A Nonparametric Contextual Bandit with Arm-level Eligibility Control for Customer Service Routing

Ruofeng Wen[1,*], Wenjun Zeng[1] and Yi Liu[1]

[1]*Customer Engagement Technology, Amazon*

Abstract

Amazon Customer Service (CS) provides real-time support for millions of customer contacts every year. While bot-resolver helps automate some traffic, we still see high demand for human agents, also called subject matter experts (SMEs). Customers outreach with questions in different domains (return policy, device troubleshooting, etc.). Depending on their training, not all SMEs are eligible to handle all contacts. Routing contacts to eligible SMEs turns out to be a non-trivial problem because SMEs' domain eligibility is subject to training quality and can change over time. To optimally recommend SMEs while simultaneously learning the true eligibility status, we propose to formulate the routing problem with a nonparametric contextual bandit algorithm (K-Boot) plus an eligibility control (EC) algorithm. K-Boot models reward with a kernel smoother on similar past samples selected by k-NN, and Bootstrap Thompson Sampling for exploration. EC filters arms (SMEs) by the initially system-claimed eligibility and dynamically validates the reliability of this information. The proposed K-Boot is a general bandit algorithm, and EC is applicable to other bandits. Our simulation studies show that K-Boot performs on par with state-of-the-art Bandit models, and EC boosts K-Boot performance when stochastic eligibility signal exists.

Keywords

Bandit, Customer Service Routing, Arm Eligibility, Nonparametric

1. Introduction

In Amazon Customer Service (CS), we dispatch human agents, also called subject matter experts (SMEs) in real-time to handle millions of customer contacts. The SME routing automation process has two steps: first there is a natural language understanding (NLU) model to process customer's input and identify the relevant domain (return policy, device troubleshooting, etc.); then it dispatches an SME who is eligible. We define *eligibility* when the SME masters the required skill in the relevant domain through training. SME routing turns out to be a non-trivial problem for four reasons. First, the NLU model is unlikely to categorize the domain with perfect accuracy. Second, the reliability in SME eligibility as identified by operation team is subject to training program quality and readiness. Third, the domain taxonomy and SMEs' eligibility status can change in a decentralized way. In reality, it is difficult to keep track of all the eligibility updates, assuming correctness. Finally, eligible SMEs do not guarantee customer satisfaction, leading to noisy feedback signals. All these uncertainties make SME routing a challenging problem.

To tackle the complexity, we formulate CS routing as a recommendation problem and use contextual Bandit as a solution. Bandit, a framework for sequential decision making, has been used for online recommendation across companies such as Amazon, Google, Netflix and Yahoo [1]. In contextual bandit, the decision maker sequentially chooses an arm/action (SME in our case), based on available contextual information (contact transcript embedding, customer profile, etc.), and observes a reward signal (customer satisfaction, contact transfer, etc.) [2]. The objective is to recommend arms at each step to maximize the expected cumulative reward over

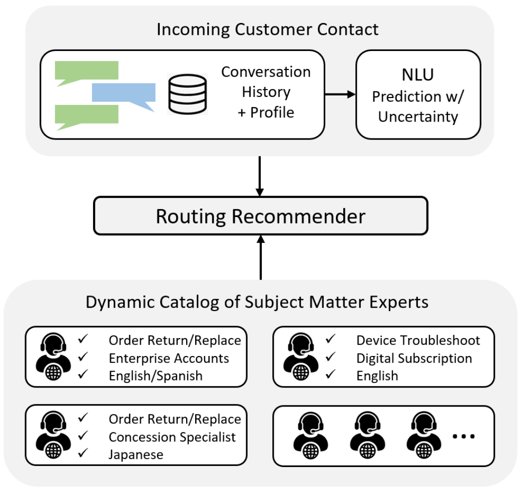

Figure 1: The CS Routing Problem. The goal of the recommender is to select an agent that will result in the best outcome of this customer contact.

Knowledge-aware and Conversational Recommender Systems (KaRS) Workshop, 2022, Seattle, WA

*Corresponding author.

ruofeng@amazon.com (R. Wen); zengwenj@amazon.com (W. Zeng); yiam@amazon.com (Y. Liu)

time. We apply bandit to the routing problem because it can optimally explore uncertainties and adapt to dynamic changes [3]. Particularly, the uncertainties in SME eligibility motivate us to formulate a new type of bandit to utilize arm-level eligibility signals.

Our contribution. We propose K-Boot, a nonparametric contextual bandit algorithm to model reward and explore, and an Eligibility Controller (EC) algorithm to model arm-level eligibility. K-Boot uses a k-nearest neighbors (k-NN) approach to find similar samples from the past, applies a Nadaraya–Watson kernel regression among the found samples to estimate the reward, and adopts Bootstrap Thompson Sampling as the exploration strategy. The EC component implements a dynamic top arm filter based on its estimated Spearman's Rank Correlation Coefficient between eligibility and reward. We are interested in this end-to-end nonparametric setup for practicality: robustness in performance, strong interpretability and simple non-technical maintenance which is friendly to business partners. For instance, the k-NN component makes it easy to investigate which historical contacts support the SME recommendation, and then deploy instant online fixes by trimming unwanted outliers. While K-Boot and EC are proposed here as a suite, each can be applied independently - K-Boot is a general Bandit algorithm and EC can control arm-eligibility for other bandits, such as LinUCB [4].

In the remainder of the paper, we review related literature in Section 2, set up the formal problem in Section 3 and detail the proposed algorithms in Section 4. We then present our model results in Section 5, and conclude the paper last.

