

Evaluation of Explore-Exploit Policies in Multi-result Ranking Systems

Dragomir Yankov
dragoy@microsoft.com
Microsoft Corporation

Pavel Berkhin
pavelbe@microsoft.com
Microsoft Corporation

Lihong Li
lihongli@microsoft.com
Microsoft Corporation

ABSTRACT

We analyze the problem of using Explore-Exploit techniques to improve precision in multi-result ranking systems such as web search, query autocompletion and news recommendation. Adopting an exploration policy directly online, without understanding its impact on the production system, may have unwanted consequences - the system may sustain large losses, create user dissatisfaction, or collect exploration data which does not help improve ranking quality. An offline framework is thus necessary to let us decide what policy and how we should apply in a production environment to ensure positive outcome. Here, we describe such an offline framework.

Using the framework, we study a popular exploration policy — Thompson sampling. We show that there are different ways of implementing it in multi-result ranking systems, each having different semantic interpretation and leading to different results in terms of sustained click-through-rate (CTR) loss and expected model improvement. In particular, we demonstrate that Thompson sampling can act as an online learner optimizing CTR, which in some cases can lead to an interesting outcome: lift in CTR during exploration. The observation is important for production systems as it suggests that one can get both valuable exploration data to improve ranking performance on the long run, and at the same time increase CTR while exploration lasts.

Categories and Subject Descriptors

H.4 [Information Systems Applications]: Miscellaneous

Keywords

explore-exploit, ranking, evaluation, thompson sampling

1. INTRODUCTION

We study “multi-result” ranking systems, i.e., systems which rank a number of candidate results and present the top N to the user. Examples of such systems are web search, query autocompletion (see Figure 1), news recommendation, etc. This is in contrast to “single-result” ranking systems which also internally utilize ranking mechanisms but in the end display only one result to the user.

One challenge with ranking systems in general is their counterfactual nature [2]: We cannot directly answer questions of the sort “Given a query, what would have happened if we had shown a different set of results?” as this is counter the fact. The fact is that we showed whatever results the current production model considered best. Learning new models is thus biased and limited by the deployed ranking model. One sound and popular approach to breaking the dependence on an already deployed model is to integrate an Explore-Exploit (EE) component into the production system [12]. Exploration allows for occasionally randomizing the results presented to the user by overriding some of the top choices of the deployed model and replacing them with potentially suboptimal results. This leads to collecting in our data certain random results generated with small probabilities. When training subsequent ranking models, these results are often assigned higher weights, inversely proportional to the probabilities with which they were explored [2, 17]. As theoretically justified and empirically demonstrated, exploration usually allows better models to be learned. However, adopting exploration in a production system prompts a set of essential questions: which EE policy is most suitable for the system; what would be the actual cost of running EE; and most importantly, how to best use the exploration data to train improved models and what improvements are to be expected?

Here, we present an offline framework which allows “replaying” query logs to answer counterfactual questions. The framework can be used to answer the above exploration questions, allowing one to compare different EE policies prior to their integration in the online system. This is very important as running an inadequate policy online can quickly lead to significant money loss, cause broad user dissatisfaction, or collect exploration data which is altogether useless in improving the ranking model.

As a concrete example, we use the offline framework to evaluate Thompson sampling, a popular EE method which is simple to implement and is very effective at trading off exploration and exploitation [4, 16, 18]. We point out that in fact there are multiple ways of implementing Thompson sam-

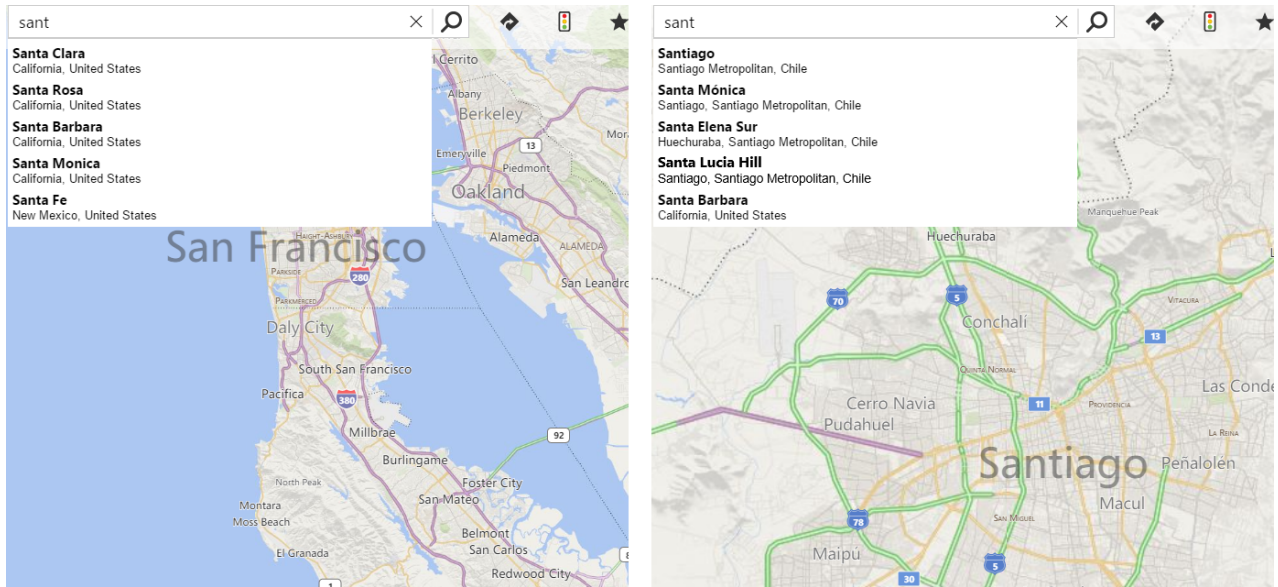


Figure 1: Maps query autocomplete system. For the same (query) prefix “santa” the system ranks on top geo-results which it deems relevant to the user context. *Left*: San Francisco, *Right*: Santiago.

pling, each having different semantic interpretation. Some of the implementations correct for bias (calibration problems) in the ranking model scores while others correct for position bias in the results. Naturally, employing different strategies leads to different costs, i.e. the price to be paid for exploring suboptimal results, and to different model improvements. We also introduce two schemes for weighting of examples, collected through exploration, during training new ranking models.

