Tensor-based Scoring Model for Cognitively Diagnostic Networks

Russell G. Almond

Educational Psychology and Learning Systems, FSU

Scoring as a service

In the game *Physics Playground*, game engine sends information to scoring server to be scored.

The game instance (on players computer) polls the scoring server for current scores:

1. For adaptive sequencing of game levels
2. To display learning progress to player.

Goal is to get a rapid scoring algorithm which provides real-time performence either embedded with the game or on a server.

Four Processes

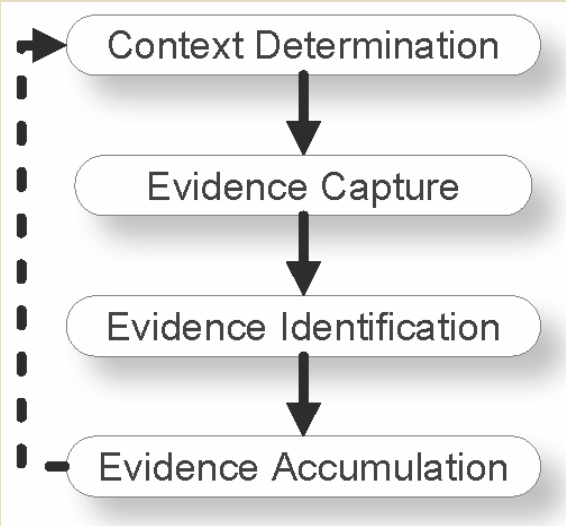


Figure 1: Four Process Architecture

Context Determination (aka Activity Selection) Determines when to stop and suggests next activity

Evidence Capture (EC) (aka Presentation) Captures student work product and process

Evidence Identification (EI) Summarizes work product and process what happens within a scoring task, producing *observable* outcomes.

Evidence Accumulation (EA) Estimates student proficiency across many scoring contexts, producing *statistics*

Scoring Engine (EA) Requirements

`studentBegins(uid)` A new student (identified by `uid`) has started the assessment, create a new stundent record for that student.

`addEvidence(uid,tid,y)` Observables with value `y` have arrived for Student `uid` for Task `tid`, update the student record.

`reportStatistics(uid)` Request for current estimates of student proficiency for Student `pid`.

`retractEvidence(uid,tid)` Remove the evidence from Task `tid` from Student `uid`'s student record.

`replaceEvidence(uid,tid,ynew)` Replace the evidence from Task `tid` in Student `uid`'s student record with new values `ynew`.

General Bayesian Psychometric model

Proficiencies $\boldsymbol{\theta}_i = (\theta_{i1}, \ldots, \theta_{iK})$ Student i's value for proficiency variables 1 through K.

Observables y_{ij} scored response (output of EI) from Scoring Context j by Student i

Observed Tasks J'

$$P(\boldsymbol{\theta}_i | \boldsymbol{y}_{iJ'}) = \frac{1}{C} P(\boldsymbol{\theta}_i) \prod_{j \in J'} P(y_{ij} | \boldsymbol{\theta}_i)$$

Proficiency Model $P(\boldsymbol{\theta}_i)$

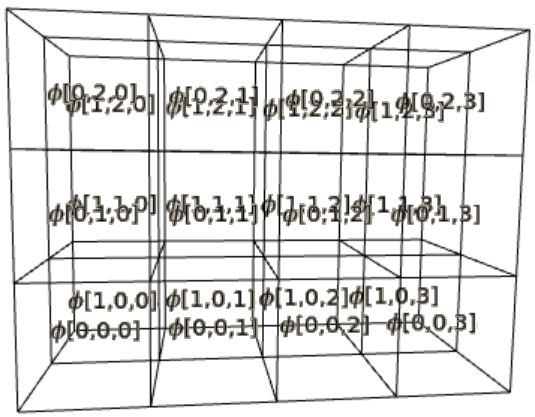
Evidence Model for Scoring Context j $P(y_{ij} | \boldsymbol{\theta}_i)$

Potentials and Tensors

Let the domain of θ_{ik} be $\{0, \ldots, M_k - 1\}$.

Then $P(\boldsymbol{\theta}_i) = \phi_0[\cdot]$ and $P(y_{ij}|\boldsymbol{\theta}_i) = \phi_j(y_{ij})[\cdot]$ can be stored in an $M_1 \times \cdots \times M_K$ array or *tensor*.

