Bayesian Mastery Estimation Algorithm for OGE Exam Preparation

Abstract

This document outlines a Bayesian algorithm designed to estimate student mastery probabilities for skills, topics, and problem number types in preparation for the OGE math exam. The algorithm leverages performance data, incorporating Bayesian updating, difficulty weighting, time decay, and sequential analysis to provide accurate and dynamic mastery estimates.

1 Overview

The OGE exam comprises 25 problems, each associated with specific topics (e.g., problem 11: correspondence between algebraic formulas and graphs, problem 10: simple probability). Problems are characterized by:

- A list of required **skills** (e.g., [38, 95, 180, 5, 67]).
- A list of **topics** (broader categories encompassing skills).
- A **problem number type** from the set $\{-1, 1, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25\}$, where -1 represents a unique problem type.

Student performance is recorded in a PostgreSQL `activity` table, capturing attempt details such as correctness, duration, and scores. The algorithm models mastery probabilities for skills, topics, and problem types, using sequential analysis to detect performance trends and adjust estimates dynamically.

2 Algorithm Description

2.1 Modeling Skill Mastery

Skills are foundational units of the OGE syllabus, with each problem requiring a subset of skills. Mastery probabilities are estimated using a Bayesian framework.

2.1.1 Representation

- For each skill i, the mastery probability p_i is modeled as a random variable with a Beta distribution: $p_i \sim \text{Beta}(\alpha_i, \beta_i)$.
  - α_i: Represents "successes" (evidence of mastery).
  - β_i: Represents "failures" (evidence of non-mastery).

Why Beta Distribution? The Beta distribution is used for the following reasons:

1. **Models Probabilities**: Naturally represents probabilities (values between 0 and 1), ideal for estimating mastery likelihoods.
2. **Conjugate Prior**: Simplifies Bayesian updates, as the Beta distribution is a conjugate prior for the Bernoulli/binomial likelihood.
3. **Flexibility**: Captures a wide range of uncertainty levels via parameters α and β.
4. **Interpretable Parameters**: α and β represent "successes" and "failures".
5. **Uncertainty Quantification**: Provides variance and confidence intervals, enabling robust mastery thresholds (e.g., $P(p_i > 0.7) > 0.95$).
6. **Incremental Updates**: Supports efficient, real-time updates by incrementing α or β.

Beta Distribution Basic Facts

- **Definition**: A continuous probability distribution defined on $[0, 1]$, ideal for modeling probabilities or proportions.
- **Parameters**: α (successes) and β (failures), both positive, shape the distribution.
- **Probability Density Function**:

$$f(x) = \frac{x^{\alpha-1}(1-x)^{\beta-1}}{B(\alpha, \beta)}, \quad 0 \leq x \leq 1$$

  where $B(\alpha, \beta)$ is the beta function.
- **Mean**:

$$\mu = \frac{\alpha}{\alpha + \beta}$$

- **Variance**:

$$\text{Var} = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

- **Conjugate Prior**: Conjugate to the Bernoulli/binomial distribution, simplifying Bayesian updates.
- **Shapes**: Flexible shapes (e.g., uniform for $\alpha = \beta = 1$, U-shaped, skewed).
- **Applications**: Used in Bayesian inference, A/B testing, and modeling uncertainty in proportions.

2.1.2 Initialization

Each student takes a diagnostic test upon registration, classifying each skill as **weak** or **non-weak**. Beta parameters are assigned as follows:

```python
def initialize_skill_prior(is_weak: bool) -> tuple[float, float]:
    return (1, 40) if is_weak else (5, 45)
```

- Weak skills: Initial mastery probability ≈ 0.02.
- Non-weak skills: Initial mastery probability ≈ 0.1.

Note: For each skill i, topic t, and problem type T, store pairs (α_i, β_i), (α_t, β_t), and (α_T, β_T) in a PostgreSQL table `student_mastery` for each `user_id`, updated after each attempt.

