A Sleeping, Recovering Bandit Algorithm for Optimizing Recurring Notifications

Kevin P. Yancey
kyancey@duolingo.com
Duolingo
Pittsburgh, Pennsylvania

Burr Settles
burr@duolingo.com
Duolingo
Pittsburgh, Pennsylvania

ABSTRACT

Many online and mobile applications rely on daily emails and push notifications to increase and maintain user engagement. The multi-armed bandit approach provides a useful framework for optimizing the content of these notifications, but a number of complications (such as novelty effects and conditional eligibility) make conventional bandit algorithms unsuitable in practice. In this paper, we introduce the `Recovering Difference Softmax Algorithm` to address the particular challenges of this problem domain, and use it to successfully optimize millions of daily reminders for the online language-learning app Duolingo. This lead to a 0.5 % increase in total daily active users (DAUs) and a 2 % increase in new user retention over a strong baseline. We provide technical details of its design and deployment, and demonstrate its efficacy through both offline and online evaluation experiments.

CCS CONCEPTS

• **Computing methodologies** → **Sequential decision making**;
• **Mathematics of computing** → *Probability and statistics.*

KEYWORDS

Multi-Armed Bandits, Notification Content Optimization, Bayesian Approaches, Machine Learning

ACM Reference Format:
Kevin P. Yancey and Burr Settles. 2020. A Sleeping, Recovering Bandit Algorithm for Optimizing Recurring Notifications. In *Proceedings of the 26th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '20), August 23–27, 2020, Virtual Event, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3394486.3403351

1 INTRODUCTION

To keep users engaged and promote recurring usage, many mobile and online apps send regular email and push notifications to their users. For Duolingo — the world's most-downloaded language-learning app with more than 300 million users — this involves sending daily practice reminder notifications, such as those shown in Figure 1. The content of these notifications is generated from pools of hand-written templates (which may contain slots for user-specific information) from which a single template is chosen to generate the content of each notification.

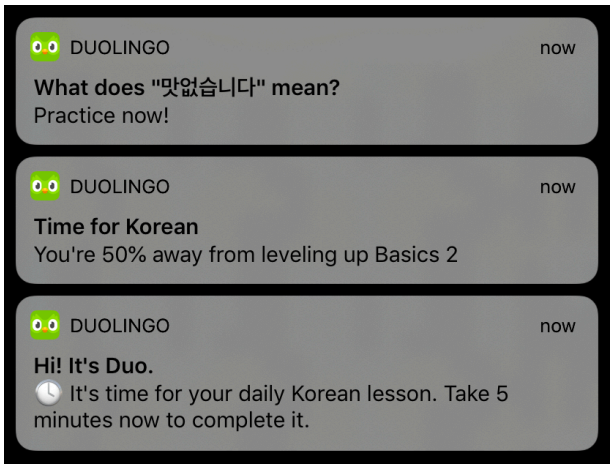

Figure 1: Example daily reminder notifications.

The multi-armed bandit problem provides a well-studied theoretical framework for optimizing the selection of templates in this context. However, two complications arise that makes conventional bandit algorithms unsuitable for our scenario:

(1) "Fresh" templates (which have not been recently seen) tend to have higher impact on user behavior. Thus, for good performance, it is essential that the bandit algorithm not only choose templates that perform well in general, but also vary the selection of templates for each user over time in order to exploit this novelty effect. Otherwise, the bandit will converge to using the the same template for a given user day after day, which will get repetitive, desensitize users, and fail to improve engagement in the long run.
(2) Many templates are based on conditions that are not applicable to all users. For example, some templates in Table 1 reference a user's "streak wager," a game mechanic applicable to only a fraction of users on any given day. As such, each template has defined eligibility criteria to prevent the template from being sent to an inappropriate user. However, the standard formulation of the multi-armed bandit problem assumes that all options are available at all times, and the violation of this assumption substantially complicates the evaluation of each option's performance.

KDD '20, August 23–27, 2020, Virtual Event, CA, USA

ACM ISBN 978-1-4503-7998-4/20/08.
https://doi.org/10.1145/3394486.3403351

Table 1: Example templates and their eligibility criteria.

Template	Eligibility Criteria
You're on fire Continue your 109 day Spanish streak on Duolingo.	*Must have a 3+ day streak.*[1]
Streak wager reminder You're on day 2 of your 7-day streak wager! Now get to day 3!	*Must have a streak wager.*[2]
Ready for your trip? Take 5 minutes to practice Italian now	*User's profile must indicate travel motivation.*

In this paper, we describe the `Recovering Difference Softmax Algorithm`, a novel variant of the bandit framework that addresses these two sub-problems. It has been deployed to select daily reminders for millions of users at Duolingo, resulting in substantial lifts in user engagement metrics.

1.1 Related Work

The multi-armed bandit problem [13] is an optimization problem in which an agent repeatedly chooses from among a fixed set of *arms* over some number of rounds. Some amount of *reward* is obtained each round based upon the arm chosen. In our setting, the arms represent different notification templates and the rewards are successful user engagements. Since an arm's reward can only be known by choosing it, there is a trade-off between *exploration* (choosing an arm to learn more about its expected reward) and *exploitation* (choosing the best arm based on current information to maximize reward). This formalization is applicable to a wide range of real-world problems and a number of algorithms for this basic formulation have been proposed and studied, including `ϵ-greedy` [16], `softmax` [17], and `Upper Confidence Bound (UCB)` [3].

Many variations of the multi-armed bandit problem exist, each making different assumptions that often require different solutions. The two variations that most directly relate to the recurring notifications setting studied in this paper are the "recovering bandits problem" [12] and the "sleeping bandits problem" [10].

