Beyond algorithms: Ranking at scale at Booking.com

Themis Mavridis*
themistoklis.mavridis@booking.com
Booking.com
Amsterdam, Netherlands

Soraya Hausl*
soraya.hausl@booking.com
Booking.com
Amsterdam, Netherlands

Andrew Mende*
andrew.mende@booking.com
Booking.com
Amsterdam, Netherlands

Roberto Pagano*
roberto.pagano@booking.com
Booking.com
Amsterdam, Netherlands

ABSTRACT

Booking.com is one of the world's largest online travel companies where millions of customers find their accommodation through an extensive and diverse list of properties including hotels, apartments, guest houses, and more. Our customers heavily rely on the ranking of search results in order to find a property that satisfies their preferences and criteria. While most of the Machine Learning literature focuses on the algorithmic aspect of ranking, not much has been published about the intricacies of developing a ranking that is Machine Learning powered and delivers business impact in a commercial environment. In this work, we describe the various challenges we faced while developing a large-scale Machine Learning powered ranking, including signal definition and feature representation complexity in the modelling aspect, information leakage and speed of innovation in experimentation for online evaluation, and response time minimization around serving at scale in production. We display an overview of the business gains of the step changes in the Machine Learning ranking of Booking.com. Our main conclusion is that there are multiple aspects that need to be carefully addressed for the creation of a Machine Learned ranker in a large-scale commercial setting.

CCS CONCEPTS

• **Computing methodologies** → **Machine learning**; • **General and reference** → **Experimentation**.

1 INTRODUCTION

Booking.com is one of the world's largest online travel companies where millions of customers search for their accommodation every day. Millions of accommodations of various types are listed on our platform and available to customers all around the globe. In the most popular destinations there are thousands of properties available for every search, ranging from private apartments, which somebody rents out while going on vacation, to traditional chain hotels. It is not reasonably possible for our customers to exhaustively review such a large list of options. Filtering, performed by the minority of our customers, can help narrow down the options to hundreds which is still a large amount. Therefore, our customers rely heavily on our ranking for finding the accommodation that is a good fit for them.

This work describes many challenges, decisions, and trade offs that creating and utilizing a large-scale Machine Learned ranker involves. The major contribution is to emphasize the aspects beyond the choice of algorithm that require attention to successfully develop Machine Learning powered ranking in a commercial environment. The rest of the paper is organized as follows: the Modelling section touches upon the challenges and design choices that creating a Machine Learned ranker entails, the Experimentation section focuses on the intricacies of evaluating a ranker in an online setting, the Serving section presents the problems and solutions to productionize a large-scale ranking system, the Quantifying business impact section displays the relative business value of the step changes introduced in the search results ranking of Booking.com, and a final section where we present our conclusions.

2 MODELLING

In order to create a ranker that satisfies the requirements of our customers on our scale, we need to learn over billions of user sessions. Training on billions of rows, with a complex feature space consisting of large amount of user, context and accommodation features is challenging and requires handling terabytes of data.

In two-sided marketplaces ranking has several objectives and typically a number of constraints. Besides surfacing accommodations best suitable for the customer, it is important to satisfy the needs of accommodation providers by delivering consistent level of exposure while also giving our partners tools to influence their visibility on the platform according to their sales strategy. For the purpose of this paper we will focus on optimisation for one objective which is by far the most important: catering to customers' preferences.

2.1 Signals Definition

In the course of a search session our users perform a variety of actions. The actions that are most important from the ranking perspective are: search results impressions, clicks, reservations, stays and reviews of a property. The proper definition of the positive and negative signals derived from these actions is of paramount importance.

There are four aspects that can help us decide which signals to use in our models; 1) relation to user satisfaction/dissatisfaction, 2) amount of data points, 3) delay to observe the action, 4) bias of the action. Analysis of different choices of signals against these aspects are presented in Table 1 for positive and Table 2 for negative ones.

