Homework 2

Social Choice and Preference Aggregation - Dr. Reshef Meir

December 29, 2020

1 Theory

1) Plurality voting with abstentions is the same as plurality, only that voters get a small ϵ added utility if they are able to abstain and do not need to vote. Consider a pure Nash equilibrium in Plurality voting with abstentions. Prove that at most one voter votes (all others abstain).

2) The voting rule f first checks if there is a Condorcet winner, and otherwise selects the Veto winner (break ties lexicographically). Provide an efficient algorithm to determine if a given coalition $S \subseteq N$ has a group manipulation in favor of candidate a.

3) Prove that every manipulation in Plurality is safe.

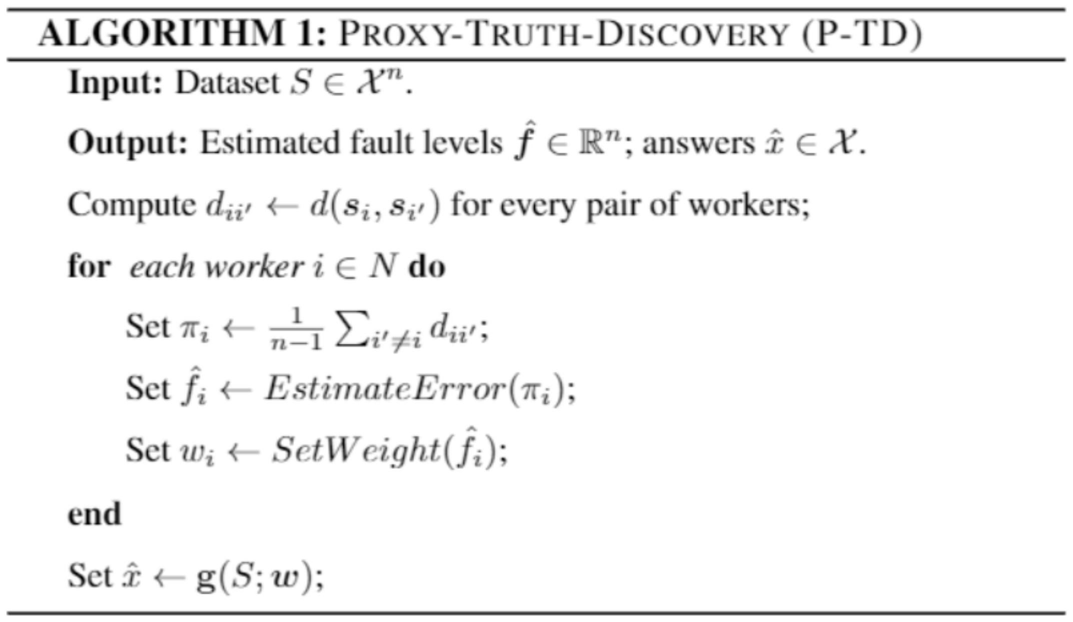
4) Consider the following distribution over cardinal types with 3 candidates $A = \{a, b, c\}$:

$$\begin{cases} (1,0,-1) & w.p. 0.4 \\ (0,1,-1) & w.p. 0.4 \\ (-1,0,x) & w.p. 0.2 \end{cases}$$

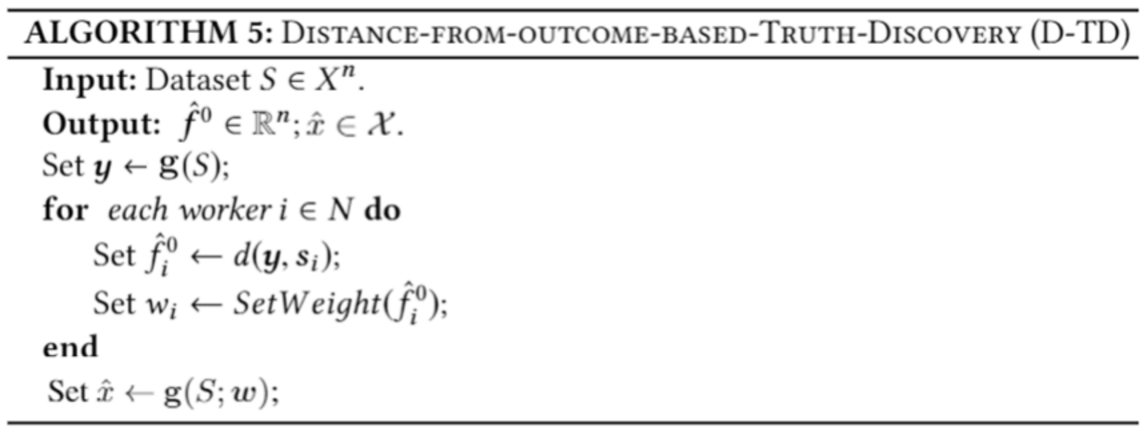
Suppose we sample n = 3 voters i.i.d. from this distribution. The voting rule is Plurality with uniform tie-breaking. For what values of x we get that truth-telling is a Bayes-Nash equilibrium?