In the recovering bandits problem, an arm's expected reward is given by an unknown function of the number of rounds since the arm was last chosen. Pike-Burke et al. [12] propose two bandit algorithms that use Gaussian processes to model the unknown reward functions. A number of other papers [6, 11, 15] explore a related problem, known as the "rested bandits problem," where an arm's expected reward depends on the number of times the arm has been selected previously (rather than the how recently it was last selected). In reality, both of these factors are likely relevant to addressing the novelty effect mentioned in the previous section. However, between the two, recency is probably more essential, since reminders sent months or even years before should have little impact on their effectiveness today, regardless of their quantity.

[1] "Streak" means the user has completed a lesson every day for some number of days.
[2] "Streak wager" is a gamification feature that awards the user extra in-app currency (i.e., "gems") for completing lessons on 7 consecutive days.

In the sleeping bandits problem, certain arms may be ineligible some rounds, as is the case with the templates in our recurring notifications setting. Kleinberg et al. [10] proposes a theoretical solution with proven regret-bounds by using the exp4 algorithm [2] to search for an optimal priority ordering of all arms in $\mathcal{A}$. However, since there exist $|\mathcal{A}|!$ possible orderings, this approach is intractable when there are more than a handful of arms. Furthermore, this approach isn't obviously compatible with the solutions addressing the recovering bandits problem.

In contrast, the `Recovering Difference Softmax Algorithm` provides a practical solution that solves both of these sub-problems simultaneously. Furthermore, while much of the bandit literature focuses mainly on proving theoretical performance bounds of algorithms, our paper provides both offline and online empirical evaluations in a real-world setting.

1.2 Problem Definition and Notation

In our recurring notification optimization setting, any notification that is sent to any user constitutes a single round. Following convention, individual rounds are denoted with the variable t.

Each arm corresponds to a template that can be chosen. $\mathcal{A}$ denotes the set of all arms, and $\mathcal{A}_t \subseteq \mathcal{A}$ denotes the set of eligible arms at round t.

Each round, a policy is applied to choose an arm, a_t, from among the eligible arms, $\mathcal{A}_t$. The policy that was applied for a given round, referred to in reinforcement learning as the "behavior policy" [17], is denoted b_t, which is a discrete probability distribution over $\mathcal{A}_t$. Hence, $b_t(a \mid t)$ is the probability that arm a would be selected at round t under the policy in place during round t. When we use the historical data to learn a new policy, we denote the new policy π.

The purpose of Duolingo practice reminders is to encourage users to complete at least one language-learning lesson per day, so our reward should reflect that. However, a substantial percentage of our completed lessons are organic: many users will complete a lesson whether we send a notification or not. It is not possible to reliably distinguish between organic lessons from notification-driven lessons, which adds a degree of noise when evaluating the performance of different arms. To minimize the amount of organic activity counted in our reward function, we only count lessons that are completed within two hours of the notification. Thus, we define the reward function, $r_t \in \{0, 1\}$, as follows: the reward is 1 if the user completes a lesson within two hours of the reminder being sent, and 0 otherwise.

2 RECOVERING DIFFERENCE SOFTMAX ALGORITHM

We now describe the `Recovering Difference Softmax Algorithm` in detail. The algorithm works by learning and periodically updating a policy, π, that attempts to maximize future rewards based on historical observations. For each arm, a set of prior rounds where that arm was eligible are gathered, denoted H_a, such that for each $t \in H_a$ the reward r_t and behavior policy b_t are known. These sets of historical rounds are used to evaluate the performance of each arm and build the policy π. This learned policy is applied in future rounds to choose an arm until the policy is updated again.

Table 2: Reward measures for example template arms.

Template	Avg. Reward When Used	Avg. Reward When Eligible	Rel. Diff.
A	0.2672	0.2682	-0.37 %
G	0.1372	0.1352	+1.48 %
H	0.1332	0.1362	-2.22 %

2.1 Sleeping Arms

As mentioned previously, not all arms are available every round. This property makes the problem more challenging because eligibility criteria can act as confounding variables with regard to the reward. For example, if arm A is available to only highly active users and arm G is available to all users, then the historical rounds where arm A was chosen will probably have much higher reward regardless of the arm's efficacy.

For example, we see in the first column of Table 2 that if we take the average reward when a given arm is chosen (independent of whether the other arms are eligible for those rounds), arm A has much higher average reward. However, we see in the second column that this difference is mostly explained by the fact the arms were eligible to distinct (but possibly overlapping) sets of rounds. In fact, when we compute the relative difference between the first two columns, we see that, in fact, arm G has much higher lift, and that arm A barely performs better than average. For example, consider a user who is eligible for both arms A and G: which arm should we show them? Among all users eligible for arm A, independent of whether they're eligible for G, we see a lift of $-0.37\,\%$, and among all users eligible for arm G, the lift is $+1.48\,\%$. These numbers indicate that for this user, we should prefer choosing arm G.

Hence, choosing the template with the highest average historical reward, such as is done by conventional bandit algorithms cited earlier (e.g., ϵ-`greedy` [16], `softmax` [17], and `Upper Confidence Bounds (UCB)` [3]) may yield poor results.

Furthermore, arms with new eligibility rules are introduced frequently. Thus, as a design constraint, we do not want the bandit algorithm to directly depend on the eligibility rules. Otherwise, the bandit algorithm implementation would need to be modified each time a new set of eligibility rules were put into production, greatly increasing the algorithm's maintenance cost.

We instead propose a method to measure the efficacy of each arm while controlling for the effects of its eligibility criteria. We do this by using the historical rounds, H_a, to estimate the typical reward when arm a is used, which we denote μ_a^+, and the typical reward when the arm a is *eligible* but a different arm is chosen instead, which we denote μ_a^-. We then compute a single performance metric from these by taking the relative difference (see Equation 1). Since both expectations are estimated from a sample of historical rounds that meet the same eligibility criteria, we effectively control for the effects of the eligibility criteria on the expected reward.

