

Main Results

Theorem 3.

This theorem shows how the generalized dimension of $\mathcal{F}$ $(D_G(\mathcal{F}_m))$ is exponential on the number of alternatives m, $D_G(\mathcal{F}) = \Omega(2^m)$. From the proof of said theorem, we can understand how expressive voting trees are.

In simpler words, this means that a large number of examples is needed to learn voting trees in the PAC model.

Theorem 4.

This theorem shows that if the number of leaves n is polynomial in m then $D_G(\mathcal{F}_m^n) \leq n(\log m + \log n)$, the dimension of $\mathcal{F}_m^n$ is polynomial in m. From an additional Lemma shown in the paper, it is implied that the sample complexity of the class is polynomial in m.

In simpler words, this means that given a training set of polynomial size in m and two constants ϵ, δ, any algorithm that returns some voting tree consistent with the training set (consistent with the voting rule) is an $(\epsilon, \delta) - learning \quad algorithm$ for $\mathcal{F}_m^n$. $\mathcal{F}_m^n$ is PAC-learnable.

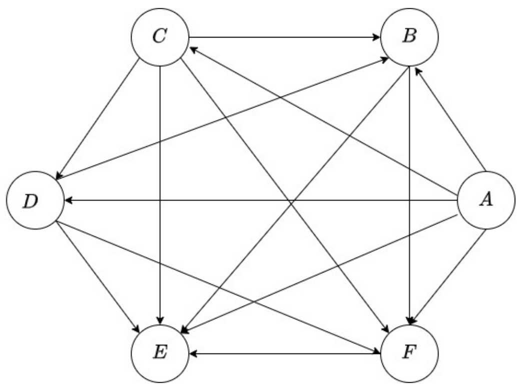
Theorem 5 and Empirical Results

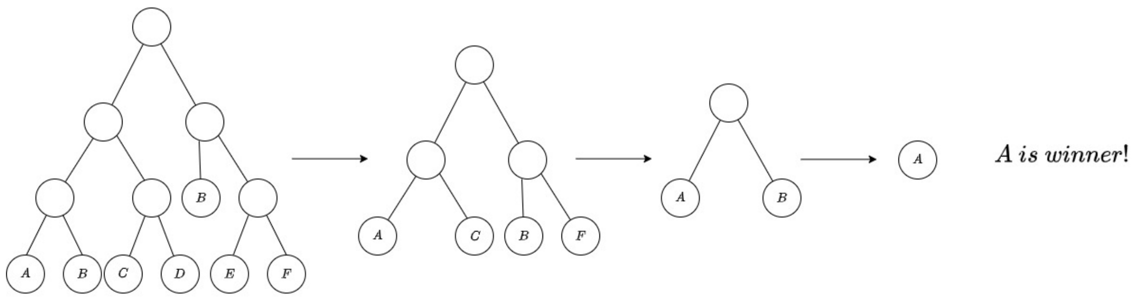
This theorem states that the TREE-SAT is $\mathcal{NP}$-complete. Despite the stated, it is shown that in practice, sometimes it is possible to solve the TREE-SAT problem. In the empirical results, simulations are used to learn a voting tree that aligns with the training set and efficiently solves the TREE-SAT problem within a reasonable timeframe.

Combining theorem five and the empirical results, it is shown that while, in theory, solving the TREE-SAT problem is a computationally challenging task, for many tree structures, the task becomes manageable and practically possible.

These results, together with the two other theorems that came from a cited paper/book on machine learning,g show us that the process of learning a desired voting rule and being able to ensure it satisfies different wished properties is a manageable task given certain preliminaries take place.

Helpful Figure
A figure that demonstrates how the process of a binary voting tree will work given a tournament:





$The\ irreflexive\ binary\ relations\ the\ tournament\ defines\ over\ the\ alternatives:$

$$A \succ C \succ D \succ B \succ F \succ E$$

Cited paper
The most relevant paper cited in this paper is "Small binary voting trees" Trick, M. (2006). Published in Proceedings of the First International Workshop on Computational Social Choice (pp. 500-511). Trick's work considers binary voting trees just as defined above, and suggests that voting rules can often be represented in a compact way, given by their insight over the existence of small binary voting trees. Also, motivate and serve as a basis for why we would be interested in learning and exploring such structures used as voting procedures.

