Recommending Accommodation Filters with Online Learning

Lucas Bernardi
lucas.bernardi@booking.com
Booking.com
Amsterdam, The Netherlands

Pablo Estevez
pablo.estevez@booking.com
Booking.com
Amsterdam, The Netherlands

Matias Eidis
matias.eidis@booking.com
Booking.com
Amsterdam, The Netherlands

Eqbal Osama
eqbal.osama@booking.com
Booking.com
Amsterdam, The Netherlands

ABSTRACT

Online Accommodations Platforms match guests searching for accommodation with hospitality service providers. A fundamental characteristic of efficient platforms is the ability to satisfy the needs and preferences of the guests. To achieve this goal, a common search tool is the Results Filtering capability which allows users to refine query results with explicit criteria. However, as supply grows and diversifies, more filtering options become available, reaching hundreds of different criteria for one query, and making it hard for customers to find the ones that are relevant to them. In this work we present the implementation of an Accommodation Filters Recommender System addressing this issue. The problem poses several challenges around recommendations feedback, user experience constraints, and non stationarity among others. We provide an end-to-end description of the System, discuss implementation issues and provide techniques to address them including a large scale distributed online learning architecture. The solution was validated through several Online Controlled Experiments performed in Booking.com, a top Online Travel Agency serving millions of daily users, showing statistically significant results on various user behaviour metrics indicating a strong positive effect on User Engagement.

CCS CONCEPTS

• **Applied computing** → **Online shopping**; *E-commerce infrastructure*; • **Information systems** → **Collaborative filtering**; **Content ranking**; **Search interfaces**.

KEYWORDS

recommender systems, online machine learning, information filtering, distributed systems

Reference Format:
Lucas Bernardi, Pablo Estevez, Matias Eidis, and Eqbal Osama. 2020. Recommending Accommodation Filters with Online Learning. In *3rd Workshop on Online Recommender Systems and User Modeling (ORSUM 2020), in conjunction with the 14th ACM Conference on Recommender Systems, September 25th, 2020, Virtual Event, Brazil.*

ORSUM@ACM RecSys 2020, September 25th, 2020, Virtual Event, Brazil

1 INTRODUCTION

With the advent of e-commerce, customers have access to a broad supply when searching for an item to purchase, which brings many benefits, but also choice and information overload [10]. The prevalence of these issues in tourism has been highlighted in [18] and [20]. [8] gives an extensive revision of the literature on this topic. Given this context, the ability to apply filters such as *Pet Friendly Hotels* or *Breakfast Included* is an important tool benefiting all parties, helping users to browse a large supply of options as well as accommodation suppliers to better market their services and also the platform, by eliciting explicit guests preferences which can be used to further personalize the user experience, potentially increasing the probability of a purchase. On the other hand, the constant growth of available filters due to supply diversification (such as vacation rentals) and the increase of product details available to use as filtering criteria, defeats the very purpose of this tool. This motivates the need of filter recommendations that allow users to refine their query without scanning a long list of options. These recommendations can be displayed for example, on the side of the Search Results Page as a quick access shortlist of filters before the full list of options. In this paper we describe a Recommender System for the case of Accommodation Filters. The problem poses several common challenges like Large Scale, Continuous Cold Start [2] and Sparsity among others. Previous work has addressed some of these issues in similar scenarios (e.g. [9] [11] [16] and [3]), but our setting deviates from them and the standard Recommender Systems or Information Retrieval settings, mainly because the item space, the filters, are not what users are interested in, but a means to find accommodations that fit their preferences. This brings several uncommon challenges around feedback and user experience consistency. Our work focuses on the design of an integral system considering all the intricacies of a full recommender system relying on well established and effective techniques, putting focus on their composition into a fully functional system that allows us to experiment with different approaches. The contributions of this work are:

- A thorough description of a Recommender System successfully deployed in Booking.com helping millions of new users daily to browse millions of accommodations options
- An architecture for Online Recommender Systems capable of producing a clean, well-formed and reliable data stream, suitable for incremental learning.