Since the probability of an arm being used varies from round to round depending on the policy in use at that time, collecting all historical instances where an arm was used will not produce a representative sample: it will be biased towards rounds where the arm was historically more likely to be used, such as rounds where there were fewer alternative arms. So, estimating μ_a^+ by computing a simple average of rewards from the rounds in H_a where arm a was selected will produce biased results. The same problem arises when estimating μ_a^-. We use weighted importance sampling to correct for this. Specifically, we define the estimates of these variables as follows:

$$\bar{\mu}_a^+ = \frac{\sum_{\{t \in H_a | a_t = a\}} w_t^+ r_t}{\sum_{\{t \in H_a | a_t = a\}} w_t^+}, \quad \text{where} \quad w_t^+ = b_t(a \mid t)^{-1}$$

$$\bar{\mu}_a^- = \frac{\sum_{\{t \in H_a | a_t \neq a\}} w_t^- r_t}{\sum_{\{t \in H_a | a_t \neq a\}} w_t^-}, \quad \text{where} \quad w_t^- = (1 - b_t(a \mid t))^{-1}$$

and where a_t is the arm chosen at round t.

From these two expected reward estimations, we compute the relative expected reward, referred to as the arm's *score*, as follows:[3]

$$\bar{s}_a = \frac{\bar{\mu}_a^+ - \bar{\mu}_a^-}{\bar{\mu}_a^-} \tag{1}$$

These scores can then be used to compare performance between arms with distinct eligibility criteria.

2.2 Small Sample-Size Arms

The estimated arm scores from the previous section will have high variance for arms with small sample sizes (i.e., small $|H_a|$). This variance can cause problems when these estimates are used to choose templates for future rounds. For example, in the case of very high estimates, the score may outweigh the recency penalty (described in the next section) enough that the arm will be used repeatedly for all eligible rounds until the arm's score converges to a more accurate estimation. Worse, an arm may never recover from an initially low score estimation because it may be so low that it is virtually never used, even with the exploration enabled by the use of `softmax` (described in Section 2.4). In this case, the bandit may take a long time to collect enough samples for that arm to correct its initially low estimation.

For example, Figure 2 plots the magnitudes of the scores for all the arms from the offline evaluation experiment described in Section 3.1. We can easily see that arms with small sample sizes tend to have higher magnitudes. Figure 3 shows the distribution of relative rewards for arms with a "large" effective sample size (defined here as being greater than 100,000). The distribution is approximately normal, with 90 % of the scores being within $\pm 3.4\,\%$. However, only 50 % of arms with smaller effective sample sizes come within this range, and 38 % dubiously have scores outside $\pm 5\,\%$, which could cause highly degenerate behavior in the bandit, if used as-is.

To regularize these scores, we use empirical Bayes estimation to shrink both $\bar{\mu}_a^+$ and $\bar{\mu}_a^-$ towards a Bayesian prior. Since both of these variables are estimates of a proportion[4], we define the prior

[3] One might instead choose to use the absolute difference, $\bar{\mu}_a^+ - \bar{\mu}_a^-$, instead of the relative difference. It's not necessarily clear which metric should be better in general. In our case, we evaluated both metrics in the offline evaluation and found the differences were negligible. As such, our decision to use relative difference was a pragmatic one: the relative differences were more intuitive when reporting arm scores to employees via dashboards.

[4] This is because our reward function is binary, and so the average reward is the proportion of positive outcomes. A different reward function may warrant a different prior distribution.

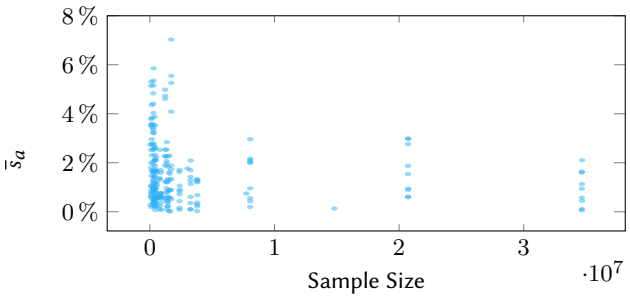

Figure 2: Scatter plot showing the absolute value of relative expected rewards $|\bar{s}_a|$ vs. the sample size from which each arm score was computed.

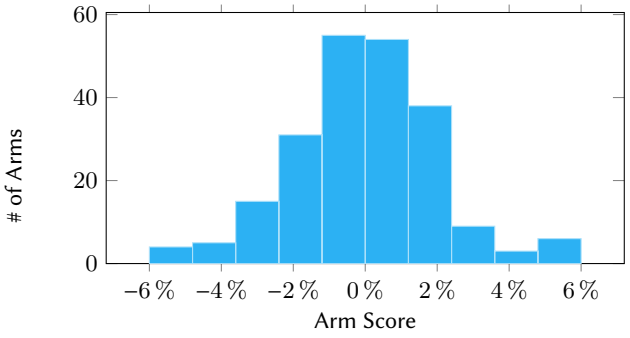

Figure 3: Histogram of arm scores with large effective sample sizes ($n > 10^5$).

using the beta distribution parameterized with σ and μ_a such that:

$$\mu_a^+, \mu_a^- \sim \text{Beta}(\mu_a/\sigma, (1-\mu_a)/\sigma)$$

where μ_a is the expected reward when arm a is eligible, which is estimated from each arm, i.e.:

$$\mu_a = \frac{\sum_{\{t \in H_a\}} r_t}{|H_a|}$$

Here σ represents how much the expected reward varies depending upon which arm is chosen. It can be thought of as the variance in $\mu_a^+ - \mu_a$ across all arms. This is treated as a hyperparameter that is estimated from all arms that have large sample sizes (i.e., arms from which we can derived estimates of μ_a^+ with narrow confidence intervals). In our case, σ was estimated from the data shown in Figure 3 to be approximately 10^5.