2.1.3 Update Rule

For each attempt in the `activity` table:

- Query the `skills` list for the problem (via `question_id`).
- If `finished_or_not = True` and `is_correct = True`:
  - Compute current mastery probability $p_i = \frac{\alpha_i}{\alpha_i + \beta_i}$.
  - Determine velocity factor f:
    - If $p_i < 0.2$: $f = 2$.
    - Else if $p_i < 0.5$: $f = 0.725$.
    - Else if $p_i < 0.7$: $f = 0.6667$.
    - Else: $f = 0.5$.
  - Increment α_i by $f \cdot w(d)$ for each skill i in the problem's skill list.
- If `finished_or_not = True` and `is_correct = False`, increment β_i by $w(d)$.
- If `finished_or_not = False` (skipped):
  - If difficulty ≥ 3, increment β_i by $w(d)$.
  - If difficulty ≤ 2, increment β_i by $0.5 \cdot w(d)$.

Difficulty Weighting ($w(d)$)

- **Linear Scaling**: $w(d) = d$. Example: A difficulty-5 problem contributes $5\times$ more than a difficulty-1 problem.
- **Exponential Scaling**: $w(d) = 2^{d-1}$. Prioritizes mastery of advanced problem types.
- **Goal**: Harder problems provide stronger evidence of mastery or gaps.

Sensitivity Tests

- After 3 failures on a difficulty-5 problem, no skill's mastery probability should drop by more than 0.3.
- Solving 5 difficulty-1 problems correctly should increase mastery by less than 0.2.

2.1.4 Mastery Estimate

The probability of mastering skill i is:

$$p_i = \frac{\alpha_i}{\alpha_i + \beta_i}$$

Mastery is achieved if $P(p_i > 0.7) > 0.95$, accounting for uncertainty (e.g., Beta(2,18) vs. Beta(20,180)).

2.1.5 Dependency Graph Propagation

- If a student struggles with a skill (e.g., skill 2: "Scientific notation", with dependencies [1, 39]), penalize both the skill and its prerequisites.
- Use Bayesian Knowledge Tracing (BKT) weights.
- Precompute prerequisite weights: Penalty $= \frac{1}{1+\text{graph\_distance}}$.

2.1.6 Difficulty-Weighted Bayesian Updates

Adjust updates based on problem difficulty $d \in \{1, 2, 3, 4, 5\}$:

$$\alpha_i{+}= w(d) \cdot \text{correct}, \quad \beta_i{+}= w(d) \cdot \text{incorrect}$$

2.2 Modeling Topic Mastery

Topics are broader categories encompassing multiple skills, with mastery derived from skill mastery.

2.2.1 Representation

For each topic t, the mastery probability p_t is a function of associated skills' mastery probabilities.

2.2.2 Skill-to-Topic Mapping

A predefined mapping links skills to topics (e.g., skills [38, 95] belong to "Linear Equations").

2.2.3 Aggregation Method

Compute p_t as the average mastery probability of skills S_t:

$$p_t = \frac{1}{|S_t|} \sum_{i \in S_t} p_i = \frac{1}{|S_t|} \sum_{i \in S_t} \frac{\alpha_i}{\alpha_i + \beta_i}$$

Alternative options:

- **Minimum**: $p_t = \min_{i \in S_t} p_i$.
- **Product**: $p_t = \prod_{i \in S_t} p_i$.

The average is recommended for simplicity and robustness.

2.2.4 Update

Recalculate p_t whenever skill mastery probabilities are updated.

2.3 Modeling Problem Number Type Mastery

Problem number types (e.g., 11 for algebraic formulas and graphs) are modeled independently to assess type-specific proficiency.

2.3.1 Representation

For each problem number type $T \in \{-1, 1, 6, \ldots, 25\}$, model the probability p_T as $\text{Beta}(\alpha_T, \beta_T)$.

2.3.2 Initialization

- Use $\text{Beta}(1, 25)$ for general types.
- For higher-difficulty types, use $\text{Beta}(1, 40)$.