- An algorithm-agnostic architecture for Large Scale Distributed Online Learning
- Online Controlled Experiments that demonstrate the effectiveness of our approach at improving the shopping experience through more effective filtering actions

The paper is organized as follows: Section 2 presents the problem, main challenges and related work. Section 3 describes the solution framework. Section 4 details the system architecture while Section 5 focuses on a Machine Learning Model as implemented in Booking.com. Section 6 reports experiments on their website and Section 7 concludes.

2 PROBLEM STATEMENT AND RELATED WORK

Our goal is to construct a system that recommends Accommodation Filters maximizing the utility users get from them with the following requirements: ability to quickly on-board new filters under specific User Experience Guidelines, in particular, browsing consistency; it must scale to hundreds of millions of daily users and thousands of filters; it must be available at all times, globally, with latency $< 100ms$. The problem poses many challenges, we highlight three of them that we consider of particular relevance and strongly influenced the design of our architecture.

User Feedback: Implicit, Delayed and Split. We elicit Filter Utility from implicit signals based on user behaviour such as applying a filter, clicking on an accommodation to see further details, or completing a reservation. Several characteristics of these signals make learning from them challenging. First, the feedback is delayed with delays ranging from minutes to days, and since the negative feedback can only be implicitly assumed by the lack of positive feedback after certain time elapsed, it is necessary to model the delay (at least in principle), this has been studied by [5] and more recently by [12] and references therein. Delayed Feedback requires to keep track of the state of each impression which at the scale of large e-commerce platforms is an engineering challenge. Another issue is the *Split Feedback Problem*: in the classic recommender system setting, the feedback indicates both the degree of satisfaction of a specific item, and which item is receiving such feedback. In our case, these two pieces of information are split; first we observe that the user applied a filter, but only later, when and *if* the user clicks through or completes a transaction we observe the utility. To the best of our knowledge this problem has not been explicitly addressed by the relevant literature.

User Experience Consistency. Filter recommendations impose certain consistency constraints. In principle, it is critical that the recommendations don't flicker with each page load, but there are some nuances since in some situations the system *must* return *the same* recommendations (e.g. right after the user applied a filter), but in other situations the system *can* return *new* recommendations (e.g. when the query dates change), and in others the system *must* return *new* recommendations (e.g. when the destination changes). Which variables trigger a new recommendation is a design choice that trades off User Experience Consistency with number of samples to learn from. Related work like [7] focuses on recommendations consistency under small deviations of the user profile while [15] studies the effect of recommending the same items multiple times and focuses on the diversification-accuracy trade off. To our knowledge, no previous work focused specifically on guaranteeing the system is compliant with the User Experience Consistency constraints.

Non Stationarity. Stationarity is a rather strong assumption. For example, the set of matching results after applying a filter depends on the number of available rooms, which are constantly booked, cancelled, and replenished. A filter giving very few results might suddenly match many options, drastically affecting its utility. Experiments in other areas such as the Ranking Algorithm or the User Interface (UI) also affect filters utility. Last, new filters affect the utility of other filters, for example, introducing a Family Friendly Property filter, reduced the utility of Family related facilities. Non Stationarity motivates exploration which introduces the exploration-exploitation trade off. This has been largely studied by the contextual-bandits literature in for example [14], [17] and more recently by [22] and references therein.

All these issues (and others, omitted due to space limitations) motivate the construction of a complex system that enables their systematic treatment. As proposed in [4], we adopt the Online Recommender System Setting that considers an event stream produced by user interactions, and relies on incremental algorithms. The following sections describe the architecture including components that produce a reliable data stream and components that host general incremental algorithms at scale, highlighting how they contribute to the solution of the Accommodation Filters Recommendations.

3 SOLUTION FRAMEWORK

The main abstraction in our system is the Feedback Loop, which models the interactions between the UI and our Recommender System. In its simplest version a Feedback Loop is defined by the following sequence:

(1) The UI requests recommendations (opens the loop).
(2) The System recommends filters based on the query, context and filter features.
(3) The UI provides feedback (closes the loop).
(4) The System uses closed loops to update the model.