Because EE can promote suboptimal results it is commonly presumed that production systems adopting it always sustain a drop in the click-through-rate (CTR) during the period of exploration. By analyzing Thompson sampling policies through our offline evaluation framework, we observe an interesting phenomenon: using the right implementation can, in fact, produce a lift in CTR of the production system. In other words, the gain is twofold - the system collects valuable training data and has an improved CTR while exploration continues.

To summarize, this paper makes the following contributions:

- We describe a novel framework for offline evaluation and comparison of explore-exploit policies in multi-result ranking systems.
- We introduce several new Thompson sampling implementations that are tailored to multi-result ranking systems, some more suitable in the case of ranking score bias and others in the case of position bias.
- We introduce two simple weighting schemes for training examples collected through Thompson sampling.
- Using the framework, we show that adopting the right policy can achieve exploration and increase the CTR of the production system during exploration.

The rest of the paper is organized as follows. In Section 2, we introduce the offline policy evaluation framework using

a maps query autocomplete system as an example. Section 3 discusses different ways of implementing Thompson sampling and weighting examples collected with them. Section 4 compares the introduced Thompson sampling policies. We finish by discussing related works in Section 5.

2. EXPLORE-EXPLOIT FRAMEWORK

2.1 A multi-result ranking system

As a working example we are going to look into the maps query autocomplete service of a popular map search engine (Figure 1). When users start typing a “query” in the system they are presented with up to $N = 5$ relevant *geo entities* as suggestions. If users click on one of the results, we assume that we have met their intent; if they do not, the natural question to ask is “Could we have shown a different set of results which would get a click?”. As pointed out in the introduction, the question is counterfactual and cannot be answered easily as it requires showing a different set of suggestions on exactly the same context. This section describes a framework that allows answering such counterfactual questions. We first go over some prerequisites.

In building ranking systems as above, the usual process goes roughly through the following three stages: 1) The query is matched against an index; 2) For all matched entities a first layer of ranking, L_1 ranker, is applied. The goal of the L_1 ranker is to ensure very high recall and prune the matched candidates to a more manageable set of say a few hundred results; 3) A second layer, L_2 ranker, is then applied which re-orders the L_1 results in a way to ensure high precision. There could be more layers with some specialized functionality but overall these three stages cover the three important aspects: matching, recall and precision.

Building the L_1 ranker is beyond the scope of this work. Here, we focus on methodologies for improving L_2 ranking, namely, precision of the system. We assume that there is a machine learned model powering the L_2 ranker, i.e. the

system adopts a learning-to-rank approach [3].

Let us now focus on the structure of the logs generated by the system as well as how an EE policy can be applied in it. Table 1 shows what information can be logged by a multi-result ranking system for a query.

Position (i)	Label (y)	Result(r)	Rank score (s)
$i = 1$	0	Suggestion 1	$s_1 = 0.95$
$i = 2$	0	Suggestion 2	$s_2 = 0.90$
$i = 3$	1	Suggestion 3	$s_3 = 0.60$
$i = 4$	0	Suggestion 4	$s_4 = 0.45$
$i = 5$	0	Suggestion 5	$s_5 = 0.40$

Table 1: Original system: example logs for one query.

Suppose that L_1 ranking extracted $M \geq N$ relevant results which then L_2 re-ranked and produced the top $N = 5$ suggestions from Table 1. We assume that, for at least a fraction of the queries, the suggestions from the table and the user actions are logged by the production system.

The first column in the table shows the ranking position of the suggested result. The *Label* column reflects the observed clicks — 1 if the result was clicked and 0 otherwise. The *Result* column contains some context about the result that was suggested, part of which is only displayed to the user and the rest is used to extract features to train the ranking model. The last column shows the score which the L_2 ranking model has assigned to the results.

2.2 Explore-Exploit in the online environment

We assume a relatively conservative exploration process taking place in the online environment. Namely, it is allowed to replace only suggestions appearing at position $i = N$. This is to ensure that we do not generate large user dissatisfaction by placing potentially bad results as top suggestions. For exploration in addition to the candidate at position $i = N$ we choose among the candidates which L_1 returns and L_2 ranks at positions $i = N + 1, \dots, i = N + t \leq M$, for some relatively small t . By doing so, we do not explore results which are very low down the ranking list of L_2 as they are probably not very relevant. Requiring that a candidate for exploration meets some minimum threshold for its ranking score is also a good idea. Naturally, if for a query there are less than $N = 5$ candidates, then no exploration takes place for it.

To enable EE online we also need to define a *policy* — a mechanism which selects with a certain probability a suggestion different from the one that the deployed ranking model would recommend. Different policies can be implemented. In Section 3 we study several such policies and how they can be simulated in an offline environment. We also discuss how the data collected from them can be weighted suitably for training better ranking models; details can be found in Section 3.2.