2.3.3 Update Rule

For each attempt where `problem_number_type = T` and difficulty $d \in \{1, 2, 3, 4, 5\}$:

- If `finished_or_not = True` and `is_correct = True`:
  - Compute $p_T = \frac{\alpha_T}{\alpha_T + \beta_T}$.
  - Determine velocity factor f:
    - If $p_T < 0.2$: $f = 2$.
    - Else if $p_T < 0.5$: $f = 0.725$.
    - Else if $p_T < 0.7$: $f = 0.6667$.
    - Else: $f = 0.5$.
  - Increment α_T by $f \cdot w(d)$.
- If `finished_or_not = True` and `is_correct = False`, increment β_T by $w(d)$.
- If `finished_or_not = False`:
  - If difficulty ≥ 3, increment β_T by $w(d)$.
  - If difficulty ≤ 2, increment β_T by $0.5 \cdot w(d)$.

Difficulty Weighting ($w(d)$)

- **Linear Scaling**: $w(d) = d$.
- **Exponential Scaling**: $w(d) = 2^{d-1}$.

2.3.4 Mastery Estimate

The probability of mastering problem type T is:

$$p_T = \frac{\alpha_T}{\alpha_T + \beta_T}$$

2.4 Change Detection with CUSUM

2.4.1 Description

Use the Cumulative Sum (CUSUM) method to detect significant shifts in performance.

2.4.2 Implementation

Track a cumulative sum for each skill i or problem type T:

$$S_n = \max(0, S_{n-1} + (x_n - k))$$

where:

- $x_n = 1$ if correct, 0 if incorrect.
- k is a reference value (e.g., 0.5).
- $S_0 = 0$.

Set thresholds (e.g., $h = 3$):

- If $S_n > h$, boost α_i or α_T by 0.5.
- If S_n drops below $-h$, increase β_i or β_T.

2.4.3 Benefit

Quickly identifies turning points in performance, enabling proactive adjustments.

2.5 Sequential Probability Ratio Test (SPRT)

2.5.1 Description

Implement SPRT to test mastery using accumulating evidence.

2.5.2 Implementation

Incorporate a time decay factor:

$$w(t) = \exp(-\lambda \cdot (t_{\text{current}} - t_{\text{attempt}}))$$

where λ is a decay rate (e.g., 0.01 per day). For skill i or type T:

- Query `answer_time_start` for the most recent attempt.
- Multiply α and β by $w(t)$.
- Define hypotheses:
  - H_0: $p_i < 0.7$ (not mastered).
  - H_1: $p_i \geq 0.7$ (mastered).
- Compute likelihood ratio:
$$\Lambda_n = \prod_{j=1}^{n} \frac{P(x_j|H_1)}{P(x_j|H_0)}$$
- Set thresholds $A = 0.05$, $B = 20$:
  - If $\Lambda_n \leq A$, accept H_0.
  - If $\Lambda_n \geq B$, accept H_1.
  - Otherwise, continue collecting data.

2.5.3 Benefit

Provides a rigorous method to confirm mastery, reducing uncertainty.

2.6 Incorporating Additional Data (Optional Enhancements)

2.6.1 Duration (`duration_answer`)

- Longer times indicate struggle. Weight α_i or α_T (e.g., +0.5) if `duration_answer` exceeds the median time.
- If insufficient data, use a default threshold (e.g., 300 seconds).

2.6.2 Scores (`scores_fipi`)

For problem types 20–25:

- Score = 0: $\beta + 1$.
- Score = 1: $\alpha + 0.5$.
- Score = 2: $\alpha + 1$.

3 Implementation Notes

- **Database**: Assumes a PostgreSQL `activity` table with fields `attempt_id`, `user_id`, `question_id`, `answer_time_start`, `finished_or_not`, `is_correct`, `duration_answer`, `scores_fipi`, `skills`, `topics`, `problem_number_type`. Store (α, β) pairs in `student_mastery`.
- **Storage**: Store α, β, and timestamps in `student_mastery`.
- **Tuning**: Adjust λ, $w(d)$, and thresholds empirically.
- **Scalability**: Use caching for efficient time decay calculations.

4 Conclusion

This algorithm provides a robust, probabilistic framework for tracking student progress in OGE exam preparation, balancing accuracy, responsiveness, and computational efficiency.