From the above Bayesian prior, we can derive the posterior estimates for μ_a^+ and μ_a^- as follows:

$$\hat{\mu}_a^+ = \frac{\bar{\mu}_a^+ n_a^+ + \mu_a \sigma}{n_a^+ + \sigma} \qquad \hat{\mu}_a^- = \frac{\bar{\mu}_a^- n_a^- + \mu_a \sigma}{n_a^- + \sigma}$$

where n_a^+ and n_a^- are Kish's effective sample size [9] for the weighted samples from which $\bar{\mu}_a^+$ and $\bar{\mu}_a^-$ were computed, respectively, which are computed as:

$$n_a^+ = \frac{\left(\sum_{\{t \in H_a | a_t = a\}} w_t^+\right)^2}{\sum_{\{t \in H_a | a_t = a\}} w_t^{+2}} \qquad n_a^- = \frac{\left(\sum_{\{t \in H_a | a_t \neq a\}} w_t^-\right)^2}{\sum_{\{t \in H_a | a_t \neq a\}} w_t^{-2}}$$

We then define a new arm score such that:

$$\hat{s}_a = \frac{\hat{\mu}_a^+ - \hat{\mu}_a^-}{\hat{\mu}_a^-} \tag{2}$$

2.3 Recovering Arms

Prior A/B tests[5] at Duolingo that introduced new templates tended to boost notification conversion rates, even when the those new templates did not outperform existing templates in the long run. We surmised that this was due to a kind of "novelty effect": templates that users had not previously seen generally had positive impact. Under this hypothesis, the more often and recently a template is used for a given user, the less impact it will have.

We address this problem via a parametric approach using a cognitively-motivated formula. When an arm has previously been selected for a given user, we apply a recency penalty to its score to model the lack of a novelty effect. However, we hypothesize that this penalty should subside as the user's memory of the prior notification fades. Exponential decay functions have been shown to model human memory well [7, 14]. Hence, we define a modified score of arm a for a given round as follows:

$$s_{a,t}^* = \hat{s}_a - \gamma 0.5^{d_{a,t}/h} \tag{3}$$

where $d_{a,t}$ is the number of days since arm a was last selected[6] for the user corresponding to round t, and γ and h are both hyperparameters representing the base recency penalty and decay half-life, respectively. The latter two are estimated in Section 3.1.2.

2.4 Arm Selection

We use `softmax` [17] to define a policy to choose arms each round based upon the previously computed arm scores. Specifically, we select arm a with probability:

$$\pi(a \mid t) = \frac{\exp(s_{a,t}^*/\tau)}{\sum_{a' \in \mathcal{A}_t} \exp(s_{a',t}^*/\tau)} \tag{4}$$

where τ is a hyperparameter that controls the amount of exploration behavior (higher values lead to more exploration).

2.5 Arm Histories

Here we detail how the histories, H_a, are compiled for each arm. Rather than including all previous rounds where arm a is eligible, we filter this set to control for variance and to favor more recent data. We discuss each of these in the following sub-sections.

2.5.1 Controlling Variance. The variance for the estimates of $\bar{\mu}_a^+$ and $\bar{\mu}_a^-$ increase as n_a^+ and n_a^- approach zero, respectively. This can cause very large errors in these estimates when $b_t(a \mid t)$ is very large or very small, which can happen for arms that are performing very well or very poorly. To prevent this, we exclude from H_a any

[5] An A/B test is a form of randomized controlled trial experiment, used to test two versions of an app. Users are split randomly into buckets, and different business logic is used depending on the user's bucket (in this case, different copytext is included in the notification rotation) and the performance of each bucket is evaluated.

[6] In the case that arm a has never been selected for the applicable user before, $d_{a,t}$ is treated as infinity, in which case the second term goes to 0 and modified relative expected reward is simply $\hat{s}_a$.

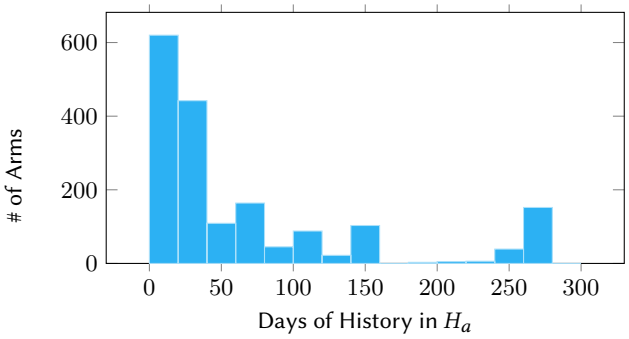

Figure 4: Histogram of # of days of data needed for an arm to reach a credible interval of ±0.15.

rounds where $b_t(a \mid t) < \theta$ or $1 - b_t(a \mid t) < \theta$, where θ is a hyperparameter set to a very small value. In our case, we chose to set the threshold to 0.5 %[7], which only excludes an average of 3 % of historical rounds.

2.5.2 Favoring More Recent Rounds. The typical reward of an arm may change gradually over time for a variety of reasons (e.g., seasonal effects, changes in the demographic distribution of users, etc.). In the bandit literature, this is referred to as *non-stationary* rewards [4]. If we include all of the historical rounds for each arm in H_a, then the scores for arms with long histories will change very slowly, as the new round data becomes diluted in the large history consisting mainly of very old rounds.

Thus, it makes sense to truncate older rounds in each arm's history in order to make the scores more responsive to non-stationary changes in arm rewards. However, there is a trade-off because truncating too much data will increase variance due to small sample sizes. To avoid this, we measure this variance using the arm score's 95 % credible interval and retain only enough data in each arm history to ensure that interval is smaller than a chosen threshold, ϕ, beyond which additional reductions in variance are of little value. This credible interval can be calculated by multiplying 1.96 by the arm score's standard error, which is:

$$SE(\hat{s}_a) = \frac{1}{\hat{\mu}_a^-}\sqrt{\frac{\hat{\mu}_a^+(1-\hat{\mu}_a^+)}{n_a^+ + \sigma} + \frac{\hat{\mu}_a^-(1-\hat{\mu}_a^-)}{n_a^- + \sigma}} \tag{5}$$

For our deployment, we chose ±0.15 percentage points for the hyperparameter ϕ. The number of days required to reach this threshold depends on how often the arm is used, which will vary based on its eligibility and performance. Figure 4 shows a histogram of the number of days of data required to reach this threshold for various template arms in our deployment.