Positive signals In our experience utilizing reservations as positive signals provides us with data points which have a strong

*All authors contributed equally to this research.

Positive Signal	Relation to user satisfaction	Amount of data points	Delay to observe signal	Bias
Property click	Weak	Large	Small: Users perform multiple clicks on properties while they browse	Clicking a property on search results does not necessarily translate to customer satisfaction
Property click with dwell time greater than threshold	Medium	Large/Medium (depending on the threshold)	Small: Users perform multiple clicks on properties while they browse	Spending time browsing does not necessarily translate to customer satisfaction
Reservation	Strong	Small	Medium: Our customers spend some time (e.g days) to reserve a property	Lack of information (e.g. potential cancellation) for some reservations since they could be for the future
Stay	Strong	Small	Large: The target is delayed since the time difference between a reservation and a stay could be long (e.g. months)	Lack of information about the experience of the customers during their stay
Stay with review score greater than threshold	Strong	Very Small	Large: Customers may write a review long after their stay	There is a bias on which customers choose to provide us with a review

Table 1: Positive signals

Negative Signal	Relation to user satisfaction	Amount of data points	Delay to observe signal	Bias
Impression of a property in search results and no click	Weak	Large	Small	Having a property displayed in the viewport with no user action does not necessarily translate to a negative user signal
Impression of a property in search results and no click, where the impression is above the property clicked	Medium	Large	Small	Assumption that the users examined all the properties above the one they clicked
Property click with dwell time less than a threshold	Medium	Large/Medium (depending on the threshold)	Small: Users perform multiple clicks on properties while they browse	Spending time browsing does not necessarily translate to customer dissatisfaction
Stay with review score less than a threshold	Strong	Small	Large: The customers can take a large period after their stay to write a review	There is a bias on which customers write reviews

Table 2: Negative signals

relation to user satisfaction, a good amount of observations and small observation delay. Utilizing property clicks or property clicks with at least a certain dwell time as secondary positive signals can

help the model learn that there could be multiple options that satisfy the requirements of a user to a certain extent, despite the user reserving only one specific accommodation.

Negative signals If a user clicked on at least one listing, we can assume that all search results above the clicked one were examined by the user and deliberately skipped. This creates a large and quickly observed set of data points indicating a weak fit for this user. If a user did not click any listing, their search result impressions are still useful as negative signals for the model to generalize and reduce the bias towards users who booked or clicked a property.

2.2 Feature Representation

A Machine Learned model that aims to serve as a ranker needs to learn information about both the preferences of our customers, and the diverse qualities of the properties providing accommodation. The representation of the feature spaces plays an important role for learning an optimal ranker. The major challenges around the features are:

Seasonality: The ranker needs to perform well across different seasons. Explicit categorical features about seasonality from the search context need to be considered (e.g. day of the week, month). Moreover, since there are seasonal effects on the amount of people travelling, the values of the property performance features e.g. amount of reservations over the last 6 months varies over time. We can represent these features as relative categorical variables, e.g. computing daily N percentiles of the reservations over the last 6 months and representing the feature by the corresponding percentile.

Non-linearity: It is important for a ranker to be able to learn non-linear relations between the target and the features, e.g. a reservation and the review score of a property. This can be addressed from the perspective of algorithm choice e.g. using Gradient Boosting. The same challenge can also be addressed from the standpoint of feature engineering, where we can represent features as categorical and learn non-linear relations even in linear algorithms.

Complexity: The feature space of accommodations is complex. Explicit human made features about a property do not necessarily describe this space in the best possible way. For example, star rating is a well-known characteristic of hotels, yet hotels are only a part of our inventory and the definition of stars does not exist for apartments. Projecting the properties in a latent feature space to enhance their feature representation can be useful. For instance, this can be achieved by training a Word2vec model over the sequence of user actions (impressions, clicks, bookings).