This simple framework captures the dynamic nature of the problem providing flexibility to experiment with different ways to address the aforementioned issues and abstracting away specific implementations. The Feedback Loop sequence needs to be extended in order to address two issues. The first one is *Censored Recommendations*, which occurs when filters are not seen by the user (e.g. because they didn't scroll enough) introducing ambiguity since the lack of feedback might be caused by either dissatisfaction or by censorship. To solve this issue we introduce another step in between 2 and 3: the *exposure* report, which notifies the System that a user was exposed to a filter, indicating that lack of positive feedback implies negative feedback. If no exposure report is received the observation is not used for learning since the user never saw the recommendation. The second issue, rather specific to our setting, is *Split Feedback* which appears when the item and the level of satisfaction are reported separately. For example if the user applies a filter and then completes a reservation we would like to credit that filter with positive feedback. But these two events are usually

separated by many other browsing events (clicking back and forth the search results page), making it hard at reservation time to know which filters where applied by the user, and therefore impossible to send a feedback report. Another instance occurs when different UI components know different pieces of the feedback. This is the case of *after filtering click-through*: we would like to credit an applied filter if it leads to a click on a details card. Which filter is applied is known by the Search Engine but whether a click happens or not, is known by the Details Page Service which has no information about the applied filters. This is addressed by a technique that we call *Confirmable Feedback*, which splits the feedback report (step 3): first, the user interface reports *unconfirmed* feedback as soon as the filter is applied, indicating which item *might* be credited. The System waits for confirmation before using it to learn. The user interface sends the confirmation for example when a reservation is made or a details page loaded, without the need to know which filters were applied. As a result of a confirmation the loop is closed and the system can learn from it. If no confirmation is sent (e.g. the user applied a filter but no click or reservation happened), the system ignores the unconfirmed feedback. Finally, if a confirmation is received but not feedback was sent before, the confirmation is ignored.

The main benefit of this framework is that the state of a loop is maintained by the System, the UI is stateless and relies on four simple primitives: open loop, report exposure, report feedback and confirm feedback, requiring minimal interventions in the UI software which is a key factor to successfully deploy the full system into production.

4 ARCHITECTURE

The Architecture implements the Feedback Loops framework and consists of two main components described below.

Instrumentation Layer. This component implements the full Feedback Loops sequence, it is used by the UI to integrate the recommendations in the platform. Its first responsibility is serving recommendations requests using the Machine Learning Model (ML Model) and providing User Experience Consistency guarantees. User Experience Consistency presents a trade-off with sampling efficiency which is addressed by a technique we call *Contextual Caching*: the first time a specific user requests for recommendations, the ML Model is invoked, a new loop is initialized, and the recommended filters stored in a cache together with the query, context and filter features. Subsequent requests are served from the cache, but if at least one feature included in the *Refresh Trigger Feature Set* changes, the loop is finalized, the cache is invalidated, and a new loop is initialized with a new request to the ML Model. Notice that at most one loop is active for a given user at any point in time, and that one loop encapsulates information about the *set* of recommended filters. This approach allows us to control the trade-off by specifying which features must trigger a loop reset. The second responsibility is to produce a data stream based on the interactions with the UI, suitable to train a ML Model in an online fashion. The instrumentation layer maintains a state machine for each loop consisting of all the feature values when the loop was opened, which filters were recommended, which ones were actually seen by the user, their expiration status, the feedback received so far, etc. This state is updated as the UI reports exposure and feedback. From this transitions a data stream is produced containing all the information needed to train a machine learning model. This process is very dependent on the way we handle Delayed Feedback. We considered two approaches: Fully Delayed, where the state machine waits for a fixed period before assuming negative feedback, and Fake Negatives Calibration where the negative feedback is assumed on exposure as described in [12]. In our experiments both methods were effective showing no significant difference. We favored Fake Negatives Calibration due to its robustness to changes in the delay distribution.

