Non-Iterative Pooled Ranking System Based on Borda Count with Tie Handling

Ektor Theoulakis

July 2025

1 System Tenets

This ranking system was designed to combine multiple candidate rankings per item under the following tenets:

1.1 Non-Iterative

A non-iterative algorithm computes the output directly through deterministic and bounded operations, avoiding any form of looping over the output space or convergence-based optimization. This ensures both speed and predictability in performance.

Mathematically, the result $\mathbf{r}$ is computed as a deterministic function of the inputs:

$$\mathbf{r} = f(\mathbf{R})$$

where $\mathbf{R} \in \mathbb{R}^{m \times n}$ represents m ranking vectors over n candidates.

1.2 Vectorized Operations Only

All computations are performed using matrix and vector operations. No for-loops or per-element iterations are used in critical parts of the computation. This ensures compatibility with high-performance numerical libraries such as NumPy and allows implicit parallelism.

1.3 Correct Handling of Equal Ranks (Ties)

In rank aggregation, tied rankings (e.g., two candidates ranked equally) should be treated fairly. This system uses **Borda count with tie-averaging**, where scores for tied candidates are averaged over their respective ranks.

For example, if candidates c_2 and c_3 are both ranked 2nd, they receive:

$$\text{Borda score} = \frac{n-2+n-3}{2}$$

where n is the number of candidates.

1.4 Social Utility Efficiency

The Borda count method is known to be **utility efficient** in positional voting theory. It maximizes the total utility assuming ranks can be interpreted as linear preferences. It ensures that if a candidate is consistently highly ranked across input rankings, they will score higher overall.

2 Rank Pooling Method Selection

The chosen method is the **Borda Count with Tie Handling**, a positional voting method satisfying all four tenets.

Given:

- n candidate items,
- m ranking vectors for a specific item, where each vector $\mathbf{r}_k \in \mathbb{N}^n$ assigns a rank to each candidate.

The method computes a pooled ranking $\mathbf{r}_{\text{pooled}}$ as follows:

1. For each ranking $\mathbf{r}_k$, compute a score vector $\mathbf{s}_k$ such that:

$$s_{k,i} = \text{average}(n - r_{k,j} \mid r_{k,j} = r_{k,i})$$

   This handles ties by assigning equal average Borda points.

2. Sum and normalize the scores:

$$\mathbf{s}_{\text{total}} = \frac{1}{m} \sum_{k=1}^{m} \mathbf{s}_k$$

3. Rank candidates by sorting $\mathbf{s}_{\text{total}}$ in descending order. The final rank of candidate i is:

$$r_{\text{pooled},i} = \text{rank}_{\text{desc}}(s_{\text{total},i}) + 1$$

3 Software Documentation

The software implementation is organized around a single core function:

Function: `borda_pool`

- **Signature:**

```
def borda_pool(ranking_vectors: List[np.array]) -> np.array
```

- **Purpose:** Takes a list of ranking vectors (for one item), each vector representing the ranks assigned to n candidate items in one source, and returns a single pooled ranking.
- **Inputs:**
  - `ranking_vectors`: A list of 1D NumPy arrays, each of shape $(n,)$.
- **Steps and Related Tenets:**
  1. **Tied Rank Handling (Tenet 3)** For each rank vector, identify tied candidates and compute the average Borda score across those candidates.
  2. **Vectorized Borda Calculation (Tenets 1 and 2)** Using NumPy masking and aggregation to compute score vectors without loops.

3. **Social Utility Efficiency (Tenet 4)** The average Borda scores reflect the relative preference strength across all sources.
4. **Final Ranking (Tenets 1 and 2)** The final rank is obtained using:

   `argsort(argsort(-scores)) + 1`

   ensuring 1-based ranking where lower values mean higher preference.

- **Output:**
  - A NumPy array of pooled ranks, shape $(n,)$, representing the aggregated ranks of each candidate item.