2 implementation

Consider two algorithms for ranking discovery:



```
ALGORITHM 1: PROXY-TRUTH-DISCOVERY (P-TD)
Input: Dataset $S \in \mathcal{X}^n$.
Output: Estimated fault levels $\hat{\boldsymbol{f}} \in \mathbb{R}^n$; answers $\hat{x} \in \mathcal{X}$.
Compute $d_{ii'} \leftarrow d(\boldsymbol{s}_i, \boldsymbol{s}_{i'})$ for every pair of workers;
for each worker $i \in N$ do
    Set $\pi_i \leftarrow \frac{1}{n-1} \sum_{i' \neq i} d_{ii'}$;
    Set $\hat{f}_i \leftarrow EstimateError(\pi_i)$;
    Set $w_i \leftarrow SetWeight(\hat{f}_i)$;
end
Set $\hat{x} \leftarrow \mathbf{g}(S; \boldsymbol{w})$;
```



```
ALGORITHM 5: DISTANCE-FROM-OUTCOME-BASED-TRUTH-DISCOVERY (D-TD)
Input: Dataset $S \in X^n$.
Output: $\hat{f}^0 \in \mathbb{R}^n$; $\hat{x} \in \mathcal{X}$.
Set $\boldsymbol{y} \leftarrow \mathbf{g}(S)$;
for each worker $i \in N$ do
    Set $\hat{f}_i^0 \leftarrow d(\boldsymbol{y}, \boldsymbol{s}_i)$;
    Set $w_i \leftarrow SetWeight(\hat{f}_i^0)$;
end
Set $\hat{x} \leftarrow \mathbf{g}(S; \boldsymbol{w})$;
```

Notes

- We use $g = \{Borda, Copeland\}$. The weighted version of Borda multiplies each voter Borda scores by its weight (even if it's negative!). Weighted Copeland: Calculate pairwise scores with regards to the weights. In the tournament graph, edge weight is the pairwise score of the winner. Scores for the nodes are the sum of weights on outgoing edges.

- EstimateFault() is the identity function. SetWeight() is $w_i = \frac{1}{2} - f_i$. $d(x, y)$ is a normalized version of the Kendall-Tau distance: $d(x, y) = \frac{KT(x,y)}{\binom{m}{2}}$ where m is the number of alternatives.

- UW() (unweighted) simply returns the relevant social choice function over S, e.g. Borda or Copeland. If you use your implementation from exercise 1, make sure you follow up on comments given in the feedback and that the function returns the correct ranking!

The Task A city planner tried to measure the heights of 25 buildings. In order to do that, he invited some of the finest architects he knows to estimate the buildings' heights by looking from his office's window. Each architect ranked the 25 buildings from lowest to highest and submitted his rankings to the city planner. He, however, had no idea how to derive the real rankings of the buildings from those estimations and contacted you for help. The city planner has some information regarding the true rankings of some of the buildings. This information is verified and he hands it to you so you can use it for an evaluation of your algorithms. Your assignment is to write a Python program according to the following demands:

1. Implement functions that calculate PTD, DTD, unweighted (UW). Each of the first three functions should have two versions: one for Borda, and one for Copeland. The input to these functions is only the preferences S and the output is the estimated ranking $\hat{x}$ and voters' fault levels $\hat{f}$ (See algorithms below).

2. Create a scatter plot graph that shows, for each architect, the distance from the truth $d(s_i, x^*)$ vs. the proxy distance π_i. Each architect is a point on the graph. The distance function throughout the exercise is the normalized Kendall-Tau and the truth is the true rankings you are given over 13 buildings. Note that you only need to consider how i ranked these 13 buildings. You may create a smaller input file S' that only has these 13 buildings.

3. Create two graphs (one for Borda and one for Copeland) that show the average error as a function of sample size. Sample sizes are in the range (5, 85) with intervals of 5. The sample size s means that you uniform-randomly take only s of the votes (architects) and run the algorithms according to them. For each sample size, calculate the average true error over 500 random samples from S'. Each graph should have 3 curves- for DTD, PTD and unweighted. The true error is the distance $d(\hat{x}, x^*)$ (distance between the estimated and true ranking) over the 13 buildings.

4. Your program should read the votes S, call the 6 functions mentioned in section 1, and create a csv file with your estimations $\hat{x}$ over all 25 buildings. The csv file should have 6 rows as follows: 1. DTD Borda 2. DTD Copeland 3. PTD Borda 4. PTD Copeland 5. UW Borda 6. UW Copeland

5. (Bonus – up to 15 points based on originality and performance on all 25 buildings): Implement your own truth-discovery method. If you choose to take the Bonus:
   - Add a written explanation of your motivation and choices.
   - Include the code
   - Add its output to the graphs in task 3
   - Add your estimated ranking over all 25 buildings as a 7th row in task 4

The Data: voters.csv: the estimations (rankings) of 85 architects (denoted by $S = (s_1, ..., s_n)$.

truth.csv: the true rankings among 13 of the 25 buildings. The first row in this file is the buildings for which the truth is supplied, and the second row is the true order among them (denoted by x^*).

Final notes:

- All ties should be broken uniformly at random

- For PTD, we use $\mu = 0$ (equivalent to having EstimateFault() be the identity function)
- Your program should read the votes files attached to the exercise - it might also be tested against different votes files. Specifically, your program should assume the votes file is in the same directory and read it from "votes.csv", and create the csv file, to be named "estimations.csv", into the same directory.
- A template is attached to the assignment- feel free to use it as a base for your code as it meets the requirements; however, you are not required to do that.

Submission instructions:

- A short PDF file that contains an explanation of your algorithm (If you chose to create one) and the graphs mentioned in sections 2, 3.
- Your code (a .py file) that creates the required csv file.
- Both files should be submitted as a .zip file.
- Submission is in pairs.
- Make sure that the ID's of both submitters are either the .zip file's name (separated by _), or mentioned in the PDF file.