The life-cycle of a loop is enforced prohibiting invalid state transitions. This brings high robustness to chaotic UI interactions (users refreshing pages, clicking copied links, multiple browser tabs, etc.), guaranteeing a well formed stream of events that can be used to train robust machine learning models in an online fashion.

Distributed Online Machine Learning. This component is responsible for maintaining a model to recommend filters when requested by the Instrumentation Layer and for updating it as soon as feedback is available. In order to handle high requests volume, we distribute the model in a cluster, each node runs one model instance serving a random sample of the recommendations requests (sharding) while learning from the full feedback stream produced by the Instrumentation Layer (replication), which is consumed from a persistent message queue. The feedback is consumed in the order it is produced, so although each node learns independently without node-node interaction, all the models are exact replicas. To achieve high availability, a special node saves checkpoints of the model to a persistent storage. The checkpoint contains the serialized model, learning state and a pointer to the last feedback message processed in the queue. If a node fails, a new one is created which reads the latest available version from the checkpoint and continues the learning process from the corresponding point in the stream.

This design is agnostic from concrete learning algorithms. Specific implementations can be constructed with little attention to fault tolerance, high availability and latency. This is an important advantage when deploying models to production since it simplifies algorithm development and debugging. Another important consequence is that all changes have an immediate effect, accelerating experimentation allowing us to quickly asses modeling techniques such as the Refresh Trigger Feature Set, the Delayed Feedback approach, etc. ultimately streamlining the iterative process.

5 MACHINE LEARNING MODEL

The model must select ~10 Filters out of ~20000 optimizing the total Utility. The latency requirement is strict, if fulfilling a request takes more than 100ms, it is canceled by the UI and no recommendations are shown to the user. This limit involves all the steps including network latency, Contextual Caching and Ranking. Because of this, we favor simple, fast and scalable models, in particular we rely on the library Vowpal Wabbit[13] embedded in an on-line process. We use a point-wise model estimating the expected utility conditioned on context, query and filter features. At recommendation time we rank by the estimated expected utility modeled with logistic regression. Most of the users are not logged-in while browsing, which means user history is not available, therefore we rely on query, context and

item features. Example Query Features are number of guests, destination, number of children, anticipation, length of stay, traveler type (couple, family, etc.); Contextual Features include temporal and geographical attributes; and some item features are Category, Id and Number of Matching Properties. The *Refresh Trigger Feature Set* contains all the query features except anticipation plus all the temporal features. Quadratic interaction features are created between the Filter and the Query and Context Features, keeping the linear and independent terms. The full model with Fake Negatives Calibration[12] is given by Equation 3, where $r \in \{0, 1\}$ is the binary output, f, q and c are the Filter, Query and Context feature vectors with dimensions d_f, d_q and d_c respectively. All the Greek letters are model parameters. $\theta_0 \in \mathbb{R}$ is the global bias, $\theta \in \mathbb{R}^{d_f}$, $\omega \in \mathbb{R}^{d_q}$, $\phi \in \mathbb{R}^{d_c}$ are linear parameters, $\alpha \in \mathbb{R}^{d_q x d_f}$ and $\beta \in \mathbb{R}^{d_c x d_f}$ are the interaction weights of Query and Context with Filter features.

$$z(f,q,c) = \theta_0 + \sum_i^{d_f} f_i\theta_i + \sum_i^{d_q} q_i\omega_i + \sum_i^{d_c} c_i\phi_i + \sum_i^{d_q}\sum_j^{d_f} q_i f_j \alpha_{ij} + \sum_i^{d_c}\sum_j^{d_f} c_i f_j \beta_{ij} \tag{1}$$

$$b(f,q,c) = \frac{1}{1+e^{-z(f,q,c)}} \tag{2}$$

$$\hat{p}(r=1|f,q,c) = \frac{b(f,q,c)}{1-b(f,q,c)} \tag{3}$$

The filter representation includes the unique id which allows the model to learn the specifics of the particular filter option, the number of matching options, which captures contextual utility (a generally very useful filter in a context where it matches many properties might be less useful) and the category (e.g. Facilities) that captures general properties across filters in the same category, which is effective to address the cold start problem: when a new filter is added, it inherits what is known about its category. In practice, all the features except Number of Matching Properties are categorical and are encoded using the hashing trick [21]. We estimated about 8 millions features (including interactions), following [1] we used a hashing space of 2^{28} buckets which resulted in about 3% collision rate. All parameters are learned through Stochastic Gradient Descent with constant learning rate and Normalized Updates [19] (Algorithm 1).