2.3 Offline simulation environment

Running the above EE process directly in the production environment can lead to costly consequences: it may start displaying inadequate results which can cause the system to sustain significant loss in CTR in a very short time. It is further unclear whether it will help us collect training examples that will lead to improving the quality of the ranking model. We therefore want to simulate the above online pro-

cess a priori in an offline system that closely approximates it. Here, we present such an offline system.

The main idea of the offline system is to mimic a scaled-down version of the production system. Specifically, we assume offline that our Autocomplete system displays $k < N$ results to the user instead of $N = 5$. Again, to replicate the online EE process from Section 2.2, different policies evaluated in the offline system will be allowed to show on its last position (i.e., on position k) any of the results from positions $i = k, \dots, N$.

To understand the offline process better, let us look into two concrete instantiations of the simulation environment which use the logged results from Table 1.

In the first instantiation, we set $k = 2$. It means that the offline system displays to the user two suggestions, as seen in Table 2. Position $i = 2$ is going to be used for exploration, and the result to be displayed will be selected among the candidates at position $i = 2, \dots, i = 5$.

	Pos	Label	Result	Score(s)	
Dis- played	$i = 1$	0	Sugg. 1	$s_1 = 0.95$	EE Can- didates
	$i = 2$	0	Sugg. 2	$s_2 = 0.90$	
	$i = 3$	1	Sugg. 3	$s_3 = 0.60$	
	$i = 4$	0	Sugg. 4	$s_4 = 0.45$	
	$i = 5$	0	Sugg. 5	$s_5 = 0.40$	

Table 2: Offline system with $k = 2$. Logs for one query derived from the logs from Table 1. Position $i = 2$ (in blue) is used for exploration.

In the second instantiation, we set $k = 3$; that is, the offline system is assumed to display three suggestions. Position $i = 3$ is used for exploration and the candidates for it are the results from the original logs at positions $i = 3, 4, 5$. This setting is depicted in Table 3.

	Pos	Label	Result	Score(s)	
Dis- played	$i = 1$	0	Sugg. 1	$s_1 = 0.95$	EE Can- didates
	$i = 2$	0	Sugg. 2	$s_2 = 0.90$	
	$i = 3$	1	Sugg. 3	$s_3 = 0.60$	
	$i = 4$	0	Sugg. 4	$s_4 = 0.45$	
	$i = 5$	0	Sugg. 5	$s_5 = 0.40$	

Table 3: Offline system with $k = 3$ suggestions. Logs for one query derived from the logs from Table 1. Position $i = 3$ (in blue) is used for exploration.

Suppose we use $k = 2$. Using only the production system we would display in our simulated environment “Suggestion 1” and “Suggestion 2” and we would not observe a click as the label in the logs for both position $i = 1$ and $i = 2$ is zero. Now suppose we use the described framework to compare two EE policies, π_1 and π_2 , each selecting a different result to display at position $i = 2$. For example, π_1 can select to preserve the result at position $i = 2$ (“Suggestion 2”) while π_2 can select to display instead the result at position $i = 3$ (“Suggestion 3”). Now we can ask the counterfactual, with respect to the simulated system, question “What would have happened had we applied either of the two policies?”. The answer is, with π_2 we would have observed a click, which we know from the original system logs (Table 1) and with π_1 we would not have. If this is the only exploration which we perform the information obtained with π_2 would be more

valuable and would probably lead to training a better new ranking model. Note also that applying π_2 would actually lead to a higher CTR than simply using the production system. This gives an intuitive idea of why CTR can increase during exploration as demonstrated in the evaluation in Section 4.2.

It should be noted that our simulation environment effectively assumes the same label for an item when it is moved to position k from another, lower position $k' > k$. Due to position bias, CTR of an item tends to be smaller if the item is displayed in a lower position. Therefore, our simulation environment has a one-sided bias, favoring the production baseline that collects the data. While the bias makes the offline simulation results less accurate, its one-sided nature implies the results which we show in Section 4.2 are conservative: if a new policy is shown to have a higher offline CTR in the simulation environment than the production baseline, its online CTR can only be higher in expectation.

3. THOMPSON SAMPLING FOR MULTI-RESULT RANKING

We now demonstrate the offline framework by comparing several implementations of Thompson sampling, a popular policy due to its simplicity and effectiveness [4, 16, 18]. In the process we identify two interesting observations. First, there are multiple ways to implement Thompson sampling for multi-result ranking problems. They have different interpretations and lead to different results. Second, if the “right” implementation for the problem at hand is selected, then Thompson sampling can refine the ranking model CTR estimates to yield better ranking results. The method then essentially works as an online learner, improving the CTR of the underlying model by identifying segments where the model is unreliable and overriding its choice with a better one. This in turn can lead to an important result — adopting EE can entail a twofold benefit: it can collect valuable data for improving ranking precision, and at the same time lift the CTR of the production system during the period of exploration.

Algorithm 1 outlines our *generic* implementation of Thompson sampling. The algorithm closely follows the implementation suggested by [4] with two subtle modifications: 1) defining exploration *buckets*¹ in line 1; and 2) sampling from the subset of buckets relevant only to the current iteration in line 4. We elaborate on these points below, and will show they are essential and lead to very different results with different instantiations. In the evaluation section we also discuss how we set the exploration constant ϵ (lines 9, 11) to update the parameters of the beta distributions.

3.1 Thompson sampling policies

Here we describe three policies based on Thompson sampling. Each is characterized by: 1) how it defines the buckets (line 1 in the algorithm); 2) What probability estimate is the bucket definition semantically representing.

Sampling over positions policy.

This is probably the most straight-forward, but not very effective implementation of Thompson sampling. It defines buckets over the ranking positions used for drawing exploration candidates. More specifically we have:

¹Aka “arms” in the multi-arm bandits literature.