2. Related Work

In most bandit applications, the items to recommend are products and webpage content (widget, news articles, etc.). Recently, we see research efforts in using bandit for routing recommendation in a *conversation* [5, 6]. In [6], they used contextual bandit to utilize query and user features along with the conversational context and build an orchestration to route the dialog. In [5], they built bandit models to assign solutions given customer's queries on Microsoft's product to a chatbot. This is the most relevant work to our problem. However, they assume an upstream model to provide a set of plausible actions to start with and the bandit itself does not deal with the uncertainty in arm eligibility. Overall, comparing to the extensive bandit literature existing for product and webpage content recommendation, few bandit works are there for routing recommendation.

Thompson Sampling (TS) is a heuristic for balancing exploration and exploitation in bandits [7]. In the parametric manner, TS is implemented by drawing the reward function parameter θ from a posterior distribution, calculating rewards with the sampled parameters, and selecting the arm that maximizes the estimated reward function. θ is often assumed to follow Conjugate-Exponential Family distributions resulting closed-form posteriors. There are also work where variational methods are used to provide an analytical approximation to the posterior and enforce trackability [8]. Bootstrap TS introduces volatility in a different way: it pulls the arm with the highest bootstrap mean, estimated from reward history in a nonparametric way [9]. One drawback of the approach is the computational cost to train one bandit model per Bootstrapped training set, which is difficult for large-scale real-time problems. [10] proposed a general nonparametric bandit algorithm but they fell back to the parametric reward function approach when there is context. [11] further improved the exploration efficiency with Bayesian Bootstrap, but also only for non-contextual bandit. We pair Bootstrap TS with a single k-NN for nonparametric contextual exploration.

Kernel-based Bandit uses a nonparametric reward estimator via kernel functions, combined with an exploration policy like TS or UCB. Gaussian Process (GP) has wide applications in bandit domain, e.g. GP-UCB [12] and GP-TS [13]. However, GP inference is expensive - the standard approach has $O(N^3)$ complexity, where N is the number of training samples. The most efficient sparse or low-rank GP approach still requires $O(ND^2)$, where D is a hyperparameter indicating the number of *inducing variables* - see [14] for a review. We use k-NN to enable $O(k \log N)$ inference time.

3. Formulation: Contextual Bandit with Arm Eligibility

We define two types of eligibility setup with different data interfaces. **(1) Eligibility Scores**: for a given context $\mathbf{x}$, a list of eligibility scores for each arm $\mathbf{e} = (e_1, e_2, \dots)$ is observed, where $e_j \in [0, 1]$ is for arm a_j. A larger score means higher confidence in the arm being eligible for the current sample/iteration. An intuitive interpretation is that the arms are bidding for themselves. This score setup assumes (the owner of) each arm is able to adaptively assess the dynamic environment in a distributed and independent manner that can be unknown to the bandit. Typical examples include ads bidding and voice assistant's skills detection - each ad/skill may be managed via the same API, but by different clients under dynamic competition. **(2) Eligibility States**: assume the system at any time can be classified into one of L possible states. Each arm *claims* whether it is eligible under a state, represented by a constant binary vector $\mathbf{c}_j$ of length L for arm a_j. If the l-th element of $\mathbf{c}_j$ is 1, arm a_j claims to be eligible under the l-th state, and 0 otherwise. Given a context $\mathbf{x}$, a stochastic distribution over the L states is observed, represented by a probability vector $\mathbf{p}$ of length L. For CS routing, the state is the type of customer issue, the claim is SME's corresponding skill-set, and $\mathbf{p}$ is the NLU predictive distribution. Note that a score can be computed as the inner product of the claimed eligibility binary vector and state probability vectors: $e_j := \mathbf{c}_j^{\intercal}\mathbf{p}$, so the state setup converges to the score setup. The major practical difference between the two interfaces is the amount of effort in the claiming side: the state interface simplifies the eligibility claims to static binary votes on a finite set of states, while the score interface mandates scoring with context awareness.

Consider a contextual bandit with a set of arms $\{a_1, a_2, \dots\}$. At the nth round, context $\mathbf{x}_n$ is observed and eligibility score $\mathbf{e}_n$ is either observed (score interface) or derived (state interface). After an arm is pulled, a reward $r_n \in [0, 1]$ is revealed. The goal is to minimize the expected regret over N rounds: $\sum_{n=1}^{N}(\mathbb{E}[r|a_*^n, \mathbf{x}_n, \mathbf{e}_n] - \mathbb{E}[r|a^n, \mathbf{x}_n, \mathbf{e}_n])$, where a_*^n is the optimal action for this round in hindsight.

4. Methodology

The proposed algorithms run in order: Eligibility Controller (EC) to filter out ineligible arms, and K-Boot to find the optimal one within the rest. Below, we first introduce K-Boot to set the base and then EC.

4.1. K-Boot

K-Boot is a nonparametric bandit model. For a given context $\mathbf{x}$, it estimates the reward for arm a as the Nadaraya-Watson kernel weighted average over the observed rewards of the k nearest neighbors of $\mathbf{x}$ from all historical samples where arm a was pulled. We use Bootstrap TS as the exploration strategy. k-NN regression was known to fit well with Bootstrap-Aggregation, with an analytical form of smoothed conditional mean estimate that does not require actual resampling [15, 16]. However, sampling from the confidence distribution of the mean estimate has little discussion. We devise a trick to shrink the resampling range from all historical samples to a local wrapper around the k-NN.

