Random Voting Report

Tariq Georges

May 2024

Introduction

In social choice theory, distortion-based analysis evaluates how effectively different voting systems identify the optimum candidate. The optimum candidate is the candidate that minimizes the distance between voters and the chosen candidate when arranged in a metric space. Here's how the experiment works: voters and candidates are distributed in a space, and voters rank the candidates by their proximity. The mechanism uses this ranking information (relative closeness) to determine the candidate that minimizes the total distance for all voters.

The experiment aims to analyze the distortion of different voting rules by assessing how well the candidates chosen align with the optimal candidate. Distortion is defined as the ratio of the total cost (sum of all distances from voters) for the candidate selected by a given voting rule relative to the cost for the optimal candidate. The experiment visualizes these results with Voronoi diagrams, plotting voter preferences based on all permutations of possible candidate rankings.

Experiments

The goal of this experiment is to evaluate how well the winners chosen by different rules align with the optimal winner. Distortion is defined as the ratio of the total cost (sum of distances from all voters) for the candidate selected by a specific voting rule compared to the cost for the optimal candidate. Here, "optimal" is the candidate that minimizes the total cost.

Along with that the experiment also plots all of the created plots as vornoi digrams based on each permutation of the possible orderings of candidates. Since the experiment tests 4 candidates with 10,000 voters we use $4! = 24$ different orderings to color each voter by their preference profile.

First, we generate 10,000 voters uniformly random on a plot. We also generate the 4 candidates at random with the same function

```python
def gen_and_plot_points(voters, candidates):
    x1 = []
    y1 = []
    for _ in range(voters):
        x1.append(random.randint(-60, 60))
        y1.append(random.randint(-60, 60))

    x2 = []
    y2 = []
    for _ in range(candidates):
        x2.append(random.randint(-50, 50))
        y2.append(random.randint(-50, 50))

    for i in range(candidates):
      print(f'canidate {i} coordinate: ({x2[i]}, {y2[i]})')
```

After, I generate the preference profile for each voter. This is done by computing the distance from a voter to each candidate. The profile is the sorted orderings of each candidate from closest to farthest away from the voter. I used a function to map each distance to an index and export the ordered indices to be used as the preference profile $Ex : [0, 3, 2, 1]$
Next we generate the cost for each candidate from the points. The cost is calculated by summing the distances of all voters to each candidate. The optimum candidate is the candidate with the minimum cost. This is shown by this function where x is a candidate and x^* is the optimum candidate. $x_d^* \in \arg\min_{x \in X} C(x)$.
To perform some analysis on our points we utilize 3 different voting rules to begin to calculate the distortion from our voting rules and candidates. The distortion of f (the voting rule used) is the worst-case ratio between the cost of the candidate chosen by f, and the optimal candidate x_d^* (determined with knowledge of the actual distances d). Formally, the distortion is defined as:

$$\rho(f) = \max_P \sup_{d:d\sim P} \frac{C(f(P))}{C(x_d^*)}$$

Utilizing the voting rules and the optimum candidate we can calculate distortion to find the voting rule that closely aligns with the optimal candidate choice.
The rules are:

1. Random Dictator: A voter is selected at random, and their top choice becomes the winner.
2. Plurality: Each voter votes for their closest candidate, and the candidate with the most votes wins.
3. Borda: Each voter ranks the candidates, assigning scores based on their rank, and the candidate with the highest total score wins.

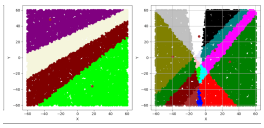
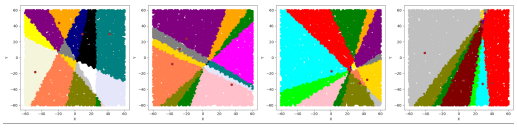
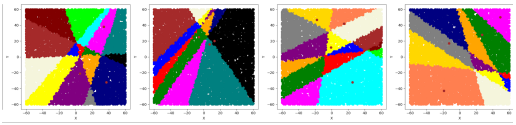
They are defined in my notebook as such:

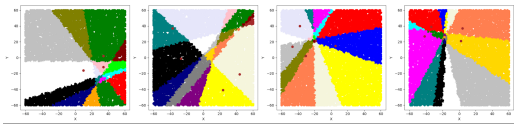
```python
# Random Dictator Rule
def random_dictator(preferences):
    dictator = random.choice(preferences)
    return dictator[0]

# Plurality Voting Rule
def plurality(preferences, num_candidates):
    # Initialize vote counts
    vote_counts = [0] * num_candidates
    for ranking in preferences:
        vote_counts[ranking[0]] += 1
    # Candidate with the most votes wins
    winner = vote_counts.index(max(vote_counts))
    return winner

# Borda Count Voting Rule
def borda(preferences, num_candidates):
    # Initialize Borda scores
    scores = [0] * num_candidates
    for ranking in preferences:
        for index, candidate in enumerate(ranking):
            scores[candidate] += num_candidates - index - 1
    # Candidate with the highest Borda score wins
    winner = scores.index(max(scores))
    return winner
```

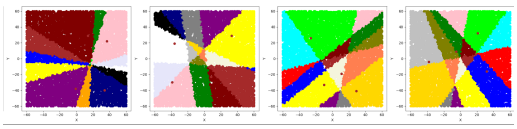
Now that we can calculate all of our data, I setup code to color each voter according to their preference profile ranking. To do this I first made a list containing all of the permutations for all the orderings of the candidate rankings. After I created a list of 24 colors (4!) so I could map voters to the color of their rankings. Using this dictionary, I regenerated the mapping by point and colored the voters to create a voronoi diagram divided by each ranking.