- A tensor with all non-negative elements is a *potential*,
  - Potentials represent unnormalized probability distributions or likelihoods.
  - Normalizing a potential produces a probability distribution.



Notes on size

- More than 10 competency variables is impractical because of the long test length need for reliability.
- Haberman (2005) suggests a 5-level ordered latent class model gives a good approximation to 2PL IRT model.
- 10 variables with 6 levels each, and 9 variables with 7 levels each both take less than 0.5 GiB
- Modern GPUs have 8 to 24 GiB

Combination, Convolution, and Normalization

- Normalizing a potential allows interpreting it as a probability distribution.

Normalization Constant

$$C = \sum_{\theta_{i1}} \cdots \sum_{\theta_{iK}} \phi[\theta_{i1}, \ldots, \theta_{iK}]$$

Normalization, $\|\phi\|$

$$\|\phi\|[\theta_{i1}, \ldots, \theta_{iK}] = \frac{1}{C}\phi[\theta_{i1}, \ldots, \theta_{iK}]$$

Convolution, $\phi_A \odot \phi_B$ (Elementwise multiplication, Hadamard product)

$$(\phi_A \odot \phi_B)[\theta_{i1}, \ldots, \theta_{iK}] = \phi_A[\theta_{i1}, \ldots, \theta_{iK}]\phi_B[\theta_{i1}, \ldots, \theta_{iK}]$$

Combination, $\phi_A \otimes \phi_B$, Convolution followed by normalization

Frames of Discernment and Projection

- Let $q_{Ak} = 1$ if dimension k is relevant in Potential A, and 0 if not.
  - This is the *frame of discernment*.
- If $q_{Ak} = 0$, then $\phi_A[\theta_{i1}, \ldots, \theta_{ik}, \ldots, \theta_{AK}$ is the same for all values of θ_{ik}.
- Can save space by setting $M_k = 1$ for this dimension in ϕ_A.
- Can *extend*, move to a bigger frame $q_{Ak} = 0$ to $q_{A'k} = 1$, by replicating over the new dimension.
- Can *marginalize*, move to a smaller frame $q_{Ak} = 1$ to $q_{A'k} = 0$, by summing over the smaller dimension.

Optimizations

In `TensorFlow` convolution and projection to a bigger frame is done automatically at the same time.

Fixing the order of the proficiency variables avoids time reshaping the array.

When combining many potentials, combine the one with the same frames first.

Can also order so that only need extend over as few variables as possible.

Scores and Statistics

- At any point in time, $P(\boldsymbol{\theta}_i|\boldsymbol{y}_{iJ'}) \propto \phi_0 \odot \bigodot_{j \in J'} \phi_j(y_{ij})$
- The normalization constant, C, is the *prior predictive probability* for the observed outcomes, $\boldsymbol{y}_{iJ'}$.
  - Outlier detection and model checking
- Most scores are functionals of $P(\boldsymbol{\theta}_i|\boldsymbol{y}_{iJ'})$
  - Usually single margin $P(\theta ik|\boldsymbol{y}_{iJ'})$

$$\mathrm{MAP}(\theta_{ik}) = \mathrm{argmax}_{\theta_{ik}} P(\theta_{ik}|\boldsymbol{y}_{iJ'})$$

Assign numeric values, w_{mk}, to states:

$$\mathrm{EAP}(\theta_{ik}) = \frac{1}{M_k} \sum_{m=1}^{M_k} w_{mk} P(\theta_{ik} = m|\boldsymbol{y}_{yJ'})$$

Scoring Engine Operation

1. Prebuild Proficiency and Evidence Tensors
2. Create initial Student Model
3. Update With Evidence
4. Calculate Statistics
5. Retract/Replace Evidence

Preparation: Prebuild Potentials

Assuming memory is cheap and computation speed is critical, precalculate and save the tensors.

- The *proficiency potential*, $\phi_0[\cdot]$ is a tensor over $\mathbf{\Theta}$.
- For each Task (item) j, there is an *evidence potential* over $Y_j \times \mathbf{\Theta}$.
  - As the observable outcome Y_j is the first dimension, it varies slowest.
  - Potentials representing $P(y_{ij}|\boldsymbol{\theta}_i)$ is a continguous block of memory.

Making a New Student Record

1. Copy the proficiency potential, ϕ_0, to make the initial student model for the student.

$$\phi_{i\emptyset} \leftarrow \phi_0$$

2. Make an empty list to store the evidence messages.

- Messages have watermarks (timestamped labels)

3. (Optional) Copy the initial values for the statistics to the cache.