K-Boot is detailed in Algorithm 1. At iteration n with context $\mathbf{x}_n$, for the mth arm a_m, let $\mathcal{D}_m$ be the set of historical samples where a_m was pulled, and $N_m = |\mathcal{D}_m|$. If N_m is zero, we sample reward from a standard uniform distribution (Line 6). If N_m is greater than K, we first build an *influential* subset $\mathcal{D}'_m$ containing the K'-NN of $\mathbf{x}_n$ ($K < K'$). Intuitively, the K'-NN serves as a buffer to cover enough sample variance around the K-NN, so that a Bootstrap on the influential samples is a good approximate of that on all samples. Formally, a sample is considered influential to make inference about $\mathbf{x}_n$, if its probability of being in the K-NN of $\mathbf{x}_n$ *after* a Bootstrap on $\mathcal{D}_m$ is greater than ε. This probability is computed analytically on Line 9, based on the derived equation (6) in [15]. Here the tolerance hyperparameter ε controls the risk of missing influential samples and thus the fidelity of approximated Bootstrap TS. In our experiments with $N_m \leq 10^4$ and $\varepsilon = 0.01$, K' is empirically well bounded by $2K$. This process shrinks all later computation to K' samples, with the overhead neighbors search cost $O(\log N_m)$ per sample (we used Hnswlib [17] for approximated search). If N_m is less than K, $\mathcal{D}_m$ itself is the influential set. We then add two pseudo samples to $\mathcal{D}'_m$ on Line 14-15, in order to expand reward's empirical range for Bootstrap exploration [10]. To give pseudo-samples proper contexts thus kernel weights, we set their distance to $\mathbf{x}$ as that of a random observation in $\mathcal{D}'_m$, so its weight shrinks as more data seen. On Line 16-18, we then draw a Bootstrap resample and select K nearest samples from it to calculate the estimated reward for a_m as the kernel weighted average. we implement $K_h(\cdot, \cdot)$ as the simplest $O(K)$ Nadaraya-Watson kernel estimator with automatic bandwidth selection [20] - more advanced models can be plugged in.

As we model reward for each arm separately as a common practice, K-Boot is flexible to add and removal of arms. This benefits applications like ours with decentralized and independent arms management. In addition, K-Boot has simple maintenance: (1) the nonparametric model has minimal assumption in data and only a single important hyperparameter K to balance between accuracy and computation. $\varepsilon = 0.01$ works well across our experiments. (2) the algorithm is business friendly in terms of decision interpretability by neighboring examples and allowing intuitive online instant modification of model behaviors. For example, CS business owners may hide contacts with undesired trending patterns during a specific period (e.g. under a legacy policy), or let newly available SMEs "inherit" past example contacts or eligibility claims (next section) via domain knowledge (e.g. a rehired agent or additional training programs). The changes on data visibility will instantly and predictably affect the bandit model in production, with clear attribution. This data-driven while human-in-the-loop fungibility is essential to fast-paced operational business like customer service.

4.2. Eligibility Control

EC is used to recognize arm-eligibility and the associated uncertainties, for optimized bandit exploration. In an ordinary contextual bandit model, the eligibility information can be trivially considered as part of the context $\mathbf{x}$ - it is assumed to be positively correlated with r, thus predictive as a plain input to a reward estimator. This

Algorithm 1: K-Boot: a fully nonparametric contextual bandit

```
Input: Number of iterations $N$, number of arms $M$, number of nearest neighbors $K$, kernel function with
       bandwidth $K_h(\cdot,\cdot)$, regularized incomplete beta function $F_{\alpha,\beta}(\cdot)$, approximation tolerance $\varepsilon$.
Initialize sample pool $\mathcal{D}_m := \varnothing$ and its size $N_m := 0$, for each arm $m = 1, \ldots, M$
for $n = 1, \cdots, N$ do
    Observe context $\mathbf{x}_n$
    for $m = 1, \cdots, M$ do
        if $N_m = 0$ then
            Estimate reward: $\hat{r}_{m,n} \sim \mathcal{U}(0,1)$
        else
            if $N_m > K$ then
                $K' := \min(k')$, s.t. $\sum_{i=1}^{K} [F_{i,N_m-i+1}(\frac{k'}{N_m}) - F_{i,N_m-i+1}(\frac{k'-1}{N_m})] > 1 - \varepsilon$
                Find influential neighbors $\mathcal{D}'_m :=$ the top $K'$ samples in $\mathcal{D}_m$, with the largest $K_h(\mathbf{x}_n, \cdot)$
            else
                Set $\mathcal{D}'_m := \mathcal{D}_m$
            end
            Draw a random sample $(\mathbf{x}_\star, r_\star)$ from $\mathcal{D}'_m$
            Add pseudo-samples: $\mathcal{D}'_m := \mathcal{D}'_m \cup \{(\mathbf{x}_\star, 0), (\mathbf{x}_\star, 1)\}$
            Draw a Bootstrap sample $\mathcal{D}^\star_m$ from $\mathcal{D}'_m$
            Find neighbors $\mathcal{D}^\star_{m,K} :=$ the top $\min(K, |\mathcal{D}^\star_m|)$ samples in $\mathcal{D}^\star_m$, with the largest $K_h(\mathbf{x}_n, \cdot)$
            Estimate reward: $\hat{r}_{m,n} := \sum_i r_i K_h(\mathbf{x}_n, \mathbf{x}_i) / \sum_i K_h(\mathbf{x}_n, \mathbf{x}_i)$, summing over $\mathcal{D}^\star_{m,K}$
        end
    end
    Pull arm $m^\star := \arg\max_m \hat{r}_{m,n}$, and observe true reward $r_n \in [0,1]$
    Update $\mathcal{D}_{m^\star} := \mathcal{D}_{m^\star} \cup \{(\mathbf{x}_n, r_n)\}$ and $N_{m^\star} := N_{m^\star} + 1$
end
```