2.6 Design Trade-offs

In the following sub-sections, we discuss some of the alternatives and trade-offs made in the design of this algorithm.

2.6.1 Relative vs Absolute Difference. In Equation 2 we use relative difference to compare the performance between arms (i.e., we normalize the differences by dividing by $\hat{\mu}_a^-$). Instead, we could have chosen to use the absolute difference (i.e., $\hat{\mu}_a^+ - \hat{\mu}_a^-$). It is not clear which is a more representative comparison, and we posit this is problem-dependent. However, our offline evaluations showed that both metrics performed similarly, and so the decision for us was a pragmatic one: since $\hat{\mu}_a^-$ was usually small (≈ 0.13 for most arms) and the differences between arms were very small (± 0.004), using the relative differences created more relatable scores ($\pm 3\,\%$) and also matched how business metrics are often evaluated at our company: percentage lift over baseline rather than absolute lift.

2.6.2 Distribution of arms under μ_a^-. The variable μ_a^- represents the expected reward when arm a is not chosen. However, this depends on the distribution with which alternative arms are chosen when arm a is not chosen. In our case, we use the distribution defined by the behavior policy in effect at the time the historical data was collected. The drawback of this is that μ_a^- tends to increase over time as the algorithm learns more optimal policies, which causes the template scores to drift downwards over time. This downward drift applies to all arms similarly, so it does not significantly affect the overall performance of the algorithm. However, this drift appears counter-intuitive to internal employees because it looks as if the arms are performing worse over time, when in fact it is simply an artifact of the baseline moving upwards! Ideally, we would instead use a fixed (e.g., uniform) distribution of alternative arms for μ_a^-, but this would increase the variance in the estimate.

2.6.3 Recovery Formulas. As mentioned previously, we take a parametric approach to modeling the recovery function for arms. Gaussian process kernels have been proposed for this purpose [12], and while they may work well for some problems, we believe the parametric approach is better suited to our case. Gaussian process kernels require many more than two parameters, and hence would need much more data to fit accurately. Since the effect size of reusing recent templates is so small in absolute terms (e.g., around -0.0006 if we reuse the last template, per Table 6), it would likely take an enormous amount of data to estimate a more complex recovery function with Gaussian process kernels, and there isn't much reason to suspect the function being estimated is complex, anyway.

Another option to consider is whether the number of times an arm has been chosen for a user should be factored in [6, 11, 15] in addition to the recency. We chose to focus on recency since it is simpler, and only requires us to know the last time each arm was used each round, rather than the user's full history. However, as future work we plan to experiment with alternative formulas that incorporate other variables about the user's prior interaction with template arms.

3 EVALUATION

In this section, we present two evaluations of the `Recovering Difference Softmax Algorithm`:

(1) Offline experiments using Duolingo's historical data to tune hyperparameters and estimate the algorithm's performance in terms of both the reward and the contribution of various components of the algorithm.
(2) An online experiment showing the impact of the bandit on Duolingo's business metrics after it was rolled out.

[7] In principle, it should be possible to find a θ that minimizes the variance for $\bar{\mu}_a^+$ and $\bar{\mu}_a^-$ for each arm, but we're not aware of a closed-form solution to this.

Table 3: Summary of the offline evaluation datasets

Dataset	Duration	Row Count
Train	15 days	88M
Test	19 days	114M

3.1 Offline Log Data Experiments

For offline evaluation, we first collected 34 days of data from the legacy system, where templates were selected from the eligible templates at random using the uniform distribution. The data included the notification's timestamp, user ID, list of eligible templates, selected template, history of templates sent to the same user over the prior 30 days, and reward (i.e., whether the user completed a lesson within 2 hours) for each captured round.

We used the first 15 days as training data to learn the arm scores from which the policy was derived, as described in the previous section. We reserved the last 19 days as test data. These are summarized in Table 3.

We used off-policy evaluation via weighted importance sampling [8, 17] to estimate the average reward of the policy being evaluated. Specifically, the average reward for a policy, π, was estimated as:

$$\bar{r}_\pi = \frac{1}{|\mathcal{T}|} \sum_{t \in \mathcal{T}} \frac{\pi(a \mid t)}{b_t(a \mid t)} r_t$$

where $\mathcal{T}$ is the set of rounds in the test set, and the behavior policy b_t equals $|\mathcal{A}_t|^{-1}$ (since the arm each round was selected via the uniform random distribution in this historical data).

Since exploration will reduce the expected reward in the short term, we used `argmax` instead of `softmax` in the arm selection policy for our initial experiments. In other words, $\pi(a \mid t)$ equals 1 if arm a has the highest modified score $s^*_{a,t}$ among eligible templates for that round, and 0 otherwise. Additionally, in the offline experiments, we did not use the empirical Bayes estimation to compensate for the small sample sizes (in effect, $\sigma = 0$). This was not a significant issue, as all arms started with 2 weeks of data and a followup experiment showed that adding the empirical Bayes estimation after the fact did not significantly change the final policy's total reward in the offline experiments.

3.1.1 Offline Experiment 1: Incorporating UI Language. Each of Duolingo's push templates are translated into 25 different user interface (UI) languages. The translation that is used for each notification is based upon what language the user has set as their UI language to display in the app. We expected that each template may perform differently depending upon the user's UI language, due to nuanced variations in meaning among translations, cultural differences, and other demographic factors.

When we compute the arm scores for each template on a per-UI language basis, we do find substantial differences between UI languages for some templates. For example, Table 4 shows the scores for three of Duolingo's templates across three of its most common UI languages (English, Spanish, and Portuguese). In some cases, the scores are different by 1 percentage point or more, which is substantial since the range of most of the scores is only $\pm 3.4\,\%$.

Table 4: Differences in arm scores by UI language.