Algorithm 1 Thompson Sampling for Multi-result Ranking

```

1: Define buckets:  $\mathbb{P} = \{P_1, P_2, \dots, P_n\}$ 
2: Initialize Beta distributions:  $\{B(\alpha_1, \beta_1), \dots, B(\alpha_n, \beta_n)\}$ 
    $\triangleright$  One per bucket
3: while (ExplorationIsEnabled) do
4:    $\hat{\mathbb{P}} \leftarrow \{P_{c_1}, \dots, P_{c_l}\} \subseteq \mathbb{P}$   $\triangleright$  Select the buckets
     involved in the current iteration
5:   Draw  $\theta_i \sim B(\alpha_i, \beta_i), \forall i \in \{c_1, \dots, c_l\}$ 
6:    $m \leftarrow \text{argmax}_i \theta_i$ 
7:   Display  $r_m$  for exploration  $\triangleright r_m$  is the result
     associated with the selected bucket  $P_m$ 
8:   if ( $r_m$  clicked) then
9:      $\alpha_m \leftarrow \alpha_m + \epsilon$   $\triangleright$  Increment with constant
10:  else
11:     $\beta_m \leftarrow \beta_m + \epsilon$   $\triangleright$  Increment with constant

```

Bucket definition: There are $n = N - k + 1$ buckets each corresponding to one of the candidate positions $i = k, \dots, i = N$. All of them can be selected in each iteration so $\hat{\mathbb{P}} = \mathbb{P}$. For instance, if we have instantiation as per Table 3 we would have three buckets, $n = 3$, for positions $i = 3, i = 4$, and $i = 5$ — the positions of the candidates for exploration.

Probability estimate: $P(\text{click}|i, k)$. In this implementation Thompson sampling estimates the probability of click given that a result from position i is shown on position k . This implementation allows for correction in the estimate of CTR per position. The approach allows also for correcting position bias. Indeed, results which are clicked simply because of their position may impact the ranking model and during exploration we may through them away eliminating their effect on the system. This makes the approach especially valuable in systems with pronounced position bias.

Sampling over scores policy.

In this implementation we define the buckets over the scores of the ranking model. Each bucket covers a particular score range. For simplicity let us assume that the score interval $[0, 1]$ is divided into one hundred equal subintervals one per each percentage point: $[0, 0.01), [0.01, 0.02), \dots, [0.99, 1]$. For the suggested division we have:

Bucket definition: There are $n = 100$ buckets one per score interval $P_1 = [0, 0.01), \dots, P_{100} = [0.99, 1]$. In each iteration only a small subset of these are active. In the example from Table 3 only the following three buckets are active $P_{c_1} = P_{61} = [0.60, 0.61), P_{c_2} = P_{46} = [0.45, 0.46)$, and $P_{c_3} = P_{41} = [0.40, 0.41)$. Suppose after drawing from their respective Beta distributions we observe that $m = 61$, i.e. we should show the first of the three candidates which turns out to result in a click. In this case, we update the positive outcome parameter for the corresponding Beta to $\alpha_{61} = \alpha_{61} + \epsilon$.

Probability estimate: $P(\text{click}|s, k)$. In this implementation Thompson sampling estimates the probability of click given a ranking score s for a result when shown at position k . In general, if we run a calibration procedure then the scores are likely to be close to the true posterior click probabilities for the results [15], but this is only true if we look at them agnostic of position. With respect to position $i = k$ they may not be calibrated. We can think of Thompson sampling as a procedure for calibrating the scores from the

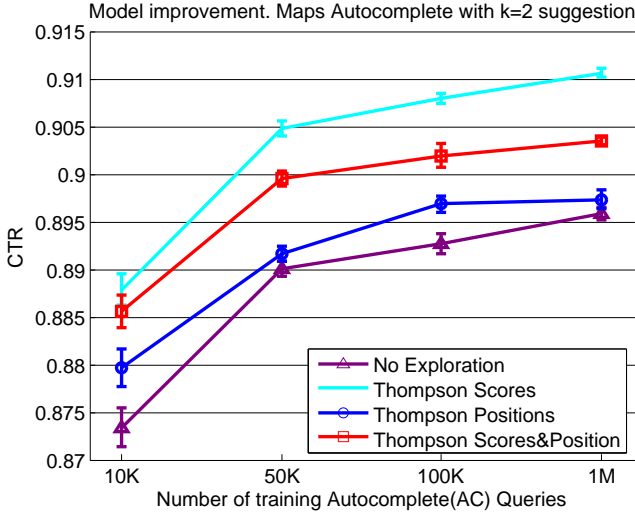


Figure 2: Model improvement for an Autocomplete system displaying $k = 2$ results to the user.

explored buckets to closely match the CTR estimate with respect to position $i = k$.

Sampling over scores and positions policy.

This is a combination of the above two implementations. Again we assume that the score interval is divided into one hundred equal parts $[0, 0.01], [0.01, 0.02], \dots, [0.99, 1]$. This, however, is done for each candidate position $i = k, \dots, i = N$. That is, we have:

Bucket definition: $(N - k + 1) * 100$ buckets. For more compact notation let us assume that bucket P_q^i covers entities with score in the interval $[s, s + 0.01)$ when they appear on position i (here $q = \lfloor 100s \rfloor + 1$). In the example from Table 3 we have $n = 300$ buckets and for the specific iteration the three buckets to perform exploration from are P_{61}^3 , P_{46}^4 , and P_{41}^5 .

Probability estimate: $P(\text{click}|s, i, k)$. In this implementation Thompson sampling estimates the probability of click given a ranking score s and original position i for a result when it is shown at position k instead. This differs from the previous case as in its estimate it tries to take into account the position bias, if any, associated with clicks.