assumes the reward model is able to eventually learn the relation between r and e after enough rounds. This trivial approach does not directly utilize eligibility to limit the range of arm exploration, and may unnecessarily explore inappropriate actions with catastrophic regret during cold-starts. The other extreme is to strictly follow the eligibility information and ignore the reward feedback: for the eligibility score interface, this could be simply pulling the arm with the highest score as if it were an oracle; for the eligibility state interface, this could be finding the state with the highest probability then remove arms that did not claim that state. Therefore, the empirical data would never be used to validate and correct the potentially biased business logic - a common pitfall in practice.

The main idea of EC is to leave only the top-k arms with the highest eligibility scores before applying a normal bandit, and adjust k periodically based on the empirical correlation between eligibility scores and rewards. We use Spearman's Rank Correlation Coefficient, denoted as ρ. Using top-k arms is a heuristic trade-off between $k = M$ (the trivial case; M is the total number of arms), and $k = 1$ (the strict rule case). Intuitively, for the case of perfect correlation between score and reward ($\rho = 1$), $k = 1$ is the oracle solution. If eligibility score has zero or even negative correlation ($\rho \leq 0$) with reward, $k = M$ is the optimal solution - otherwise the arm exploration is restricted adverserially or randomly for no benefits. In the no-correlation case, the probability of "the best arm by reward is not in the top-k by score", defined as a *leak*, is linear in k: $P(\text{leak}|k, \rho = 0) = 1 - k/M$. It is obvious $P(\text{leak}|k, 0 < \rho < 1) < P(\text{leak}|k, \rho = 0)$. This observation reveals two insights: (1) $\alpha := P(\text{leak}|k, \rho)$ characterizes a type of risk in applying a top-k score filter to arms; (2) when controlling the risk of a leak at certain level, e.g. $\alpha = 0.01$, k is a function of ρ. The true correlation ρ between reward r and eligibility score e is unknown, but can be estimated from historical observations: $\{(r_i, e_i)\}_i$. Therefore, EC essentially calculates k such that the risk $P(\text{leak}|k, \hat{\rho})$ is controlled at a given level.

To pair with K-Boot, EC also models $P(\text{leak}|k, \rho)$ in a nonparametric fashion to avoid extra assumptions. Per ρ, although Spearman's rank correlation coefficient is used for estimation, Kendall's τ or any other rank-based correlation measure applies similarly. Across the arms and rounds, we assume the rank of e is a noisy perturbation of the rank of r, and parameterize this perturbation as "performing p random neighbor inversions". An inversion is simply switching two elements in a sequence, and

Algorithm 2: Eligibility Control

```
Input: Number of iterations $N$, number of arms $M$, risk level $\alpha$, score initial threshold $k_0$, a pre-computed
       empirical dictionary $G(n, \alpha, \rho) \rightarrow k$
1  Initialize threshold $k := k_0$ and observation pool $\mathcal{D}_e := \varnothing$
2  for $n = 1, \ldots, N$ do
3      Observe context $\mathbf{x}_n$, and eligibility scores $\{e_1, \cdots, e_M\}$, for each arm
4      Find the $k$th largest element $e_{[k]}$ among the scores
5      Sample reward for each arm if $e_m \geq e_{[k]}$ (Algorithm 1, Line 5-18)
6      Pull the $m^\star$th arm with highest reward estimate (Algorithm 1, Line 21-22)
7      Observe true reward $r_n$, and update $\mathcal{D}_e := \mathcal{D}_e \cup \{(r_n, e_{m^\star})\}$
8      (Periodically) compute $\hat{\rho}$ from $\mathcal{D}_e$ and set $k := G(n, \alpha, \hat{\rho})$
9      Set $k := M$ if $k \geq (1 - \alpha)M$
10 end
```

a neighbor inversion switches two neighboring elements. Note $p = 0$ indicates $\rho = 1$ because the perturbed rank sequence is identical to the original one; $p \rightarrow \infty$ is effectively a random permutation thus $\rho = 0$. This setup provides a generative process to simulate the joint distribution of all parameters for a rank sequence of length n: (1) do p random neighbor inversions on the sorted sequence of ranks $(1, \cdots, n)$; (2) see if there is a *leak*, namely if the element 1 is now at or beyond the kth position; (3) compute $\hat{\rho}$ between the original and current sequences. The three steps can be replicated for a sufficient number of times, to obtain the final correlation coefficient $\hat{\rho}$ and risk level $\hat{\alpha}$ by averaging individual $\hat{\rho}$ and counting the frequency of leaks. For different combinations of (k, p, n), as the above process does not depend on any actual data, it can be performed offline in batch to get the corresponding $(\hat{\rho}, \hat{\alpha})$. The resulting empirical dictionary $G : (n, \alpha, \rho) \rightarrow k$ is stored, and later queried during online inference. To avoid unnecessary control in the edge case where eligibility score turns out to be pure noise, k is reset to M (no EC) if G outputs a k that is greater than the trivial value $(1 - \alpha)M$. Finally note the generative distribution of random neighbor inversions is assumed to be uniform along the whole sequence, yielding the unbiasedness of the estimator $\hat{\rho}$ even if the observed data points $\{(r_i, e_i)\}_i$ are censored by the top-k filter policy.