Template	en	es	pt
A	$0.13\,\%$	$-0.74\,\%$	$-1.77\,\%$
F	$-1.60\,\%$	$0.62\,\%$	$0.19\,\%$
L	$1.13\,\%$	$2.98\,\%$	$2.17\,\%$

Table 5: Results from offline experiment 1 (UI language).

Bandit Algorithm	**Avg. Reward** ($\bar{r}$) ± 0.00015	**Rel. Diff.**
Baseline (random)	0.1295	-
Template	0.1311	$+1.2\,\%$
Template+UI Language	0.1318	$+1.8\,\%$

As such, our first experiment evaluated two variations of the bandit algorithm: one where each template corresponded to a single arm, and another where each template+UI-language pair was treated as a distinct arm. For this experiment, we ignored the recency penalty, effectively making it 0. Both were compared to a baseline where the template was chosen at random using the uniform distribution, as it was in the legacy system.

The results are presented in Table 5. As expected, both variations of the bandit algorithm beat the random baseline, gaining more than 1 % lift in reward. However, the variation that treats each template+UI-language pair as a distinct arm performs substantially better than the one-template-per-arm variation (1.8 % vs 1.2 %), confirming that it is worthwhile to take the different translations of each template into account when scoring arms.

In principle, it's possible to incorporate other user properties (such as region, device type, age, etc.) in a similar manner to exploit other differences in template performance between users. However, this would greatly multiply the number of arms, and thus the amount of data needed to estimate arm scores. Hierarchical modeling or contextual bandit approaches [1, 5] could address these concerns, but we leave them for a future work.

3.1.2 Offline Experiment 2: Estimating the Recency Penalty and Half-life. We also evaluated the effects of the recency penalty described in Section 2.3. Based upon previous A/B tests at Duolingo, we guessed that the recency penalty should mostly decay within the span of a few weeks to a month, and so we chose a half-life of 15 days. We then performed a grid search to estimate an optimal value for γ. As shown in Figure 5, the maximum is around 0.017.

We then compared the performance of the policy with the recency penalty to three baselines: a policy where the arms are selected with the uniform random distribution, a policy that always chooses the same arm as was last chosen for that user, and the template+UI language policy from the previous section.

The results are presented in Table 6. We see that the policy with the recency penalty beat the one from the last section by 0.2 percentage points ($p < 0.1$). However, this is likely an underestimate of the gain in reward for applying the recency penalty in production. This is because of a feedback effect that is not corrected for

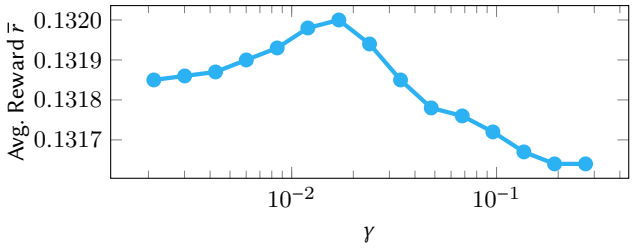

Figure 5: Average reward for values of γ (given $h = 15$).

Table 6: Results from offline experiment 2 (recency).

Bandit Algorithm	Avg. Reward $\bar{r}$ ±0.00015	Rel. Diff.
Baseline (random)	0.1295	-
Reuse Last Template	0.1289	-0.5 %
Template+UI Language	0.1318	+1.8 %
Template+UI Language w/ Recency Penalty	0.1320	+1.9 %

in our offline evaluation setup: under the optimized bandit policy, high-peforming arms are likely to have small $d_{a,t}$ since they are more likely to have been selected in previous rounds, whereas these are uncorrelated in the test dataset since arms were selected using the uniform distribution.[8] However, the baseline with a policy that always reuses the last template gives us an idea of how important the recency penalty is in production: reusing the same template (as the bandit algorithm would often do if the recency penalty were not applied) hurts reward by 0.5 %.

3.2 Online User Experiment

Following the offline evaluation, we evaluated the algorithm in production using an online controlled experiment (i.e., A/B test) for Duolingo's millions of daily practice reminders. Based on the results from the offline experiments, we scored one arm for each Template+UI Language pair, using $\gamma = 0.017$ and $h = 15$. See Section A for more details on the production system.

Unlike with the offline experiments, we wanted the bandit to exhibit some exploration behavior in production so that it could adapt to non-stationary changes in arm performance and to allow new templates to be introduced. Instead of choosing arms via `argmax`, we used `softmax`, which includes a hyperparameter τ to control the amount of exploration. Table 7 summarizes our offline evaluation to estimate the short-term impact of different values of τ.[9] Based on this, we selected a value of 0.0025, which we estimated would allow a reasonable amount of exploration while retaining more than 90 % of the potential gains.

[8]For this same reason, the γ estimated via grid search in the previous paragraph may be underestimated. But, subsequent production A/B tests showed that increasing γ even to as much as 3.4 did not significantly impact business metrics.

[9]We should emphasize that, since the training dataset is fixed in this offline setup, this experiment only estimates the short-term cost of exploration. Hence, these results do not imply that `argmax` would yield the best the reward in production in the long-run.

Table 7: Estimated impact of τ values on average reward.

Algorithm	Avg. Reward ($\bar{r}$)	Rel. Diff.	Explore %[†]
Baseline (random)	0.1295	-	89 %
softmax $\tau = 0.0200$	0.1306	+0.8 %	73 %
softmax $\tau = 0.0100$	0.1311	+1.2 %	57 %
softmax $\tau = 0.0050$	0.1316	+1.6 %	35 %
softmax $\tau = 0.0025$	0.1319	+1.9 %	17 %
argmax	0.1320	+1.9 %	0 %

[†] percent of the time the top-scoring arm is not chosen

During the test, a subset of Duolingo's users were split randomly between two experimental conditions: control and experimental. The control group's templates would be selected via the legacy algorithm (i.e., using a uniform random distribution over the eligible templates each round). Since the pools of templates and other aspects of the notification system had already been optimized through years of A/B tests, this provided a strong baseline to compare against. The experimental group's templates would be selected using the bandit algorithm as described in this paper. We tracked a variety of business metrics, but the main metrics we were interested in were:

Daily Active Users (DAUs): The number of distinct users in the bucket that opened and interacted with the Duolingo application on their device each day.