To address non stationarity, we rely on active exploration for which we adopt the contextual ϵ-greedy algorithm [6] in which a proportion ϵ of the recommendation requests (after Contextual Caching) is served with random uniform recommendations (exploration), and the rest by choosing the best according to the estimations of the model (exploitation). Two small adaptations are required for our case. First, since our system recommends many items, the exploration branch is computed by sampling from a uniform distribution between 0 and 1 for each candidate filter, and returning the top-k filters. The exploitation branch, simply sorts the candidate filters by their estimated utility given the context and query, and returns the top-k. Second, the model cannot be updated right after the recommendations are made since the feedback is delayed. We update the model independently from the recommendations requests, as soon as the feedback is available. All weights are initialized with zero and ties are broken randomly uniformly (see Algorithm 1).

Algorithm 1 top-k Delayed Contextual ϵ-Greedy

ϵ: exploration factor
k: number of filters to recommend
c, q: context and query features
F: set of candidate filters
procedure RECOMMENDFILTERS(ϵ, k, c, q, F)
  With probability ϵ: ▷ (explore)
  for each filter F_i in F with features f **do**
    $scores[F_i] \leftarrow$ sample from Uniform(0,1)
  end for
  or with probability $1-\epsilon$: ▷ (exploit)
  for each filter F_i in F with features f **do**
    $scores[F_i] \leftarrow \hat{p}(r=1|f,q,c)$ ▷ as given by Eq. 3
  end for
  return topk($scores, k$) ▷ (breaks ties uniformly)
end procedure
c, q: context and query features, f: rewarded filter, r_o: observed feedback
procedure ONFEEDBACKAVAILABLE(c, f, q, r_o)
  Update $\hat{p}$(r=1|c, q, f) with r_o ▷ detailed update rule in Algorithm 1 in [19])
end procedure

6 EXPERIMENTS

We performed several Online Controlled Experiments to validate our approach. We highlight 2 where our model was used to feed a *quick filters* section of the search results page of Booking.com. 50% of the traffic was exposed to the current baseline model, and 50% to our new model. Statistical significance was computed at 90% confidence (two-sided) using g-test at a predefined time duration. In these experiments we used click after filtering as feedback with $\epsilon = 2\%$. The baseline is a popularity model based on the same features with a thick layer of business logic on top, blacking out some filters, up-ranking others, etc. It is based on years of data on filter usage and it is updated manually at arbitrary moments. It is considered a robust baseline since many attempts of using more complex models failed in the past. The metrics of interest for these experiments are: Overall Filter Usage (proportion of users using at least one filter), Recommended Filter Usage (proportion of users using at least one *recommended* filter), After Filtering Click Through Rate (AFCTR, proportion of users filtering (any filter) and landing on a property detail page), Recommended Filter Utility (ratio between number of users applying a *recommended* filter and number of users applying any filter). All these metrics indicate the utility of the recommendations from the users point of view. In particular, Recommended Filter Utility is a strong indicator since it quantifies how easy is for customers to find a relevant filter.