The full online EC process is described in Algorithm 2 (excluding the offline dictionary generation). The eligibility state interface is converted into the score interface before EC, so the scores are the only required inputs. The algorithm takes K-Boot (Algorithm 1) as the bandit counterpart just for demonstration, and applies to any bandit that has arm-independent reward models. EC has a single hyperparameter: the controlled risk of a leak α. Note α is the risk of missing *the best* arm, so its influence on cumulative rewards depends on the arm reward distribution, specifically the gap between the first and second best arm. We leave other types of risk control mechanism with distributional assumptions to future work.

5. Experiments

In this section, we evaluate the performance of K-Boot with two benchmarks: LinUCB [4] and NeuralUCB [21]. The experiment setup mostly follows the methodology reported in [21] with 4 synthetic simulations and 4 realworld datasets. EC is tested on synthetic data with an eligibility score setup, then on an Amazon Customer Service routing dataset. Datasets are introduced below.

Synthetic Datasets. The bandit has $M = 10$ arms, and runs on $N = 5000$ samples/rounds. It observes context with $d = 20$ dimensions, each following independent $\mathcal{N}(0, 1)$. Four types of true reward functions are tested - (1) *linear*: $h_0(\mathbf{x}) = 0.1\mathbf{x}^T\mathbf{a}$; (2) *quadratic*: $h_1(\mathbf{x}) = 0.05(\mathbf{x}^T\mathbf{a})^2$; (3) *inner-product*: $h_2(\mathbf{x}) = 0.002\mathbf{x}^T\mathbf{A}^T\mathbf{A}\mathbf{x}$; (4) *cosine*: $h_3(\mathbf{x}) = \cos(3\mathbf{x}^T\mathbf{a})$; where each entry of the parameters $\mathbf{A} \in \mathbb{R}^{d\times d}$ and $\mathbf{a} \in \mathbb{R}^d$ is drawn from an independent $\mathcal{N}(0, 1)$, different for each arm. The observed reward has noise: $r := h(\mathbf{x}) + \varepsilon$, where $\varepsilon \sim \mathcal{N}(0, \sigma_r^2)$ and σ_r is drawn from $\mathcal{U}(0.01, 0.5)$ independently for each arm. 20 runs are replicated by random seeds.

Synthetic Datasets with Eligibility. Given above, the eligibility scores are further simulated by perturbing the counter-factual reward of each arm: $e := wr + \varepsilon'$, to induce the correlation. Here w is an arm-specific scaling coefficient draw from $\mathcal{U}(-0.1, 1)$. It has a small chance to leave e and r negatively correlated, as tolerating eligibility claim error in systems under the overall positive correlation - the only assumption of EC. The noise term $\varepsilon' \sim \mathcal{N}(0, \sigma_e^2)$ controls the correlation effect size. By setting a proper σ_e, the Spearman's rank correlation coefficient ρ between reward and eligibility score can be fixed at an arbitrary positive value.

UCI Classification. Four real-world datasets from

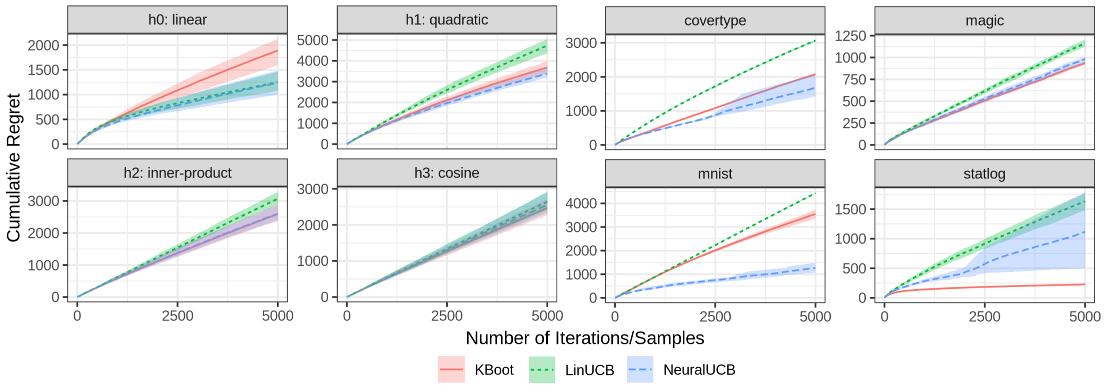

Figure 2: Benchmarking K-Boot, LinUCB and NeuralUCB. Left half: synthetic datasets; right half: UCI datasets. Model hyperparameters are set as the best on synthetic datasets. Metric is averaged across 10 runs with an 80% confidence band.

UCI Machine Learning Repository [22] are gathered: *covertype*, *magic*, *statlog*, and *mnist*. These multi-class classification datasets are converted into multi-arm contextual bandits problems, following the method in [4]: each class is treated as an arm, and reward is 1 if bandit pulls the correct arm (identify the correct class) and 0 otherwise. Each dataset is shuffled and split into 10 mutually exclusive runs with 5000 samples each, and fed to bandit one in each round (*magic* has less than 50000 samples so resampling is performed).