Paper that cited this paper

The paper 'Automated design of scoring rules by learning from examples' in Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems-Volume 2 (pp. 951-958) by Procaccia, A. D., Zohar, A., & Rosenschein, J. S. (2008, May), cites our paper and relies on its findings and the setting it presents which makes possible to translate a voting rule into a compact representation of it that can be easily computed, just like done in our paper by using voting trees as this representation.

Part 2 - Implementation

In our work we will use simulations to learn different structured binary voting trees with various leaf numbers, that is, we want to find a good enough leaves assignment of the alternatives as discussed in the theoretical part. We are interested on the mean accuracy of the voting trees for each of the leaves number over all of the different structures used.

The paper provides a thorough discussion on the complexity of learning the assignment. Our focus is on understanding how the number of leaves in a voting tree impacts its conciseness with respect to a given voting rule, as well as its overall accuracy.

Differently than the work shown in the paper, we are not interested in learning a "perfect" voting tree, but using a greedy algorithm that will help us create a good approximation to a concise voting tree. In the implementation part we:

- The data set used is the Jester dataset, we altered the data in order to turn it into a ranking set, then turned 80% of it into a train set, and the remaining data was turned into the test set (what will be used to calculate the accuracies). From each of the sets we created tournmanets from subsets of 100 rankings each. In order to figure out the winner in each tournament we implemented various voting rules and used them.

- To reduce the randomness in our results we decided to randomly generate 5 trees for each amount of leaves that we decided to check on, moreover, we generated a balanced tree with the same amount of leaves as the former trees. The learning algorithm was performed over the sets of all of the trees generated.
  As we stated above, we implemented 4 voting rules in order to pick a winner, the voting rules were: Plurality, Borda, IRV and Copland, we used then created tuples of tournaments and its winner to create a "labeled" dataset as defined in the theoretical part.

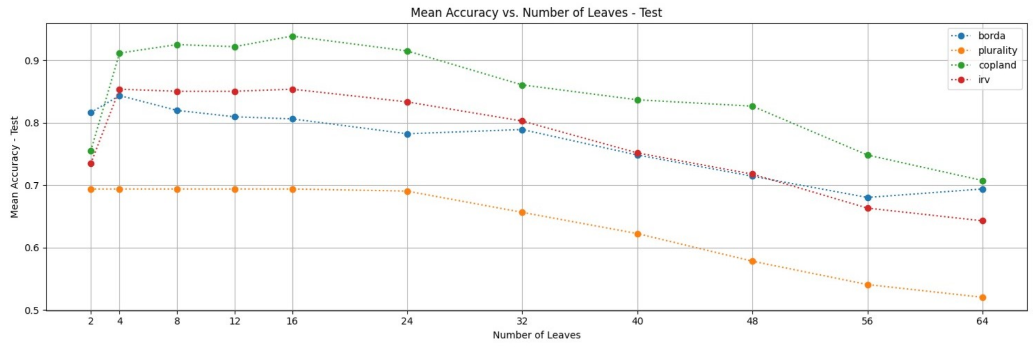
- We learn each voting rule using a greedy iterative approach that worked in the following way:

  - In the beginning we were given a tree and a set of its leaves, we filled all of the leaves with "empty candidates", these candidates lose to every other candidates in pairwise comparisons. We used them in order to fill the tree starting from an empty state.

  - Now, for 20 iterations we ran the following:
    We went over all of the leaves, for each leaf we checked each all of the possible alternatives and for each alternative how many tournaments in which the tree picks the correct winner this alternative adds, after we check all of the alternatives we pick the alternative that adds the most.

- We calculated the accuracy of the learned leaves assignments for the test set defined in the beggining. The accuracy is calculated by the number of times the backwards induction returns to us the actual winner, given a tournament (a batch) and a tournament divided by the number of batches.

Below we can see the distribution of the accuracies mean over all tree structures for each voting rule as a function of the number of leaves in the tree. On the expected figure we gave in in Part **1**



The highlights of the implementation part are stated here, where the detailed discussion on our results and methods are presented more thoroughly in the notebook.