Locality: The ranker should be able to learn user preferences per destination. This can be achieved by either training one ranker per destination or cluster of destinations, or alternatively by introducing destination as a feature. However, we need to keep in mind that if the dimensionality of the destination feature is large, the "Curse of dimensionality" can be an issue for some algorithms [10]. Moreover, detailed information about the location of a property within a destination should be considered as well. Representing the location in neighborhoods could be a useful method, however the amount of neighborhoods around the world is very large, and the feature is sparse since a given neighborhood appears only in one specific destination. Transforming the feature to whether a property belongs to the top N most popular neighborhoods in a destination can alleviate these issues.

In-session adjustment: The ranker needs to be able to adjust while our customers search for their desired destination and accommodation. We can capture explicit behavioral characteristics such as clicks on certain components, and utilize them as features to infer a user's interest in specific properties. For example we can compute similarity scores of a ranked property with the latest properties clicked by the user in the current search and leverage those as a feature.

2.3 Ranking Biases

There are biases in the data that need to be addressed in order for a ranker to learn the user preferences objectively. Otherwise, the ranker may be biased towards certain types of properties which will result in a suboptimal experience for the customers. Three prominent biases that we aim to reduce are:

Impression bias: In our logs we capture our customers examining various properties. Our customers do not perform an exhaustive and equally distributed examination of all the available properties in a given destination. Therefore, the propensity of a property to be examined by a customer is influenced by the Machine Learned rankers historically used.

Position bias: Customers tend to click and reserve properties that are placed higher in the search results since there is an inherent bias that items closer to the top are "better".

User bias: Different users display different behavior in terms of the amount of property impressions, clicks and bookings. This leads to bias towards some users that create more training data points, while the ranker should be performing equally well for all users.

There have been multiple publications on the topic of bias reduction over the years that all revolve around various methods to perform propensity estimation and utilizing it in the learning of a ranker with reduced biases, such as treating the biases as counterfactual and weighting the actions by the Inverse Propensity Weighting [17], query level Inverse Propensity Weighting [20] or representing the position trust bias as position dependent noise [2].

2.4 Model Creation

The ranker needs to learn the user preferences over thousands of destinations and millions of diverse accommodations.

Large-scale training: Training a model on billions of rows and terabytes of data poses some challenges. Online Learning [6] techniques can help us learn models on such sizes of training data by enabling us to use out-of-core incremental learning [19] and benefit from a reduced memory footprint. Utilizing the Hashing Trick [21] also enables us to have variable size feature vectors, and limit the memory footprint. If the dataset is larger than the disk or the training time is longer than the expected time to release a re-trained ranker, distributed Machine Learning can be utilized to speed up the training process. There are intricacies that come with distributed Machine Learning e.g. node synchronization, load balancing, fault tolerance or parameter averaging. Distributed Machine Learning is an area which is actively developing, yet there

seems to be no consensus about which methods are the best to use [1, 8, 14, 18].

Generalization: It is really important to evaluate the ability of the ranker to generalize. Since the goal is to train a ranker using historical observations and produce the optimal ranking for our customers in the future, out-of-time validation should be used. One of the methods to achieve this is walk-forward validation, which allows us to evaluate a ranker across different time periods [13]. Furthermore it is essential to notice that during training we are utilizing the properties displayed to the users to learn the optimal ranking, yet during testing we need to include all the properties that were available at every point in time in order to simulate what would happen in reality on inference time and calculate offline evaluation metrics such as precision, recall, mrr, ndcg that can provide us with a more accurate health check of a ranker.

Multi-scenario evaluation: Evaluating the ranker in different scenarios is important. Because we are utilizing behavioral features and property clicks to learn the optimal ranking, we should evaluate a ranker not only through global ranking metrics but also pay attention to customer subgroups. The ranker could be performing very well for customers that have already interacted with our platform and inadequately for customers that just landed.