CS Routing with Eligibility. Agent skill-level routing is considered: CS agents are assigned to different skill groups by operation teams, and the routing system determines which skill is required to resolve a given customer contact based on the customer profile, policy rules and natural language understanding. As a proof of concept before any online experiments, we formulate an offline bandit dataset with historical observations. About 2 million Amazon US customer service text chats with bot-to-agent routing are gathered. To introduce stochasticity, a historical routing policy emulator is trained on a 95% random subset of the chats, by taking the customer-bot transcripts before the routing action as input to predict which skill the system chose. Specifically, the emulator consists of a pretrained GPT-2 transformer encoder [23], with a multi-class sequence classifier head to predict the top 20 skill groups. The model is trained (fine-tuned) for one epoch with learning rate 10^{-4} and batch size 32. For the bandit simulation, the final actual resolving skill, after all the potential agent-to-agent transfers since the initial routing, serves as the ground truth - if the routed skill agrees with the final skill, the action is considered as accurate and having avoided a potential transfer. At each round, the bandit receives a 256-dimensional text embedding from the same GPT-2 encoder as observed context, and a 20-dimensional probabilistic prediction from the emulator as eligibility scores for each arm/skill. If the bandit chooses a skill matching the final one, reward is 1 and otherwise 0. In this case, the cumulative regret is an upper bound for the number of transferred contacts, because agents assigned to different skills may resolve the contact regardless (e.g. digital order agents are often equipped to resolve retail order issues, so they may not transfer an incorrectly routed contact). The rest 5% chats are used to generate 10 bandit runs with 8000 samples each[1]. We observe that the empirical ρ across the runs is 0.3492. We leave the below to future work: (1) online experiments to measure real transfers and positive customer reviews as the reward metric; (2) agent-level routing given agent profiles and finer grain eligibility definitions than skill groups.

5.1. K-Boot Benchmarks

We compare K-Boot with LinUCB and NeuralUCB on both synthetic and UCI classification datasets. With the 4 synthetic datasets, we first select the following hyperparameters for each Bandit algorithm of interest (1) number of nearest neighbors $k \in \{20, 50, 100\}$ for K-Boot; (2) weight parameter for exploration $\alpha \in \{0.1, 1.0, 10\}$ for LinUCB; (3) regularization parameter $\lambda \in \{0.1, 0.01, 0.001\}$ and exploration parameter $\nu \in \{0.2, 0.02, 0.002\}$ for NeuralUCB - other hyperparameters such as learning rate, batch settings and number of layers are set the same as in the original code base[2]. We settle on $k = 100, \alpha = 10, (\lambda, \nu) = (0.001, 0.002)$ per having the lowest cumulative regret averaged across $h_0 \sim h_3$ and 10 replicated runs, and carry these values over to compare the actual model performance on the 4

[1]From another perspective, this offline bandit-with-eligibility simulation setup is equivalent to "*using a bandit for online improvement of a black-box probabilistic classifier with unknown accuracy*".
[2]https://github.com/uclaml/NeuralUCB

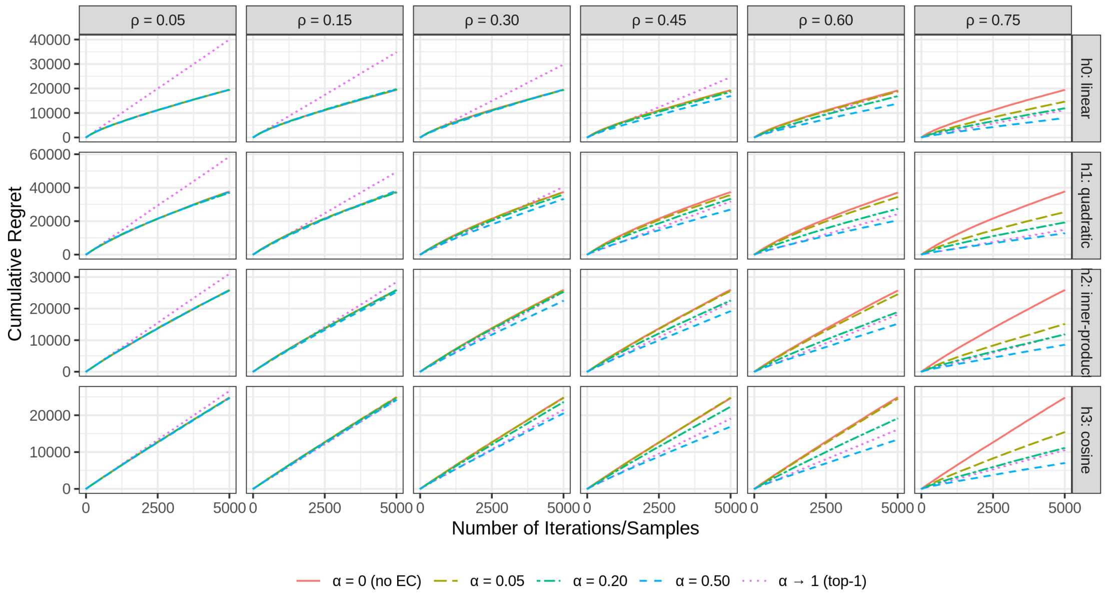

Figure 3: EC under different Spearman's Correlation ρ between eligibility score and reward, with different risk level α. The bandit part is K-Boot and eligibility scores are not in context.