Total Lessons Completed: The total number of language lessons that users in the bucket completed.

D*X* Recurring Retention: The percentage of users who opened the Duolingo app on a given day who also did so X days later. This is further divided into *new* users (those whose accounts were created less than 48 hours before their first notification in the experiment) and *existing* users.

We expected the short-term impact on DAUs to be less than the 1.9 % boost in the 2-hour conversion rate predicted by the offline evaluation. This is because there is a significant amount of organic activity not originating from push notifications that would dilute the gains. On that basis, we hypothesized an increase in DAUs of 0.5–1.5 %. We expected similar gains in lessons completed. We also expected gains in recurring retention, but the magnitude of the impact was more difficult to predict.

Table 8 summarizes the results of our online user experiment after two weeks. We do in fact see a gain in daily active users and lessons completed of 0.5 % and 0.4 %, respectively. More striking, however, is that new user recurring retention increased by 2 %, showing that the optimized reminders significantly help in retaining new users (a critical component to an app's growth). This may be because the behavior of new users may be more sensitive to these kinds of optimizations, while their daily interaction with the app is still establishing habitual routines.

After the experiment completed, the bandit was launched to all users. Five months later, the bandit's performance in production was re-evaluated against a holdout set created by randomly using the uniform arm selection policy for 5 % of rounds (see Section A.4

Table 8: Operational A/B test results.

Metric	Gain
Daily Active Users (DAUs)	+0.5 % *
Total Lessons Completed	+0.4 % *
Existing User D1 Retention	+0.5 % *
Existing User D7 Retention	+0.2 %
New User D1 Retention	+2.2 % *
New User D7 Retention	+2.0 % *

* statistically significant ($p < 0.05$)

for details). The 95 % of rounds that used the bandit's optimized policy had 2.5 % higher reward than the holdout set, further demonstrating that the bandit continues to maintain business gains on par with what was predicted in our offline experiments.

4 CONCLUSION

In this paper, we have introduced the `Recovering Difference Softmax Algorithm`, a novel and practical multi-armed bandit variant for optimizing the content of recurring notifications. This algorithm solves a pair of key problems that are common in this setting that were not adequately addressed in the literature previously. Via offline evaluations, we demonstrated how the relative difference scoring and recency penalties — the latter of which was inspired by cognitively-motivated theories about novelty effects — each contribute to maximizing the reward. We furthermore showed via our online evaluation that the algorithm can be practically scaled to millions of users to boost user engagement.

Going forward, we plan to apply this algorithm to other types of messages, media, and calls to action; to experiment with recovery formulas that model quantity in addition to recency; and to incorporate additional features that will yield more personalized reward predictions for each user. To facilitate further research in this area, we have made a version of the data for our experiments in Section 3.1 available at: https://doi.org/10.7910/DVN/23ZWVI.

ACKNOWLEDGMENTS

We would like to offer our special thanks to our colleagues, Klinton Bicknell, Andrew Runge, Chris Brust, and Will Monroe, who provided invaluable feedback in the writing of this paper.

REFERENCES

[1] Peter Auer. 2002. Using confidence bounds for exploitation-exploration tradeoffs. *Journal of Machine Learning Research* 3, Nov (2002), 397–422.
[2] Peter Auer, Nicolo Cesa-Bianchi, and Paul Fischer. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning* 47, 2-3 (2002), 235–256.
[3] Peter Auer, Nicolo Cesa-Bianchi, Yoav Freund, and Robert E Schapire. 2002. The nonstochastic multiarmed bandit problem. *SIAM journal on computing* 32, 1 (2002), 48–77.
[4] Omar Besbes, Yonatan Gur, and Assaf Zeevi. 2014. Stochastic multi-armed-bandit problem with non-stationary rewards. In *Advances in neural information processing systems*. 199–207.
[5] Wei Chu, Lihong Li, Lev Reyzin, and Robert Schapire. 2011. Contextual bandits with linear payoff functions. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 208–214.
[6] Corinna Cortes, Giulia DeSalvo, Vitaly Kuznetsov, Mehryar Mohri, and Scott Yang. 2017. Discrepancy-based algorithms for non-stationary rested bandits. *arXiv preprint arXiv:1710.10657* (2017).
[7] Hermann Ebbinghaus. 1885. Memory: a contribution to experimental psychology. 1885. *New York: Teachers College, Columbia University* (1885).
[8] Daniel G Horvitz and Donovan J Thompson. 1952. A generalization of sampling without replacement from a finite universe. *Journal of the American statistical Association* 47, 260 (1952), 663–685.
[9] Leslie Kish. 1965. Survey Sampling.] ohn Wiley. *New York* (1965).
[10] Robert Kleinberg, Alexandru Niculescu-Mizil, and Yogeshwer Sharma. 2010. Regret bounds for sleeping experts and bandits. *Machine learning* 80, 2-3 (2010), 245–272.
[11] Nir Levine, Koby Crammer, and Shie Mannor. 2017. Rotting bandits. In *Advances in neural information processing systems*. 3074–3083.
[12] Ciara Pike-Burke and Steffen Grunewalder. 2019. Recovering Bandits. In *Advances in Neural Information Processing Systems*. 14122–14131.
[13] Herbert Robbins. 1952. Some aspects of the sequential design of experiments. *Bull. Amer. Math. Soc.* 58, 5 (1952), 527–535.
[14] Burr Settles and Brendan Meeder. 2016. A trainable spaced repetition model for language learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics*. ACL, 1848–1858.
[15] Julien Seznec, Andrea Locatelli, Alexandra Carpentier, Alessandro Lazaric, and Michal Valko. 2018. Rotting bandits are no harder than stochastic ones. *arXiv preprint arXiv:1811.11043* (2018).
[16] Aleksandrs Slivkins et al. 2019. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning* 12, 1-2 (2019), 1–286.
[17] Richard S Sutton and Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

A DEPLOYMENT DESIGN

In this appendix, we discuss how the `Recovering Difference Softmax Algorithm` was productionized at scale at Duolingo to optimize its push notifications.