3 EXPERIMENTATION

Online experimentation in the form of Randomized Controlled Trials (RCTs) has long played a central role in the development of products at Booking.com [12, 15]. It enables us to measure the business value of a new product, feature or model. Ranking is a self-learning system where a model that is used to sort the items also influences the data that it is trained on. This poses a data leakage challenge in our experimentation that needs to be carefully handled. Moreover, considering that we know that offline evaluation is only a health check and does not necessarily correlate with business value [4], evaluation of multiple rankers relies on efficient online experimentation, and poses speed and sensitivity challenges which if successfully addressed can accelerate product development.

3.1 Leakage

In a RCT with our users as the experimental units, we randomly split the audience in Control and Treatment Group which are exposed to two different rankers. There are two data leakage challenges in such a setup: re-training leakage and feature leakage. They can both compromise the observations of our trial if not addressed.

Re-training leakage: If the rankers are re-trained during the trial and the training data is derived from all the logged data of our platform regardless of the group it is generated from, there will be a "leakage" of customer preferences between the two rankers, and this might lead the two rankers to recommend similar properties with similar sorting. Therefore, Isolated Feedback Loops of the data logging of all the groups in a RCT are necessary.

Feature leakage: Even if the rankers are not re-trained during trial, in case they rely on time-dynamic features such as property performance in the last day e.g. amount of reservations, there will be "leakage" of the effect of one ranker on the other. In this case, there are two options: 1) "Freezing" of the data sources to contain only pre-experimental data points. This solution requires minimal infrastructure investment from a data logging standpoint. Yet it assumes that the world is stationary which is not acceptable when a ranker relies on features that reflect very recent performance such as amount of reservations over the last day, and 2) Isolated Feedback Loops per group of the RCT which provide us with a clear separation of the data, yet create complexity on how we utilize the data collected.

3.2 Speed of Innovation - Experimentation via Interleaving

Another challenge regarding online testing in Ranking is the speed of experimentation. When experimenting with Ranking algorithms we cannot just run multiple tests at the same time as the independence assumption - that simultaneously running experiments do not impact the outcome of each other - does not hold true.

We have tried various setups in the past such as executing multivariate experiments, splitting traffic into equal random subgroups and running an experiment per subgroup or leveraging indirect metrics in RCTs as an early indicator of customer satisfaction. However, none of these are desirable as we would either have to accept a considerable loss in power or deal with increased complexity. Based on reports from other companies [5] on promising results, we investigated how interleaving experimentation can be used in Search Ranking at Booking.com.

With interleaving (IL) we do not randomise on the customer, instead we construct our search results by merging the results from two rankers applying team draft IL [16] and use each position in the search ranking as a unit of randomisation. We then measure if our users interact more with search results coming from ranking A or ranking B [7].

Figure 1 visualises the relation between the results of the RCT experiment and the corresponding pairwise IL trials. The RCT compares control and treatment based on an important business metric and the related confidence intervals. In the IL trials we examine user preference based on comparison of relative wins (i.e. which algorithm the user interaction is attributed to) of one ranker to the other and we use bootstrapping to compute a corresponding distribution [11]

The results of our IL trial have been very promising: IL tests were very sensitive to differences in user preferences - in general we saw potential for a 10-100x speedup due to the high sensitivity of results. Thus we can run IL on less traffic and test more variants simultaneously leading to substantial decrease in runtime requirements. Based on statistical characteristics only we could reduce our experiment runtime even further to just a few hours. However, as we want to observe a global representation of our traffic and not make decisions based on preferences of users in only one timezone we run online tests for at least a day to obtain sufficient exposure.

It is important to note the limitations of interleaving: the detection of a conclusive ranker preference regarding a metric in IL does not necessarily translate into a significant change of the metric in the RCT experiment given our usual expected effect and also cannot be used to estimate the effect size of an RCT test. Hence we can only use IL to make a preselection between algorithms and the maximal risk is that we do not choose the most promising model,

but rather a suboptimal one and test that one in the following RCT test.