Absorbing Evidence from a Scoring Context

Message is Learner i attempted Task j and got outcome y_{ij}.

1. Fetch the current potential for Learner i, ϕ_{iJ}.
2. Fetch the evidence potential for Task j and select the sub-potential corresponding to $Y_j = y_{ij}$, $\phi_{j|y_{ij}}$.
3. Add the evidence message from Task j' to the list of evidence for the tasks in J, making a new evidence set J'
4. Update the studnet potential through covolution:

$$\phi_{iJ'} \leftarrow \phi_{iJ} \odot \phi_{j|y_{j'k}}$$

5. (Optional) Calculate Reporting Statistics

Reporting Statistics

Two strategies for calculating statistics:

1. Only calculate statistics when requested (at end of assessment or on demand).
2. Calculate statistics after each update and cache in database.

- Can respond quickly to request for statistics
  - Although may be incomplete
- Can store history of how statistics change with evidence
  - Evidence Balance Sheet

Revisions and Updates

- To Pin (remove evidence from) a Task divide by the evidence potential

$$\phi_{iJ'} \leftarrow \phi_{iJ} \odot \frac{1}{\phi_{j|y_{jk}}}$$

- To update the evidence from a Task (rescoring) divide by the old evidence and replace with the new observable y'_{jk}

$$\phi_{iJ'} \leftarrow \phi_{iJ} \odot \frac{1}{\phi_{j|y_{jk}}} \odot \phi_{j|y'_{jk}}$$

- Use watermarks to eliminate out of sequence messages.

Learning Parameters from Data

- Not a part of real time scoring

EM Algorithm:

E-step Score students with current parameters, and make tables of expected counts.

M-step Use gradient decent algorithm to update model parameters

Discussion

- Rewritten scoring algorithm uses tensors
  - Fast: trades storing precalculated potentials for speed
  - Can exploit GPU computation (`CUDA`, `TensorFlow`, `PyTorch`).
  - Work can be factored across multiple processors in cluster computing
- Works with any fully Bayesian scoring model
  - Requires prior (population) distribuiton over proficiency variables
  - Provides real-time scoring algorithm
  - Models with continuous proficiencies can be approximated with discrete models
- Can be embedded in real-time scoring
  - `TensorFlow Lite` uses only precalculated weights, designed to work on phones, in browsers and in other lightweight computing settings.
- Can use watermarks to ensure data integrity when scoring is done on a central server.

Thanks

https://ralmond.net/EvidenceStream

Size of the Tensor

Size of tensor is $\prod M_k$.

	M = 2	M = 3	M = 4	M = 5	M = 6	M
K = 8	2.0 KiB	51.3 KiB	512.0 KiB	3.0 MiB	12.8 MiB	44.0
K = 9	4.0 KiB	153.8 KiB	2.0 MiB	14.9 MiB	76.9 MiB	307.9
K = 10	8.0 KiB	461.3 KiB	8.0 MiB	74.5 MiB	461.3 MiB	2.1

E-Step:

- Stack the student model tensors to make a higher-dimensional tensor with Students as the extra dimension.
- Work of scoring students is associative and communitive, so can take advantage of cluster computing.
- Marginalizing across the students makes a cross-tab of expected counts for all proficiency profiles,

$$\Phi_0[\boldsymbol{\theta}] = \sum_i \phi_{iJ(i)}[\boldsymbol{\theta}]$$

- For evidence models, stack potentials by observed outcome:

$$\Phi_j[y, \boldsymbol{\theta}] = \sum_{i:y_{ij}=y} \phi_{iJ(i)}[\boldsymbol{\theta}]$$

TensorFlow Modules and Models

A `TensorFlow` module has:

a) A function for creating a tensor (potential) from a set of weights (parameters), $\phi_j(\eta_j)$

b) Current values for the weights, η_j

c) An *objective function* to maximize when learning the weights.

$$\|\Phi_j \odot \log\left(\phi_j(\eta_j)\right)\| = \sum_{\boldsymbol{\theta} \in \boldsymbol{\Theta}} \Phi_j[\boldsymbol{\theta}] \log\left(\phi_j(\eta_j)[\boldsymbol{\theta}]\right)$$

- `TensorFlow` will do gradient decent learning, and save the weights.