A.1 Architecture

Figure 6 diagrams how the bandit algorithm was integrated with Duolingo's notification service. The main components of the bandit are identified by the dashed box.

Each time a notification is sent, the selected arm and other information needs to be captured so that the reward can be calculated and the arms can be scored. We refer to this as the "Decision Log". The minimum information needed to support the algorithm is described in Table 9. The decision log must handle a large volume of writes, rows are never updated once inserted, and data is retained indefinitely, so we chose to implement the decision log using a data streaming service, writing to flat files that can later be processed via `Spark` or another big data framework.

In addition, an auxiliary database, referred to in the diagram as the `User Arm History`, is used to track the last time each arm was selected for each user. This is needed to calculate the recency penalties each round. Since, unlike the decision log, this data is frequently overwritten, a relational database or key-value store is well-suited for this purpose.

The `Scorer` runs as a batch job to compute and update each arm's score. It does this by joining the decision log data with event information stored in the events database to compute the reward for each decision. The arm scores are written to a data store, which is read and used by the `Arm Selector`. The `Arm Selector` integrates with the notification service and decides which arm to use for each send. It is provided a list of eligible arms, and it queries the `Arm Scores` and `User Arm History` data stores to compute the probability for each arm using Equation 4. It then uses a random number generator (RNG) function to select the arm using those

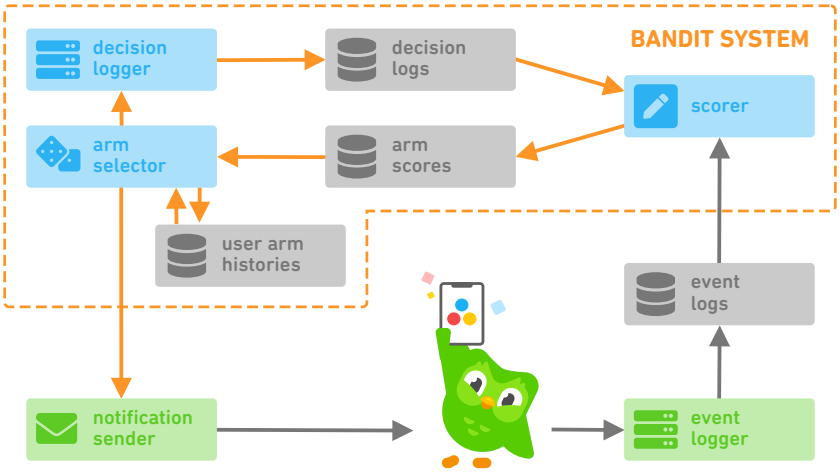

Figure 6: System diagram of our deployment architecture.

Table 9: Minimal list of decision log fields.

Name	Type	Description
`Timestamp`	`DateTime`	The date and time that the bandit decision was made.
`User ID`	`UserID`	Identifies the user that the decision applies to. This along with timestamp are used to compute reward when reconciling the data with the metrics database.
`Arm Prob.`	`Map[ArmID, float]`	A map of the pre-decision probability of each arm being selected. This is used to determine which arms were eligible and to compute the importance weights when estimating μ_a^+ and μ_a^-.
`Selected Arm`	`ArmID`	The arm that was selected.

probabilities and logs the decision via the `Decision Logger` before returning the selected arm to the notification sender, which then applies the corresponding template.

A.2 Introducing New Arms

The design of the bandit system allows new templates to be introduced without any special handling. Without historical data a new arm's score will be computed as 0. The recency penalty and exploration provided by `softmax` means that the bandit algorithm will occasionally try new templates so that data can be collected, and a more accurate score can be estimated.

A.3 Batch Processing Arm Scores

As mentioned previously, the arm scores are updated via a batch process daily, rather than immediately after the reward of each decision is known. So that millions of rows do not have to be reprocessed each time the batch job runs, values for $\bar{\mu}_a^+$, $\bar{\mu}_a^-$, n_a^+, and n_a^- can be computed from a single day's data and saved. Those daily values can then be aggregated to compute the complete arm scores.

The batch processing greatly simplifies the arm score update process and makes it more scalable. However, this delays the feedback loop between a decision and an arm's score being updated. Most bandit algorithms, in fact, assume instantaneous feedback. This is not a significant drawback for our application, however, because the arm scores do not change much in a single day. In fact, it takes roughly 1–5 weeks of data to narrow the 95 % confidence interval to ± 0.15 percentage points for most arms (see Figure 4). However, this does mean that the use of Bayesian priors to regularize relative reward estimates for small sample sizes, as described in Section 2.2, is very important. Otherwise, even a very poor template could initially get a very high arm score (because of the small sample size of its historical data) and so would get used almost exclusively once the arm scores were updated. In that case, without the Bayesian priors, it would take the bandit algorithm a full day to correct its mistake.

A.4 Holdout Set

5 months after launching the algorithm, we added a provision that, for a small percentage of rounds (5 %), the arm is selected using the random uniform distribution instead of the bandit's learned policy. This small holdout set has a very small impact on the gains of the algorithm, while providing two advantages:

(1) It provides us a way to monitor the bandit's ongoing performance and detect if the bandit ever fails to beat the random baseline, as might happen if a bug were introduced that would otherwise be too subtle to be noticed.
(2) It provides valuable training data, especially for arms that perform poorly and are thus rarely used by the bandit algorithm. The probabilities for these templates can get so low that the importance weighting creates high variance in the arm score estimates. The training data provided by this 5 % holdout set can offset that.