Why do we see diverging results and why is interleaving so much more sensitive? If we segment our customers on search results by strength of intent, we will always have a large share of low-intent users, who most probably will not book in this session (think of early stage exploration), some smaller share of high-intent users, who will book a room no matter what (think of a business traveller who definitely needs an accommodation in the next few days) and a very small part of customers, who are ready to book but only if they are convinced that they found the right offer. The low-intent users are inevitable noise in all cases, they produce searches and page views, but no bookings. In an RCT experiment, the high-intent users will also create noise, because they are highly likely to book even if we show accommodations in a suboptimal order, effectively providing no evidence if the ranking is good or bad. We are only getting relevant signals from the extremely sparse group of users whose decision still has the freedom to go one or the other direction. Ranking interleaving provides an opportunity to also gather signals about the quality of a ranker from the high-intent users as they will still pick the property that satisfies their requirements best. This can substantially increase the number of meaningful votes providing feedback on which ranking algorithm is better.

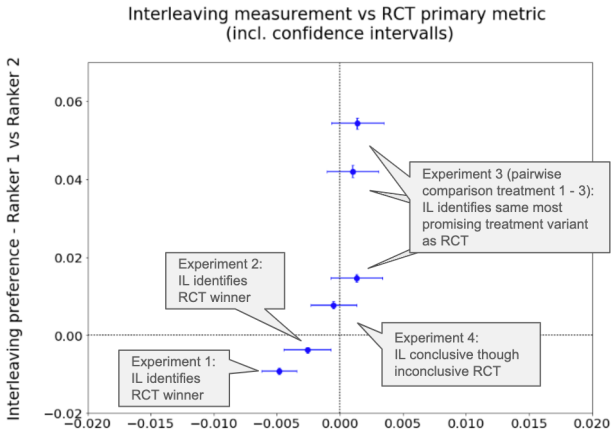

Figure 1: Result of Interleaving (IL) trial

4 SERVING

A Machine Learned ranker needs to serve millions of users looking for accommodations every day. Each customer may perform several searches to choose the best accommodation over a set of tens of thousands of properties in some cases. We support searches up to a year from the search date, making the problem of fetching the availability in real time for all properties, check-in, check-out, policy, room combinations a challenge on its own. Response time minimization [4] and efficient feature retrieval are critical for the application of real time Machine Learning based ranking.

Response Time Minimization: A way to tackle this problem is to parallelize the computation of the availability and ranking score of each property, thus building a top N list (i.e. the first page) to be returned in a distributed fashion by using sharding [9], where each shard has a subset of properties to be ranked.

Let N be the number of properties to be returned to the search results, each shard utilizes a reverse index that maps destinations to properties, selects the properties that belong to the destination, computes the availability and the ranking score on its subset of properties and returns its (partial) top N to the coordinator. The coordinator aggregates the partial top N lists from each shard into the final top N which is presented in the search results. Since each shard is responsible for a subset of properties, it becomes feasible to pre-compute and materialize the availability information for all the possible combinations of check-in/out date, room and policy. This mechanism is very convenient for the deployment of a Machine Learned model, although it comes with the limitation that each shard does not have complete visibility to which properties are available globally.

Feature Deployment: Feature deployment for models which are called in real time poses another challenge: all features a model was trained with must be available at prediction time. Our model relies on customer, search and property features. Customer features can be thought of as two types: historical, such as past bookings, and in-session, such as the latest property clicked by the user in the current search. A solution to the productionization of the historical features is to have a key-value storage and retrieve them as soon as the customer comes to the website. In-session and search features can be kept in memory with some expiration timeout.

Property features, on the other hand, are either static features such as location-related features or performance features which are updated in regular intervals. Passing the features on the request is inefficient since the features do not depend on the user context or actions. They can be productionized via in-memory lookup tables.