First, an experiment was run for two weeks to validate the technical health of the system. The variance of the cluster was very

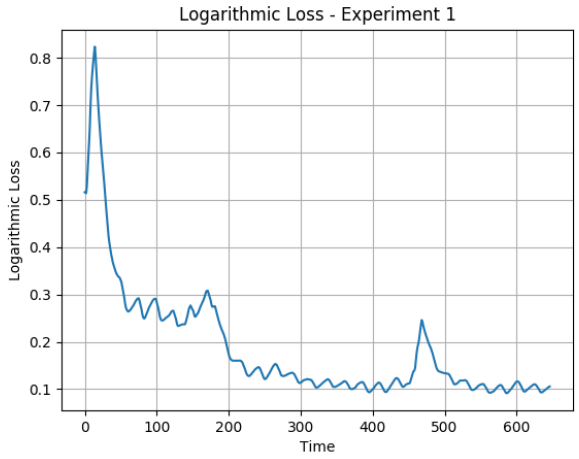

Figure 1: Logarithmic loss of model predictions.

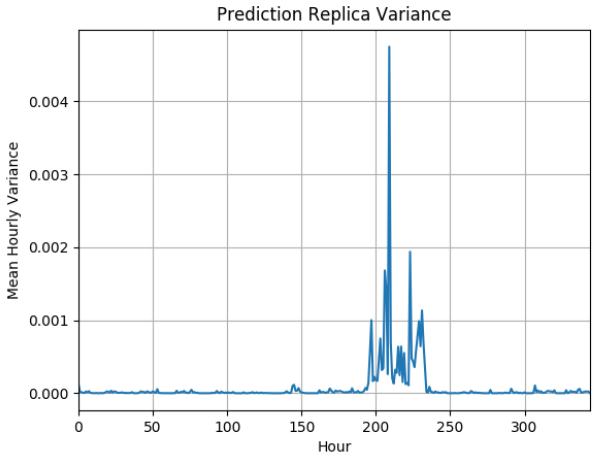

Figure 2: Prediction variance of replicas across 6 nodes.

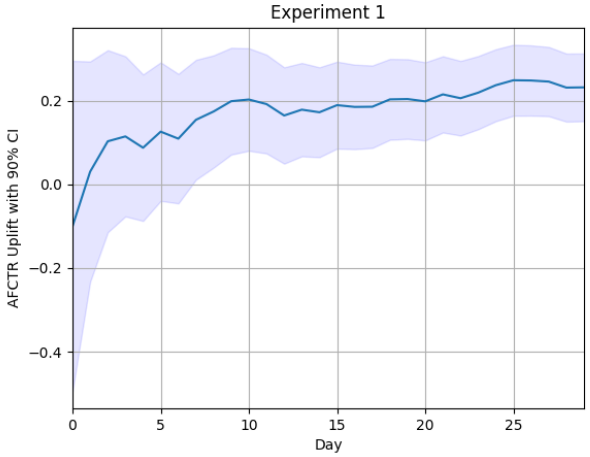

Figure 3: Cumulative Uplift w.r.t. Baseline on After Filtering CTR with 90% CIs

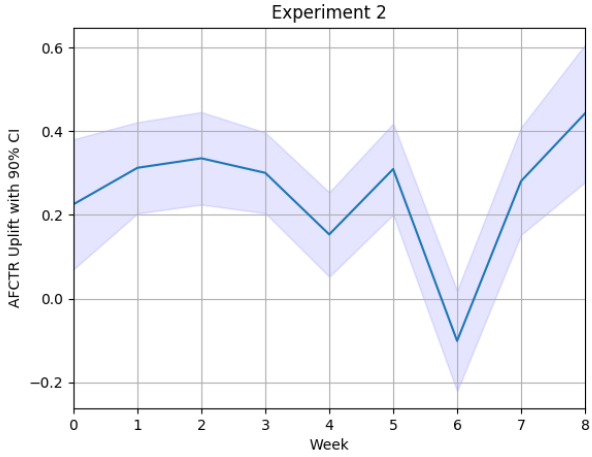

Figure 4: Weekly Uplift w.r.t. Baseline on After Filtering CTR with 90% CIs

Table 1: Experiments results with 90% CIs. All statistical significant with p-value < 0.001.

