

Smart Discount Allocation Engine for Red Health

Problem Statement

Red Health provides discounts to patients based on the sales performance of its agents. The total discount budget ("siteKitty") must be distributed among agents based on multiple performance metrics. The goal is to ensure fair, data-driven, and explainable allocation across any number of agents without manually assigning weights to metrics.

Solution Overview

To address this, we built a robust and fully automated allocation engine using:

- Entropy Weighting Method to derive the importance of each metric from the data.
 - TOPSIS (Technique for Order Preference by Similarity to Ideal Solution) to rank agents based on closeness to the ideal performer.
 - JavaScript (Node.js) to implement a modular, scalable, and dynamic engine.
-

Methodology

Step 1: Input Parsing

- Load input.json containing:
 - siteKitty: total discount budget to be distributed
 - salesAgents: array of agent performance data
- Dynamically extract all metrics from the first agent object (except id).

Step 2: Data Normalization

- Normalize each metric using vector normalization.
- For cost-type metrics (e.g., lateDeliveries), invert the value so that lower is better.

Step 3: Entropy Weight Calculation

- Use entropy formula to compute the variability of each metric.
- Assign higher weights to metrics with greater discriminatory power.
- Normalize the weights so they sum to 1.

Step 4: Apply Weights

- Multiply each normalized metric by its entropy weight.

- Create a weighted normalized matrix.

Step 5: Determine Ideal and Anti-Ideal Solutions

- Ideal solution: highest (best) values across all metrics.
- Anti-ideal solution: lowest (worst) values across all metrics.

Step 6: Calculate Closeness Score (TOPSIS)

- For each agent:
 - Calculate distance from ideal and anti-ideal solutions.
 - Compute closeness score = $D^- / (D^+ + D^-)$






Step 7: Allocate Discounts

- Use closeness scores to proportionally distribute siteKitty.
- Round down each allocation.
- Add leftover to the last agent to ensure the total matches exactly.

Step 8: Generate Justification

- Use closeness score to assign a qualitative justification:
 - ≥ 0.8 : Exceptional overall performance
 - ≥ 0.65 : Strong performance
 - ≥ 0.5 : Moderate performer
 - < 0.5 : Needs improvement

Features

-  Fully dynamic: works with any number of agents and metrics.
-  Data-driven: entropy weighting eliminates manual tuning.
-  Fairness: uses a multi-objective decision model (TOPSIS).
-  Justifications included for transparency.
-  No external libraries required (pure JavaScript).

Technologies Used

- Node.js (JavaScript)
 - JSON for input/output
 - Console for debugging and verification
-

Outcome

- Input: A JSON file of agents and performance data.
- Output: An output.json file with each agent's assigned discount and justification.

Example:

```
{
  "allocations": [
    {
      "id": "A3",
      "assignedDiscount": 3259,
      "justification": "Exceptional overall performance"
    },
    {
      "id": "A5",
      "assignedDiscount": 3167,
      "justification": "Exceptional overall performance"
    }
  ]
}
```

Conclusion

This project solves the discount allocation problem in a scalable, interpretable, and mathematically sound way. It ensures agents are rewarded fairly based on real performance data and can handle changes in the number of agents or criteria without any code changes.