UCI datasets. We believe this setup is more realistic because most bandit applications do not have the luxury for intensive hyperparameter tuning due to small sample size or non-stationary environment. Finally, all experiments in this paper implement online, single-sample model updates. Figure 2 left shows the performance of the models with final selected hyperparameters. Except for the linear reward scenario where LinUCB/NeuralUCB has advantage in correct/close model specification, there is no significant performance difference between K-Boot and NeuralUCB while LinUCB is inferior. Figure 2 right shows the performance comparison on the UCI real-world datasets. K-Boot and NeuralUCB have a tie of winning on two datasets each, and LinUCB is consistently the worst. By examing the 80% confidence band (P10-P90 across 10 runs), we find NeuralUCB has larger performance volatility (wider bands) on real data than the other two models, due to learning instability in certain runs.

5.2. Eligibility Control Testing

For the synthetic data, we control σ_e to simulate different levels of reward-reflecting reliability in eligibility scores, with resulting $\rho(r, e) \in \{0.05, 0.15, 0.3, 0.45, 0.6, 0.75\}$, for each true reward function h_0 - h_3. In the algorithm, we set different risk level α in EC. Note $\alpha = 0$ is effectively not using EC, and we abuse the notation $\alpha \to 1$ to denote the strict top-1 policy: pull the arm with the highest eligibility score.

Figure 3 shows the results of applying EC to K-Boot, with eligibility scores used by EC only (explained later). When the signal in e is weak (low ρ), EC adaptively raises the top-k threshold close to M, achieving the same performance as no EC. As ρ grows, the advantage reveals - using EC is consistently better than not. The dynamic thresholding may dominate the top-1 policy even under strong eligibility signals, with the right α. EC performs the best for $\alpha = 0.5$ across all synthetic datasets, and the same value is carried over to the next experiment on CS Routing data.

EC can be applied to other bandits. Figure 4 shows the same synthetic data results but with EC + LinUCB. Here eligibility scores are also added to bandit context. The reason why the previous experiment did not take scores as K-Boot input is because k-NN has no native feature selection mechanism. If eligibility score is close to white noise (low ρ), model struggles to fit the reward, leading to a distraction from other presented visual patterns. For LinUCB, such side-effect is minimal for the simulated noise is Gaussian, so it is a better condition to test the difference between the trivial way versus the EC way of utilizing eligibility. EC still dominates no-EC in most cases. There are a few exceptions for the linear true reward function h_0, where $\alpha = 0.5$ is worse because LinUCB learns so well that taking a high risk of missing the best arm is not cost-effective. An interesting observation is that, while the strict top-1 rule was dominated by K-Boot + EC (Figure 3), it actually beats LinUCB + EC

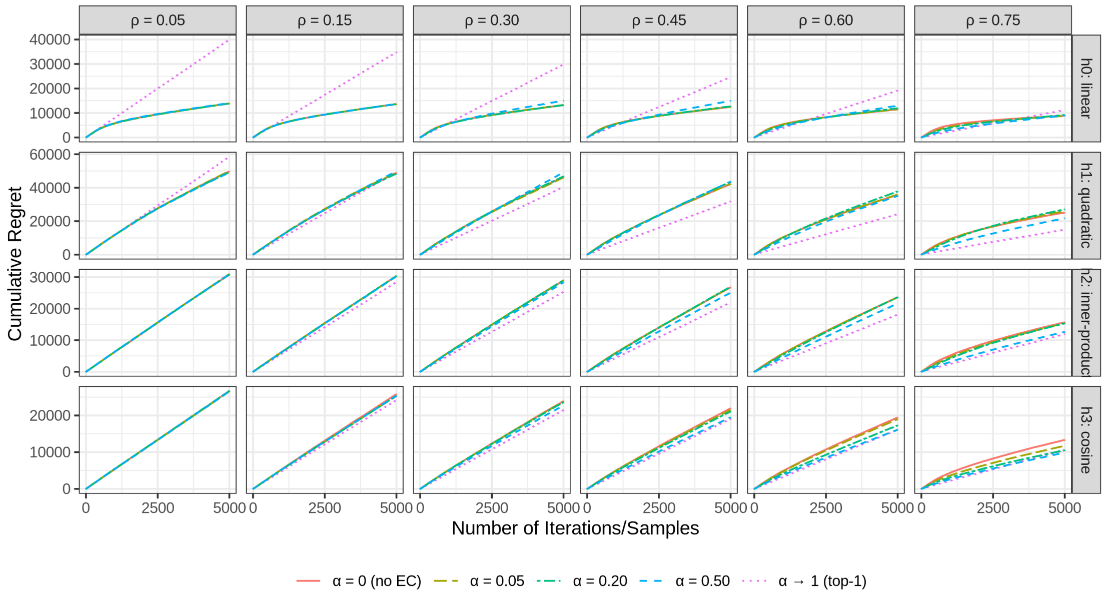

Figure 4: EC under different Spearman's Correlation ρ between eligibility score and reward, with different risk level α. The bandit part is LinUCB and eligibility scores are part of context.

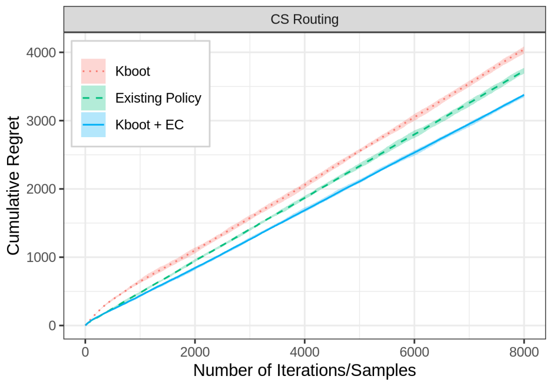

Figure 5: CS Routing Results. Metric is averaged across 10 runs with an 80% confidence band.

for nonlinear true reward functions even at $\rho = 0.15$. This indicates the power of the bandit model is still a key driver for ML to out-perform static rules.