These are definitely not all buckets that could be defined. Depending on the concrete system there may be others that are more suitable and lead to even better results. The point we are trying to convey is that how one defines buckets can vary and it plays a crucial role for the success of the exploration process.

3.2 Example weighting

Once we have an EE procedure in place a natural question to ask is how to best use the exploration data to train improved models. Here we introduce two schemes for assigning training weights to examples collected through exploration.

Propensity based weights.

In training new rankers it is a common practice to re-weight examples selected through exploration inversely proportional to the probability of displaying them to the user.

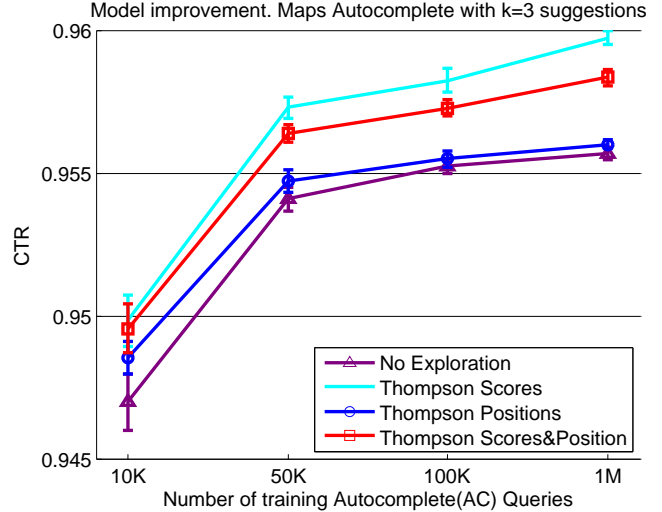


Figure 3: Model improvement for an Autocomplete system displaying $k = 3$ results to the user.

The probability of selecting an example for exploration is called *propensity score* [12]. More specifically, if we denote the propensity score for example x_j with $p(x_j)$, then its training weight is set to $w_j = \frac{1}{p(x_j)}$.

Computing propensity scores for Thompson sampling in the general case of more than two Beta distributions involved in each iteration ($l > 2$, line 5 in Algorithm 1) does not have an analytical solution [5]. We thus derive the following empirical estimate: if we draw for exploration x_j from bucket P_i (line 6 of the algorithm) then we set $p(x_j)$ to the ratio between examples that have been explored thus far from P_i over the sum of all examples explored from buckets in \mathbb{P} .

Multinomial weights.

We also analyze a different, very simple weighting scheme based on the scores of the baseline ranking model. Let x_i be the result displayed to the user from bucket P_i and let its ranking score be s_i . If x_j is the selected example for exploration then we first compute the “multinomial probability” $p(x_j) = \frac{s_j}{\sum_{i \in \{c_1, \dots, c_l\}} s_i}$. The weight is then computed again as the inverse proportional $w_j = \frac{1}{p(x_j)}$. If in Table 3 we have selected for exploration the example at position $i = 3$ then its probability is $\frac{0.6}{0.6+0.45+0.40} = \frac{0.6}{1.45}$ and the weight is $\frac{1.45}{0.6} = 2.41$. We call this weighting scheme *multinomial weighting*.

In both weighting schemes we cap the weight assigned to an example to avoid stressing excessively a single piece of evidence as suggested in previous work [2, 12, 17].

4. POLICY EVALUATION AND COMPARISON

The proposed offline framework allows us to evaluate different EE policies, such as the Thompson sampling policies described in the previous section. We mainly focus on two aspects: 1) Expected ranking model improvement; 2) Change in CTR during exploration. We start with our experimental setup.

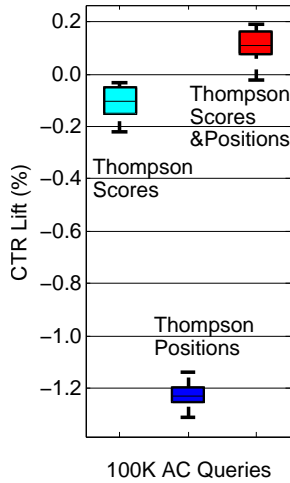


Figure 4: CTR lift during exploration for $k = 2$ - 100K and 1M AC Queries.

4.1 Evaluation setup

Data and Baseline Model: We use maps autocomplete (AC) logs spanning over two months collected by the production model. The first month is used to simulate on-line exploration, as described in Section 2.3, to compute the expected CTR loss and subsequently to train new ranking models. The second month is used to test and report performance results for the models trained on data from the first month.

Our baseline, *No Exploration* method (Figures 2 and 3), closely replicates the L_2 ranking model used in the production system — a variant of boosted regression trees [6]. The model adopts a point-wise learning-to-rank approach, that is it is trained on individual suggestions and assigns a clickability score to each result in scoring time optimizing squared loss. Point-wise models have been shown to be very competitive on a variety of ranking problems [3]. It is important to note, though, that the ideas presented here are agnostic to the exact L_2 ranker deployed in production. For instance, we obtained directionally similar results with more elaborate pairwise rankers optimizing NDCG [20].

Evaluation process: We measure the performance for datasets of sizes 10K, 50K, 100K and 1M queries which we sample uniformly at random from the training month. Keep in mind, that each query leads to up to five suggestions, i.e. for the largest set of 1M queries we train on over 2M individual examples. Each example is represented as a high dimensional vector of features capturing information about the result and the user context — static rank of the result, similarity to the query, distance to the user, etc.

For each of the query sets we run ten times the following steps, summarizing in the figures the mean and the variance across the ten runs.