Another challenge to the feature productionization comes from the latency and memory footprint of the feature weights look-up. One solution is to have a key-value storage where the key is the feature and the value is its associated weight. However, the feature space can be large and the memory footprint of this key value data structure can be problematic. Utilizing the hashing trick also in production alleviates this problem.

Hashing is fast (scale of nanoseconds) and it adds a minimal performance effect on the look-up process, and using a hash of a pre-specified length provides us with the benefit of limited memory footprint of the feature space in production, while having negligible impact on model performance if the hash space size is chosen correctly [3, 21].

5 QUANTIFYING BUSINESS IMPACT

This section presents the relative improvement in business value that the introduction of each step change in the Machine Learning ranking yielded, which are summarized in Figure 2. Each bar represents the improvement measured in RCTs against the control group. The first bar represents the improvement that the initial Machine Learned model brought. We take this improvement as a reference value (i.e. it has value of 1) in order to indicate the scale of the improvements of the following step changes we introduced in ranking.

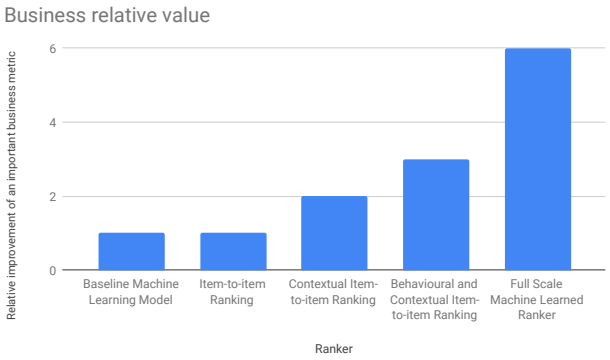

Figure 2: Relative improvement for each step change

The next step change was the utilization of item-to-item similarity derived from historical co-occurrence data to adjust the search results of every user within their session based on their property pageviews and was working in conjunction with the Machine Learned model to provide the users with an improved user experience while they were browsing our platform. The introduction of the user context in the item-to-item similarity ranking helped us take a step forward and provide in-session personalisation for different traveller profiles. Following that, utilizing the user behavioral features (e.g. reading reviews of a property or time since last click on a property) helped us understand the user preferences even better while our customers browse our platform.

As a result, over the years, the ranking of the search results was an ensemble consisting of various Machine Learning components that were incrementally added through RCTs. These components had been continuously optimized to improve customer experience until we experienced a plateau in the business gains. It was clear that the next step change was the incorporation of all the learnings from the introduction of these components to a single full-scale Machine Learned ranker. The last bar represents the improvement in business value brought by the introduction of the first full-scale Machine Learned ranker which replaced all the various components that were working in conjunction before, and compared to the previous step changes, the gains were substantial.

6 CONCLUSION

In this work we shared the challenges and viable solutions that we have faced while developing Machine Learned rankers in a large-scale e-commerce setting. We covered three aspects beyond the choice of algorithm: Modelling, Experimentation and Serving. Several challenges arise in modelling, such as defining which signals to utilize based on the various observed user actions, how to represent features and treat biases, how to train Machine Learned rankers based on large datasets and how to evaluate such rankers. Validating a ranker through RCTs poses the problem of information leakage among control and treatment, which requires special care. We also display the power of interleaving in reducing noise and increasing power in the ranker online evaluation, albeit its limitation in providing quantitative information regarding the business metrics of interest. Serving a Machine Learned model on a high-traffic platform poses challenges in terms of latency and model deployment, which often require carefully crafted engineering solutions. Lastly, we showed the business value of our step changes in ranking over time. Our main conclusion is that there are multiple important aspects beyond the choice of algorithm that need to be carefully addressed for the introduction of a Machine Learned ranker in a commercial setup. We wish this work would inspire practitioners around the application of Machine Learning for ranking problems and serve as guidance on their path addressing the challenges we faced.

ACKNOWLEDGMENTS

We thank all the colleagues who have worked hard on improving the ranking at Booking.com over the years.