Figure 5 shows the result on CS Routing data, where EC significantly improves routing accuracy. Routing to the skill with the highest probability from the emulator serves a stochastic surrogate of the existing policy. Pure K-Boot result without knowing the emulator outputs is set as a baseline. With only 8000 samples, the average accuracy of the emulator policy is improved by K-Boot + EC from 53.37% to 57.78%.

6. Conclusions

We proposed a nonparametric contextual Bandit algorithm K-Boot with arm-level eligibility control (EC) for routing customer contacts to eligible SMEs in real-time. While K-Boot and EC are proposed here as a suite, each can be applied independently - K-Boot is a general Bandit algorithm and EC can control arm-eligibility for other bandits. We compared K-Boot with LinUCB and NeuralUCB. When looking at average regret performance over simulation runs, K-Boot and NeuralUCB had a tie in winning scenarios and were comparable in terms of robustness to different data distributions. Both worked better than LinUCB. However, we observe larger performance variability for NeuralUCB as opposed to K-Boot. We further found the EC component improved K-Boot's performance on both synthetic datasets and a simulation scenario in CS routing when eligibility exists.

References

[1] G. Elena, K. Milos, I. Eugene, Survey of multiarmed bandit algorithms applied to recommendation systems, International Journal of Open Information Technologies 9 (2021) 12–27.

[2] Y. Liu, L. Li, A map of bandits for e-commerce, arXiv preprint arXiv:2107.00680 (2021).

[3] T. Lattimore, C. Szepesvári, Bandit algorithms, Cambridge University Press, 2020.

[4] L. Li, W. Chu, J. Langford, R. E. Schapire, A contextual-bandit approach to personalized news article recommendation, in: Proceedings of the 19th international conference on World wide web, 2010, pp. 661–670.

[5] S. Sajeev, J. Huang, N. Karampatziakis, M. Hall, S. Kochman, W. Chen, Contextual bandit applications in a customer support bot, in: Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining, 2021, pp. 3522–3530.

[6] S. Upadhyay, M. Agarwal, D. Bounneffouf, Y. Khazaeni, A bandit approach to posterior dialog orchestration under a budget, arXiv preprint arXiv:1906.09384 (2019).

[7] O. Chapelle, L. Li, An empirical evaluation of thompson sampling, Advances in neural information processing systems 24 (2011).

[8] T. Graepel, J. Quiñonero Candela, T. Borchert, R. Herbrich, Web-scale bayesian click-through rate prediction for sponsored search advertising in microsoft's bing search engine, in: Proceedings of the 27th International Conference on Machine Learning ICML 2010, Invited Applications Track (unreviewed, to appear), 2010.

[9] I. Osband, B. Van Roy, Bootstrapped thompson sampling and deep exploration, arXiv preprint arXiv:1507.00300 (2015).

[10] B. Kveton, C. Szepesvari, S. Vaswani, Z. Wen, T. Lattimore, M. Ghavamzadeh, Garbage in, reward out: Bootstrapping exploration in multi-armed bandits, in: International Conference on Machine Learning, PMLR, 2019, pp. 3601–3610.

[11] C. Riou, J. Honda, Bandit algorithms based on thompson sampling for bounded reward distributions, in: Algorithmic Learning Theory, PMLR, 2020, pp. 777–826.

[12] N. Srinivas, A. Krause, S. M. Kakade, M. Seeger, Gaussian process optimization in the bandit setting: No regret and experimental design, arXiv preprint arXiv:0912.3995 (2009).

[13] S. R. Chowdhury, A. Gopalan, On kernelized multi-armed bandits, in: International Conference on Machine Learning, PMLR, 2017, pp. 844–853.

[14] J. Hensman, N. Fusi, N. D. Lawrence, Gaussian processes for big data, arXiv preprint arXiv:1309.6835 (2013).

[15] B. M. Steele, Exact bootstrap k-nearest neighbor learners, Machine Learning 74 (2009) 235–255.

[16] G. Biau, F. Cérou, A. Guyader, On the rate of convergence of the bagged nearest neighbor estimate., Journal of Machine Learning Research 11 (2010).

[17] Y. A. Malkov, D. A. Yashunin, Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs, IEEE transactions on pattern analysis and machine intelligence 42 (2018) 824–836.

[18] J. Johnson, M. Douze, H. Jégou, Billion-scale similarity search with GPUs, IEEE Transactions on Big Data 7 (2019) 535–547.

[19] R. Guo, P. Sun, E. Lindgren, Q. Geng, D. Simcha, F. Chern, S. Kumar, Accelerating large-scale inference with anisotropic vector quantization, in: International Conference on Machine Learning, PMLR, 2020, pp. 3887–3896.

[20] B. W. Silverman, Density estimation for statistics and data analysis, Routledge, 2018.

[21] D. Zhou, L. Li, Q. Gu, Neuralucb: Contextual bandits with neural network-based exploration (2019).

[22] D. Dua, C. Graff, UCI machine learning repository, 2017. URL: http://archive.ics.uci.edu/ml.

[23] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al., Language models are unsupervised multitask learners, OpenAI blog 1 (2019) 9.