Step 1. We choose a setting for the results to be displayed by the offline system $k = 2$ (Figures 2 and 4) and $k = 3$ (Figures 3 and 5).

Step 2. For the *No Exploration* baseline we take the top k suggestions and train the baseline model on them. If the production model is trained on only the top k results from its logs it will achieve exactly the same performance as the

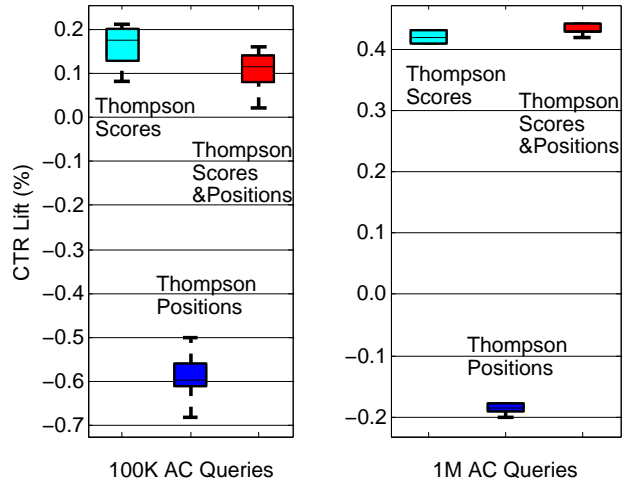


Figure 5: CTR lift during exploration for $k = 3$ - 100K and 1M AC Queries.

No Exploration model from the plots.

Step 3. On the same set of queries, used to train the baseline model, we run the discussed Thompson sampling policies, with n buckets as described in Section 3.1, dividing the scores interval into 100 equal subintervals as mentioned in the same section. The policies involve a stochastic step - each of them decides randomly to place on position $i = k$ one of the suggestions from positions $i = k, k + 1, \dots, N$, as described in Section 2.3. The resulting top k suggestions are what a scaled down version of the production system would log if it utilizes the respective policy. We use this data to train the same type of model configured identically as the baseline model. Therefore, the improvement observed on the test period can be attributed entirely to the exploration data collected through the policies.

Step 4. If a query has fewer than, or equal to, k results in the production logs we add it to the training set as is because in this case there are no results to explore from. We estimate that approximately 60% of the queries have more than two results and approximately 50% have more than three results. These are the queries which contribute to exploration for the two settings of $k = 2$ and $k = 3$ respectively.

Step 5. Finally, in training the models, we re-weight the examples collected with the exploration policies using the weighting schemes discussed in Section 3.2.

4.2 Evaluation of policies

Let us now discuss in some more detail the results from the different experiments.

Model improvement. Figures 2 and 3 show the model improvement achieved with different policies for $k = 2$ and $k = 3$ suggestions respectively. CTR is measured as the fraction of queries that would result in a click over all queries from the test month in an Autocomplete system that would display up to k suggestions.

Naturally, having more data to train on improves the performance of all models. The CTR for all models for $k = 2$ (Figure 2) ranges from 0.87 to 0.92 while for $k = 3$ it is between 0.94 and 0.96 (Figure 3). The larger CTR for $k = 3$ is due to two factors: 1) For $k = 3$ the system displays more results which are more likely to contain the clicked sugges-

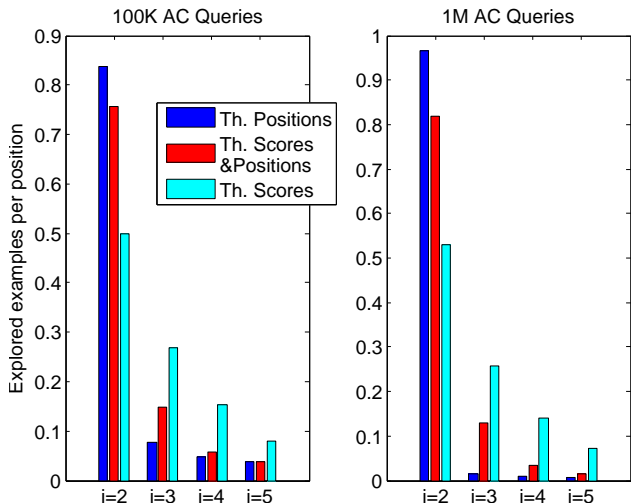


Figure 6: Examples explored from different positions for $k = 2$.

tion from the production logs; 2) There is difference in the number of training examples too. For instance, for $k = 2$ we still use 1M queries in the largest experiment, but each query contributes with up to two results, i.e. we have approximately 1.5M training examples in total. For $k = 3$ there are approximately 2.5M examples for the same set of queries.

We can see from the figures that in both cases the policies which account for bias in the model scores, *Thompson over Scores* (*Scores* for simplicity) and *Thompson over Scores&Positions* (*Scores&Positions* for simplicity), achieve large improvement over the production *No Exploration* system. The advantage of the *Scores* policy is especially striking for $k = 2$ yielding a consistent 1.5% model improvement across all query set sizes. In fact, using *Scores* or *Scores&Positions* with just 50K training queries results in models which have better test performance for both settings of k than the production *No Exploration* model trained on 1M queries!

What is the reason for *Scores* to perform better than *Scores&Positions*, and for both of them to outperform *Positions* which barely improves on the production model for 1M queries? We believe the answer is twofold.

First, there is a not too pronounced position bias associated with the problem. In web search often multiple results are relevant to a query. Showing them on different positions frequently yields different CTR. Here the intent is usually very well specified. Users are mostly interested in one particular geo-entity, e.g. a specific restaurant, park or a residential address. There are also only up to five suggestions per query which are easy to inspect visually.