REFERENCES

[1] Alekh Agarwal, Oliveier Chapelle, Miroslav Dudík, and John Langford. 2014. A Reliable Effective Terascale Linear Learning System. *Journal of Machine Learning Research* 15, 32 (2014), 1111–1133.
[2] Aman Agarwal, Xuanhui Wang, Cheng Li, Michael Bendersky, and Marc Najork. 2019. Addressing Trust Bias for Unbiased Learning-to-Rank. In *The World Wide Web Conference*. 4–14.
[3] Lucas Bernardi. 2018. Don't be tricked by the Hashing Trick. https://booking.ai/dont-be-tricked-by-the-hashing-trick-192a6aae3087. Accessed August 28, 2020.
[4] 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.
[5] Netflix Technology Blog. 2017. Innovating Faster on Personalization Algorithms at Netflix Using Interleaving. https://medium.com/netflix-techblog/interleaving-in-online-experiments-at-netflix-a04ee392ec55. Accessed May 25, 2020.
[6] Léon Bottou. 1998. Online Algorithms and Stochastic Approximations. In *Online Learning and Neural Networks*, David Saad (Ed.). Cambridge University Press, Cambridge, UK. revised, oct 2012.
[7] Olivier Chapelle, Thorsten Joachims, Filip Radlinski, and Yisong Yue. 2012. Large-scale validation and analysis of interleaved search evaluation. *ACM Transactions on Information Systems (TOIS)* 30, 1 (2012), 1–41.
[8] Adam Coates, Brody Huval, Tao Wang, David Wu, Bryan Catanzaro, and Ng Andrew. 2013. Deep learning with COTS HPC systems. In *International conference on machine learning*. 1337–1345.
[9] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. 2013. Spanner: Google's Globally Distributed Database. *ACM Trans. Comput. Syst.* 31, 3, Article 8 (Aug. 2013), 22 pages.
[10] David L Donoho. 2000. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS math challenges lecture* 1 (2000), 32.
[11] Roger W Johnson. 2001. An introduction to the bootstrap. *Teaching Statistics* 23, 2 (2001), 49–54.
[12] Raphael Lopez Kaufman, Jegar Pitchforth, and Lukas Vermeer. 2017. Democratizing online controlled experiments at Booking.com. *arXiv preprint arXiv:1710.08217* (2017).
[13] Charles D Kirkpatrick II and Julie A Dahlquist. 2010. *Technical analysis: the complete resource for financial market technicians*. FT press.
[14] Mu Li, David G Andersen, Jun Woo Park, Alexander J Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J Shekita, and Bor-Yiing Su. 2014. Scaling distributed machine learning with the parameter server. In *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*. 583–598.
[15] Bahattin Tolga Öztan, Zoé van Havre, Caio Gomes, and Lukas Vermeer. 2018. Mediation Analysis in Online Experiments at Booking.com: Disentangling Direct and Indirect Effects. *arXiv preprint arXiv:1810.12718* (2018).
[16] Filip Radlinski, Madhu Kurup, and Thorsten Joachims. 2008. How does clickthrough data reflect retrieval quality?. In *Proceedings of the 17th ACM conference on Information and knowledge management*. 43–52.
[17] Paul R Rosenbaum and Donald B Rubin. 1983. The central role of the propensity score in observational studies for causal effects. *Biometrika* 70, 1 (1983), 41–55.

[18] Alexander Smola and Shravan Narayanamurthy. 2010. An Architecture for Parallel Topic Models. *Proc. VLDB Endow.* 3, 1–2 (2010), 703–710.
[19] Jeffrey Scott Vitter. 2001. External memory algorithms and data structures: Dealing with massive data. *ACM Computing surveys (CsUR)* 33, 2 (2001), 209–271.
[20] Xuanhui Wang, Michael Bendersky, Donald Metzler, and Marc Najork. 2016. Learning to rank with selection bias in personal search. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. 115–124.
[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.