Datathon

Hao Guo

February 2026

1 Model

1.1 Assumptions

- Single global half-life parameter for all user and all words (can be adjusted to improve power)
- independent factorization of difficulty via language, type of speech, and word length
- Approximate memory decay outside of Beta distribution

1.2 Derivation

The model we use to answer the question:

What is the probability $\mathbb{P}$ that the user U still remembers the word W in language L, given W is in type T with length l after time t, where

$$T \in \{noun, verb, number, adjective, pronoun, preposition, adverb\},$$

and

$$L \in \{English, Spanish, German, French, Italian, Portuguese\}.$$

uses uses a Bayesian Beta–Binomial model and the posterior mean $\mathbb{P} = \frac{\alpha}{\alpha+\beta}$ as the recall probability.,

$$\mathbb{P} \to f(\mathbb{P} \mid x) = \frac{f(x \mid \mathbb{P})g(\mathbb{P})}{\int_{\mathbb{P}} f(x \mid \mathbb{P})g(\mathbb{P})d\mathbb{P}}$$

where $\mathbb{P} \sim Beta(\alpha, \beta)$.

Let α_p and β_p denote prior beliefs of α and β, then we can derive them as:

$$p_{prior} = \sigma(b_0 + b_L f(L) + b_T g(T) + b_l h(l))$$

where

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

and

$$\alpha_p = z \cdot p_{prior}$$
$$\beta_p = z \cdot (1 - p_{prior})$$

where f, g, h return difficulty related features (higher=easier), which are combined in a logistic regression to get the prior recall probability p_{prior}

$f(L)$ is computed based on studies FSI language difficulty,

$$f(L) = \epsilon + (1 - 2\epsilon)\Big[1 - \frac{H(L) - H_{min}}{H_{max} - H_{min}}\Big]$$

where ϵ is a tiny number that ensures $f(L) \in (0, 1)$, $H(L)$ returns the hours needed to learn language L.

$g(T)$ is retrieved from the study of Suranto [1], which describes the percentage of identified words in different word types for students.

$h(l)$ is estimated using a linear regression with rows with history_seen ≤ 3:

$$h(l) = \beta_0 + \beta_1 l$$

Once we have α_p and β_p, we obtain α and β by:

$$\alpha = \text{History correct} + \alpha_p$$
$$\beta = \text{History seen} - \text{History correct} + \beta_p$$

and $\mathbb{P}$ is calculated by:

$$\mathbb{P} = \frac{\alpha}{\alpha + \beta}$$

we can further apply a memory decay function

$$d(t) = 2^{-t/h}$$
$$t = \frac{delta}{3600}(\text{unit in hours})$$

with the half life model. Then the probability $\mathbb{P}_t$ of the user U remembering the word W after time t is calculated by:

$$\mathbb{P}_t = d(t) \cdot \mathbb{P}$$

2 Fitting

Once we have the model working, we need to fit our model, then test the correctness and evaluate. (How good is the model?)

2.1 Fit&Correctness

To fit our model, we use the following procedure:

1. randomly split users into train and test (say 80%:20%)
2. for each user, sort by timestamp (likely already done so)
3. use the train group to train the model, and test it with the test group

Then for each row i,

- n_i = session_seen
- m_i = session_correct

compute Binomial log-likelihood:

$$logY_i = m_i log\mathbb{P}_{ti} + (n_i - m_i)log(1 - \mathbb{P}_{ti})$$

then calculate the average negative log-likelihood (NLL):

$$NLL = -\frac{1}{N}\sum_i logY_i$$

The smaller the NLL, the better the model describes the data.

Parameters we can tune to improve the model:

- z: adjust the weight of prior beliefs
- h: half life parameter
- $\vec{b} = \{b_0, b_L, b_T, b_l\}$

We want:

$$\theta^* = (z^*, h^*, \vec{b^*}) = argmin_\theta NLL(\theta),$$

where $\theta = (z, h, \vec{b})$. We can use scipy.optimize.minimize.