Second, there is very stable CTR associated with each position. Higher positions have higher CTR. Thompson sampling policies that utilize position information quickly identify this and start apportioning most of the exploration to the optimal position. Therefore, exploration stops too early.

Exploration rate ϵ . For all experiments we use exploration rate $\epsilon = 1$ (line 9, 11 in Algorithm 1). Decreasing ϵ forces Thompson sampling to continue exploration longer. As we mentioned above, *Positions* stops exploration too

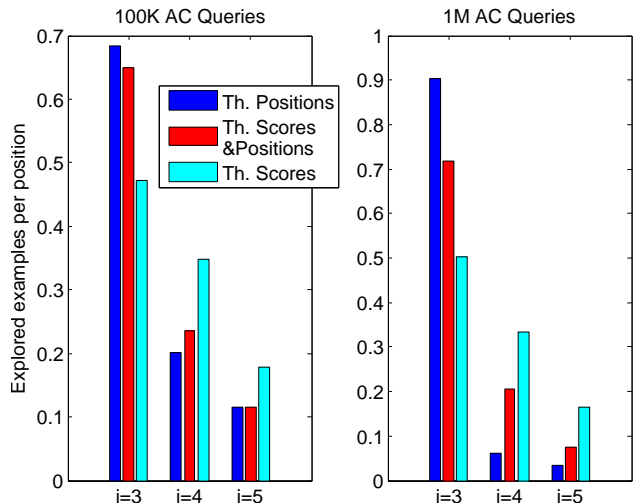


Figure 7: Examples explored from different positions for $k = 3$.

early. To force further exploration, for this policy only, we decrease the rate to $\epsilon = 0.01$. Still, as we see in Figures 2 and 3 *Positions* only marginally improves on *No Exploration* for 1M examples. It should be noted that one cannot decrease ϵ dramatically because then there is significant drop in CTR during exploration, which might be too steep of a price to pay for model improvement.

Weighting scheme. We repeated all experiments from Figures 2 and 3 with both weighting schemes discussed in Section 3.2. Though propensity based weighting is considered a more suitable weighting scheme in the literature [14], for our multi-result ranking problem we observed a different outcome. For $k = 2$ the multinomial weighting produced over 0.5% improvement compared to propensity weights. Multinomial weighting was also slightly better, though less pronounced, in the case of $k = 3$. All model improvement results presented here reflect multinomial weighting.

CTR lift in exploration. So far we saw that we can train models that outperform the production one by using exploration data from specific policies. We now look into what price the production system will pay for this model improvement. The common understanding is that as EE selects potentially suboptimal results production systems should always sustain a drop in CTR during the period of exploration.

Figure 4 and 5 demonstrate one of the major contributions of this work. They show that the above assumption is not necessarily true. On the contrary, if the right policy is selected multi-result ranking systems can even record a lift in their CTR. Here, *CTR lift during exploration* is defined as the CTR of each policy minus the CTR of the production *No Explore* system both computed during the training month.

The *Positions* policy indeed impacts the CTR of the production system negatively: 1.2% and 0.3% drop in CTR for $k = 2$, 100K and 1M queries Figure 4, and 0.6% and 0.2% drop in CTR for $k = 3$ Figure 5. The results are for $\epsilon = 0.01$, yet even for $\epsilon = 1$ we observe a drop in CTR.

With the *Scores* and *Scores&Positions* policies, however, we observe stable lift in CTR especially as the query set size grows to 1M. For $k = 2$ and dataset size of 1M queries the

policies improve the production CTR with 0.5% and 0.8% Figure 4. For $k = 3$ the lift is approximately 4.2% and 4.3% Figure 5. The reason that for $k = 3$ the lift is lower is due to the fact that in this setting there is a smaller pool of candidates for the system to explore — it runs exploration among only three positions $k = 3$, $k = 4$ and $k = 5$.

Finally, Figure 6 and Figure 7 show from which positions the selection was performed during exploration in the case of $k = 2$ and $k = 3$ respectively. As can be seen, *Positions* is very conservative and explores mostly the optimal position $i = 2$ in Figure 6 and $i = 3$ in Figure 7, when the number of examples increases to 1M, even though we have set $\epsilon = 0.01$. Another interesting observation is that *Scores&Positions* is more conservative, selecting fewer examples from suboptimal positions. As we saw above, this leads to a greater CTR lift during exploration than *Scores*. However, it also produces less of an improvement in the ranking model.

5. RELATED WORK

Explore-exploit techniques hold the promise of improving machine-learned models by collecting high-quality, randomized data. A common concern in production teams, however, is that they may invest resources integrating EE in their systems, suffer high cost during the EE period, and end up with data that does not lead to substantially better models. To address this concern multiple efforts have focused on building offline systems that try to quantify a priori the effects that EE will have on the system [2, 10, 12–14, 17]. All of these works focus on how data collected with a production model or another policy [10] can be used to estimate a priori the performance of a new policy. Under the assumption of stationary data distribution, it can be proved that weighting data inversely proportional to the propensity scores leads to unbiased offline estimators, i.e. models for which we can provide guarantees will behave in a certain way in production [2, 17]. Many recent works adopt the evaluation approach (e.g., [2, 14]). While theoretically sound, these offline frameworks make a few assumptions which may not always be present. For instance, as we noted earlier, in certain cases the propensity scores can not be computed in closed form, e.g. in Thompson sampling [5], so one needs to use approximations as the one implemented here. Another problem lies in the fact that it is often impractical to assume stationary distribution — for instance we find in our data many seasonal queries, queries which result from hot geo-political news etc, all of which impact the CTR of the system significantly.