Metric Uplift w.r.t. Baseline (%)	Exp 1. After 2 weeks	After 4 weeks	Exp. 2 After 2 weeks	After 4 weeks
Overall Filter Usage	0.26 ± 0.06	0.22 ± 0.04	0.19 ± 0.06	0.22 ± 0.03
Recommended Filter Usage	37.59 ± 0.1	40.85 ± 0.08	38.02 ± 0.1	32.98 ± 0.05
After Filtering CTR	0.19 ± 0.1	0.23 ± 0.08	0.30 ± 0.1	0.24 ± 0.05
Recommended Filter Utility	37.22 ± 0.1	40.53 ± 0.07	32.67 ± 0.1	37.75 ± 0.07

low, (median 8.2e-6, 99th percentile 0.0012) indicating that the models are indeed close replicas. Figure 2 shows the hourly average for the first 2 weeks of the experiment. The peaks around hour 220 are due to one of the nodes lagging behind (due to uneven resource allocation which is dynamic and not uniform across the cluster), but quickly caught up. The model starts with all the weights set to 0 (making random recommendations), so we measured the time to learn reasonable recommendations defined as the moment where the AFCTR matches the baseline (y=0 in Figure 3) which was roughly 20 hours. This is evidence of the ability of the model to effectively incorporate new filters and contexts since at the beginning, *all* filters and features are new for the model. Logarithmic

Table 2: Challenges and corresponding techniques applied in our system.

Problem	Techniques
User Experience Consistency	Contextual Caching
Delayed Feedback	Fake Negatives Calibration [12]
Censored Recommendations	Exposure Reports
Split Feedback	Confirmable Feedback
Non Stationarity	Online Learning and ϵ-greedy
Low Latency	Sharded Inference, Replicated Learning
High Availability	Replicated Learning, Redundant Checkpoints
Continuous Cold Start	ϵ-greedy and Item Features

loss (Figure 1) showed a periodic pattern, likely following the general probability of clicking after filtering during the day. The peaks are likely due to changes in the environment. The general trend is negative, suggesting that the system is able to adapt to changes. We remark that many new filters were added while this experiment was running, and some of them were consistently picked in the top 10. Regarding the latency requirement, we observed a degradation in total page load time of 15ms which is inline with the requirements. We let the experiment run for two more weeks to make sure results are stable.

In a second experiment we stress-test the adaptability of the system by allowing automated traffic from scrapers and crawlers for a few days during week 6 which completely changes the utility of almost all filters (the proportion of automated traffic was significant). The system degraded but after normalization of the traffic it recovered, we interpret this as evidence of adaptability to changes in the environment. This behaviour is depicted in Figure 4.

Results of both experiments are summarized in Table 1. From these results we conclude that our system is able to make recommendations that are useful for our customers and superior to the ones by the baseline model. We also conclude that the system is reliable, stable and robust, meeting all the requirements specified in Section 2.

7 CONCLUSION

This work presented a Recommender System for Accommodation Filters, a relevant problem for Booking.com and other e-commerce platforms. Our solution features the implementation of well established techniques for which practical aspects were discussed in detail, and new ideas addressing several setting-specific problems. The Feedback Loops Framework and the Distributed Online Learning Learning Architecture allowed us to address requirements and trade-offs in a systematic way enabling a fast iterative process. The effectiveness of our solution was demonstrated by Online Controlled Experiments conducted in Booking.com, on millions of users and accommodations options, showing clear positive effects on User Engagement. Table 2 summarizes challenges and techniques addressing them. Looking forward, the presented system will allow us to experiment with more advanced online learning algorithms such as tree based models and more sophisticated exploration techniques and to solve new and different business cases.

REFERENCES

[1] Lucas Bernardi. 2018. Don't be tricked by the Hashing Trick. https://booking.ai/dont-be-tricked-by-the-hashing-trick-192a6aae3087 Retrieved 2020-06-16.

[2] Lucas Bernardi, Jaap Kamps, Julia Kiseleva, and Melanie JI Müller. 2015. The continuous cold start problem in e-commerce recommender systems. *arXiv preprint arXiv:1508.01177* (2015).