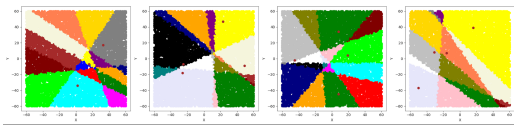
Results

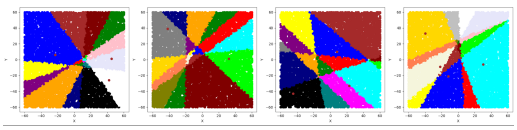
I ran this code over 100 trials so that I could plot the different voronoi digrams created. Using matplotlib, I created a figure that contains all the plots. The red/brown points in the regions represent the candidates!

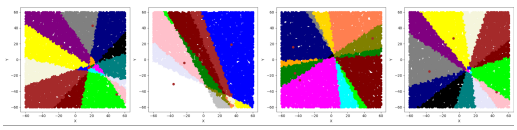
1. 
2. 
3. 

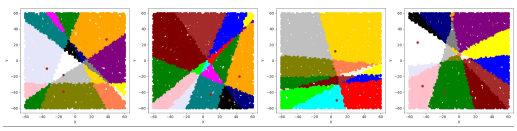
4. 

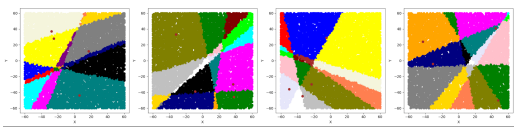
5. 

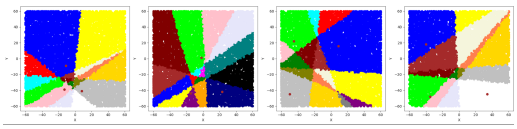
6. 

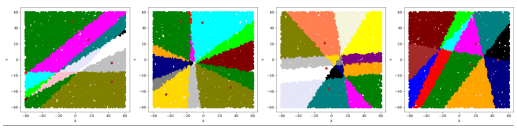
7. 

8. 

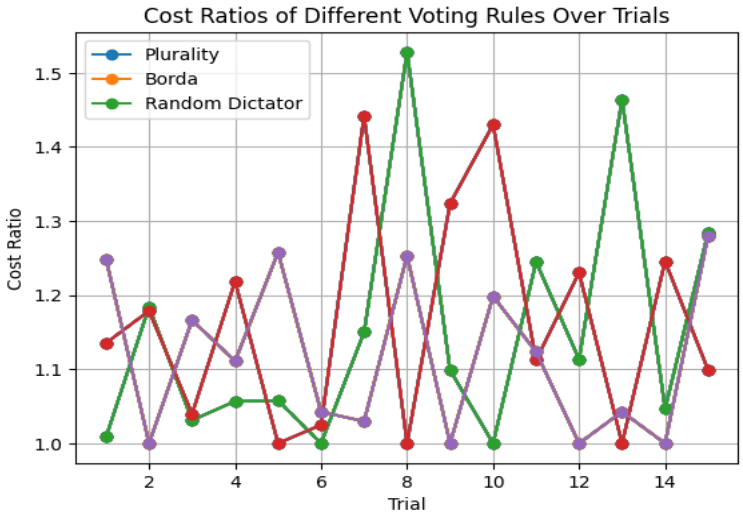
9. 

10. 

11. 

12. 

Using the cost ratios calculated during each epoch, I was also able to plot the ratios over each trial. For each election, we can see the most effective voting rule for each distribution of points



Over a sample of 15 trials we see that the plurality scores an average value of 1-1.3 on the cost ratio while Borda and Random Dictator reach a ratio of 1.4 and 1.5. The high peaks with those voting rules hint at randomness because they can choose very close and far from optimal. Plurality did not have any crazy peaks and maintained slightly neutral.

Conclusion

In this experiment, I compared three voting rules — Random Dictator, Plurality, and Borda — to see how closely each rule could pick a candidate that matches the "optimal" choice, minimizing the overall distance between voters and their selected candidate. I used Voronoi diagrams to visualize how voter preferences were distributed for different permutations of candidates.

The results show that the Plurality voting rule was the most consistent, with cost ratios ranging from 1 to 1.3. Borda sometimes got close to the optimal choice but had spikes up to 1.5, meaning it wasn't always reliable. Random Dictator was the least predictable, often picking candidates that were far from the best option.

Overall, Plurality voting provided steadier results and avoided extreme variations, while Borda could work well with more data but might be inconsistent. Random Dictator was too random to depend on. Moving forward, it would be interesting to investigate other voting rules or ways to combine rules to consistently reduce distortion and get closer to the ideal candidate selection.