Our work can be considered a special case of the generic *Contextual Bandit* framework [1, 11, 13, 19]. Unlike these works where the context is assumed to come in the form of additional observations [19] or features, e.g. personalized information [13], the context in our case is in the rich structure of the problem. For example, most of the works mentioned in this section focus on the single result case, i.e. there are k -arms to choose from but ultimately only one result is displayed to the user. We focus on multi-result ranking systems instead. As we saw here many real-world problems follow the multi-result ranking settings, which require special handling due to presence of position and ranking score bias. More recently [8] has discussed the multi-result setting. The authors describe a non-stochastic procedure for optimizing a loss function which is believed to lead to proper exploration. While the work is theoretically sound it does

not show whether the approach leads to improvement of the underlying model. The method also lacks some of the observed convergence properties of Thompson sampling which starts apportioning examples to buckets which it overtime finds likely to lead to higher CTR.

The effectiveness of Thompson sampling has been noted previously by [4, 7, 16] and others. Subsequently efforts have focused on understanding better the theoretical properties of the algorithm (e.g., [9]) leaving aside the important implementation considerations which we raised, namely, that in the context of multi-result ranking there are multiple ways to define the buckets (arms), and that different definitions lead to different semantic interpretation and different results.

6. CONCLUSION

We presented an offline framework which allows evaluation of EE policies prior to their deployment in an online environment. The framework allowed us to define and compare several different policies based on Thompson sampling. We demonstrated an interesting effect with significant practical implications. Contrary to the common belief, that a production system often has to pay a price of (possibly steep) CTR decrease during exploration, we show that the opposite can happen. If implemented suitably, a Thompson sampling policy can, in fact, have twofold benefits: it can collect data that improves the baseline model performance significantly and at the same time it can lift the CTR of the production system during the period of exploration.

7. REFERENCES

- [1] A. G. Barto and P. Anandan. Pattern-recognizing stochastic learning automata. *IEEE Transactions on Systems, Man, and Cybernetics*, 15(3):360–375, 1985.
- [2] L. Bottou, J. Peters, J. Quiñero Candela, D. X. Charles, D. M. Chickering, E. Portugaly, D. Ray, P. Simard, and E. Snelson. Counterfactual reasoning and learning systems: The example of computational advertising. *J. Mach. Learn. Res.*, 14(1):3207–3260, Jan. 2013.
- [3] O. Chapelle and Y. Chang. Yahoo! learning to rank challenge overview. *Journal of Machine Learning Research - Proceedings Track*, pages 1–24, 2011.
- [4] O. Chapelle and L. Li. An empirical evaluation of thompson sampling. In *Advances in Neural Information Processing Systems 24*, pages 2249–2257, 2011.
- [5] J. D. Cook. Numerical computation of stochastic inequality probabilities. *UT MD Anderson Cancer Center Department of Biostatistics Working Paper Series*, Working Paper 46, 2008.
- [6] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *Annals of Statistics*, 29:1189–1232, 2000.
- [7] T. Graepel, J. Q. Candela, T. Borchert, and R. Herbrich. Web-scale Bayesian click-through rate prediction for sponsored search advertising in Microsoft’s Bing search engine. In *Proceedings of the Twenty-Seventh International Conference on Machine Learning, ICML’10*.
- [8] S. Kale, L. Reyzin, and R. E. Schapire. Non-stochastic bandit slate problems. In *Advances in Neural*

- Information Processing Systems NIPS-2010*, pages 1054–1062, 2011.
- [9] E. Kaufmann, N. Korda, and R. Munos. Thompson sampling: An asymptotically optimal finite-time analysis. In *Proceedings of the 23rd International Conference on Algorithmic Learning Theory, ALT'12*, pages 199–213. Springer-Verlag, 2012.
 - [10] J. Langford, A. Strehl, and J. Wortman. Exploration scavenging. In *Proceedings of the 25th International Conference on Machine Learning, ICML '08*, pages 528–535, New York, NY, USA, 2008.
 - [11] J. Langford and T. Zhang. The epoch-greedy algorithm for multi-armed bandits with side information. In J. Platt, D. Koller, Y. Singer, and S. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 817–824. 2008.
 - [12] L. Li, S. Chen, A. Gupta, and J. Kleban. Counterfactual analysis of click metrics for search engine optimization. Technical report, Microsoft Research, 2014.
 - [13] L. Li, W. Chu, J. Langford, and R. E. Schapire. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th International Conference on World Wide Web, WWW '10*, pages 661–670, New York, NY, USA, 2010.
 - [14] L. Li, W. Chu, J. Langford, and X. Wang. Unbiased offline evaluation of contextual-bandit-based news article recommendation algorithms. In *Proceedings of the 4th ACM International Conference on Web Search and Data Mining, WSDM '11*, pages 297–306, 2011.
 - [15] A. Niculescu-Mizil and R. Caruana. Predicting good probabilities with supervised learning. In *Proceedings of the 22Nd International Conference on Machine Learning, ICML '05*, pages 625–632, 2005.
 - [16] S. L. Scott. A modern Bayesian look at the multi-armed bandit. *Applied Stochastic Models in Business and Industry*, 26:639–658, 2010.
 - [17] A. Strehl, L. Li, J. Langford, and S. M. Kakade. Learning from logged implicit exploration data. In *NIPS*, pages 2217–2225, 2011.
 - [18] W. R. Thompson. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3–4):285–294, 1933.
 - [19] C. Wang, S. R. Kulkarni, and H. V. Poor. Arbitrary side observations in bandit problems. *Adv. Applied Math*, 34:903–936, 2005.
 - [20] Q. Wu, C. J. Burges, K. M. Svore, and J. Gao. Adapting boosting for information retrieval measures. *Inf. Retr.*, 13(3):254–270, June 2010.