[3] Lucas Bernardi, Themistoklis Mavridis, and Pablo Estevez. 2019. 150 successful machine learning models: 6 lessons learned at booking. com. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. 1743–1751.

[4] Shiyu Chang, Yang Zhang, Jiliang Tang, Dawei Yin, Yi Chang, Mark A Hasegawa-Johnson, and Thomas S Huang. 2017. Streaming recommender systems. In *Proceedings of the 26th International Conference on World Wide Web*. 381–389.

[5] Olivier Chapelle. 2014. Modeling Delayed Feedback in Display Advertising. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (New York, New York, USA) *(KDD âĂŹ14)*. Association for Computing Machinery, New York, NY, USA, 1097âĂŞ1105. https://doi.org/10.1145/2623330.2623634

[6] David Cortes. 2018. Adapting multi-armed bandits policies to contextual bandits scenarios. *ArXiv* abs/1811.04383 (2018).

[7] P. Cremonesi and R. Turrin. 2010. Controlling Consistency in Top-N Recommender Systems. In *2010 IEEE International Conference on Data Mining Workshops*. 919–926.

[8] Basak Denizci Guillet, Anna Mattila, and Lisa Gao. 2019. The effects of choice set size and information filtering mechanisms on online hotel booking. *International Journal of Hospitality Management* (2019), 102379.

[9] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. 2014. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*. 1–9.

[10] Sheena S Iyengar and Mark R Lepper. 2000. When choice is demotivating: Can one desire too much of a good thing? *Journal of personality and social psychology* 79, 6 (2000), 995.

[11] Rishabh Iyer, Nimit Acharya, Tanuja Bompada, Denis Charles, and Eren Manavoglu. 2018. A Unified Batch Online Learning Framework for Click Prediction. *arXiv preprint arXiv:1809.04673* (2018).

[12] Sofia Ira Ktena, Alykhan Tejani, Lucas Theis, Pranay Kumar Myana, Deepak Dilipkumar, Ferenc Huszár, Steven Yoo, and Wenzhe Shi. 2019. Addressing delayed feedback for continuous training with neural networks in CTR prediction. In *Proceedings of the 13th ACM Conference on Recommender Systems*. 187–195.

[13] John Langford, Lihong Li, and Alex Strehl. 2007. Vowpal wabbit online learning project. http://hunch.net/?p=309

[14] John Langford and Tong Zhang. 2007. The epoch-greedy algorithm for contextual multi-armed bandits. In *Proceedings of the 20th International Conference on Neural Information Processing Systems*. Citeseer, 817–824.

[15] Neal Lathia, Stephen Hailes, Licia Capra, and Xavier Amatriain. 2010. Temporal diversity in recommender systems. In *Proceedings of the 33rd international ACM SIGIR conference on Research and development in information retrieval*. 210–217.

[16] Cheng Li, Yue Lu, Qiaozhu Mei, Dong Wang, and Sandeep Pandey. 2015. Click-through prediction for advertising in twitter timeline. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. 1959–1968.

[17] Lihong Li, Wei Chu, John Langford, and Robert E Schapire. 2010. A contextual-bandit approach to personalized news article recommendation. In *Proceedings of the 19th international conference on World wide web*. 661–670.

[18] Jeong-Yeol Park and SooCheong Shawn Jang. 2013. Confused by too many choices? Choice overload in tourism. *Tourism Management* 35 (2013), 1–12.

[19] Stéphane Ross, Paul Mineiro, and John Langford. 2013. Normalized online learning. *arXiv preprint arXiv:1305.6646* (2013).
[20] Nguyen T Thai and Ulku Yuksel. 2017. What can tourists and travel advisors learn from choice overload research? *Consumer Behavior in Tourism and Hospitality Research* (2017), 1.
[21] Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola, and Josh Attenberg. 2009. Feature hashing for large scale multitask learning. In *Proceedings of the 26th annual international conference on machine learning*. 1113–1120.
[22] Qingyun Wu, Naveen Iyer, and Hongning Wang. 2018. Learning contextual bandits in a non-stationary environment. In *The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval*. 495–504.