4 Example Visualization

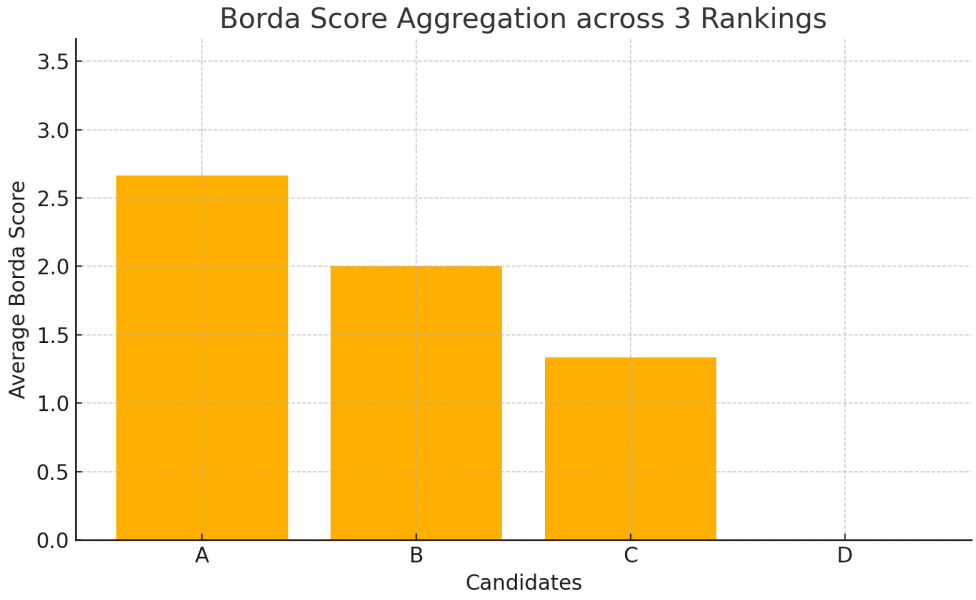

Figure: Borda score aggregation across three different rankings of four candidates.

5 Formal Properties and Proofs

5.1 1. Social Utility Efficiency

The Borda method selects candidates with the highest total utility under a linear utility model.

Theorem: If each voter's rank can be mapped to a linear utility function $u(i) = n - r(i)$, then the Borda winner maximizes the sum of individual utilities:

$$c^* = \arg\max_{c \in C} \sum_{v \in V} u_v(c)$$

Proof: Each rank $r(i)$ contributes $n - r(i)$ to the total Borda score. Summing across all voters is equivalent to summing utility values. Hence, the candidate with the highest Borda score maximizes the sum of utilities.

5.2 2. Neutrality and Anonymity

The Borda count is neutral (independent of candidate identity) and anonymous (independent of voter identity).

Proof: Permuting candidates or voters does not change the aggregated score calculation since scores depend only on ranks, not identifiers.

5.3 3. Resistance to Ties

Let multiple candidates share the same rank in a vote. The method assigns the average Borda score to all tied candidates:

$$\text{score}_i = \frac{1}{|T|} \sum_{j \in T} (n - r_j)$$

which ensures fairness and continuity under minor rank adjustments.

6 Limitations of Borda Count

Despite its simplicity and utility efficiency, Borda count has several well-known issues:

6.1 1. Spoiler Effect

Adding a candidate similar to an existing one can change the outcome unfavorably. This violates the *independence of irrelevant alternatives*.

Example: If candidate A is preferred over B, introducing a similar candidate A' may split A's points, allowing B to win.

6.2 2. Susceptibility to Strategic Voting

Voters may manipulate their rankings to demote strong competitors by ranking them artificially low. This is due to Borda's positional nature.

6.3 3. Lack of Condorcet Consistency

The Borda winner may not be the Condorcet winner (who beats every other candidate in pairwise comparison). This means the method can contradict majority preferences in some configurations.

7 Conclusion

The Borda count with tie handling offers a non-iterative, vectorized, and socially efficient method for rank aggregation. While it performs well under many practical constraints, awareness of its limitations is essential for critical applications.

8 Conclusion

This system provides an efficient, interpretable, and vectorized approach to aggregate candidate rankings across multiple sources. By satisfying the four outlined tenets, it guarantees robustness, scalability, and fairness in scenarios requiring rank fusion.