2.2 Evaluation

We can compare our model to other common models. Namely:

- Half-Life Regression (HLR) (The ultimate model Duolingo uses)
- ACT-R Memory Model (A decent model)
- Logistic Regression / Neural Classifiers for Recall (We should better beat this)
- Simple Exponential Forgetting Curve (The baseline we must beat)

For testing, we can do a clustered sampling of our dataset (maybe 3 users per language) to speed up the evaluation.

We might not have time to implement these models, but we could mention them in the presentation.

3 Advantages and limitations

3.1 Limitations

- Likely not as accurate as the HLR model (obviously)
- Cannot fully utilize time data
- The linear regression used in $h(l)$ may not predict well for words with length exceeding the longest word in the dataset.

3.2 Advantages

- Computationally cheap
- Does not require a lot of data
- Since it is structurally simply, modifications can be made easily.

4 Test results

The result is obtained by running model_stream.py with parameters:

- CHUNK_SIZE = 200_000 # The size of each chunk of data
- RANDOM_SEED = 42
- EPS = 1e-9 # to avoid log(0)
- EPS_F = 1e-3 # epsilon in f(L)
- USE_SUBSAMPLE_FOR_TRAINING = True
- TRAIN_SUBSAMPLE_ROWS = 1_000_000 # rows used per NLL eval while fitting
- MAXITER = 50 # optimizer iterations
- PRIOR_SAMPLE_MAX = 1_000_000 # sample size for estimating the prior
- half-life upper bound = 20000.0

Note that for performance improvements, $\vec{b}$ is optimized first, then (z, h) follows.

```
Pass 1: collecting unique users...
Total unique users: 115222
Train users: 92177, Test users: 23045
Pass 2: estimating h(l) = beta0 + beta1*l via streaming regression...
h(l): beta0 = 0.8880, beta1 = 0.0013  (from 3415409 rows)
Pass 3: sampling low-history rows to fit prior coefficients (b0, bL, bT, bl)...
```

```
Prior logistic regression sample size (before cleaning): 1046364
Prior logistic regression sample size (after cleaning): 1045821
Prior coefficients (from GLM):
b0 = 11.4914, bL = 0.3322, bT = 0.0086, bl = -10.1882
Pass 4: precomputing numeric feature file (this is a one-time cost)...
Feature file written to: precomputed_features.csv
Fitting z and half-life h on training users (using subsample for speed)...

Optimization result for z and h:
  message: CONVERGENCE: NORM OF PROJECTED GRADIENT <= PGTOL
  success: True
   status: 0
      fun: 0.5604880087063433
        x: [ 2.743e+01  1.560e+04]
      nit: 26
      jac: [ 1.588e-06 -6.661e-08]
     nfev: 90
     njev: 30
 hess_inv: <2x2 LbfgsInvHessProduct with dtype=float64>

Estimated z  = 27.4349
Estimated h  = 15604.8663 hours (half-life)

Computing final NLL on full train and test sets (this may take a bit)...

Final NLL (train): 0.565432
Final NLL (test) : 0.569647
```

h(l): beta0 = 0.8880, beta1 = 0.0013

$\beta_0 = 0.888$: baseline easiness at length 0

$\beta_1 = 0.0013$: easiness increases by this number per additional character

Users are slightly more likely to to get longer words correct.

b0 = 11.4914, bL = 0.3322, bT = 0.0086, bl = -10.1882

Different languages and word type barely have any effect. Word length seem to have huge effect, but it is mostly canceled by b0.

Estimated z = 27.4349

The prior behaves like 27 "unseen" trails.

h = 15604.8663 hours

Memory basically does not decay.

Final NLL (train): 0.565432
Final NLL (test) : 0.569647

tiny gap between train and test meaning the model is not overfitting.

NLL (test) implies a strong model with correctness = 0.743 (if we guess randomly at each trail (correctness = 0.5), we would get NLL = 0.693)

Overall the coefficients seem a bit weird, but the model works fine. Probably due to Duolingo's learning method, i.e. user can tap on the word and see translation.

As a comparison, HLR results in NLL = 1.052842, which is also weird since the model should not have behaved this bad.

All these cues may imply bugs in our code.

References

[1] Suranto and Yuspik. "The Analysis of Student's Ability to Identify Parts of Speech". In: *English Teaching and Applied Linguistics Journal* 1.1 (2024), pp. 18–26.