Predicting Potential Customer Support Needs and Optimizing Search Ranking in a Two-Sided Marketplace

Do-kyum Kim
Airbnb
CA, USA
do-kyum.kim@airbnb.com

Han Zhao
Airbnb
CA, USA
han.zhao@airbnb.com

Huiji Gao
Airbnb
CA, USA
huiji.gao@airbnb.com

Liwei He
Airbnb
CA, USA
liwei.he@airbnb.com

Malay Haldar
Airbnb
CA, USA
malay.haldar@airbnb.com

Sanjeev Katariya
Airbnb
CA, USA
sanjeev.katariya@airbnb.com

ABSTRACT

Airbnb is an online marketplace that connects hosts and guests to unique stays and experiences. When guests stay at homes booked on Airbnb, there are a small fraction of stays that lead to support needed from Airbnb's Customer Support (CS), which may cause inconvenience to guests and hosts and require Airbnb resources to resolve. In this work, we show that instances where CS support is needed may be predicted based on hosts and guests behavior. We build a model to predict the likelihood of CS support needs for each match of guest and host. The model score is incorporated into Airbnb's search ranking algorithm as one of the many factors. The change promotes more reliable matches in search results and significantly reduces bookings that require CS support.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; • **Information systems** → **Online shopping**; • **Applied computing** → **Online shopping**.

KEYWORDS

Search Ranking, Customer Support, Two-sided Marketplace, AUC Maximization, Neural Networks

1 INTRODUCTION

As of December 31, 2023, Airbnb had more than 7.7 million active listings from more than 5 million hosts worldwide [2]. While the platform invests heavily on its growth, it also strives to provide pleasant trip experience to both guests and hosts. In 2022, Airbnb launched AirCover that provides comprehensive protection for guests and hosts. For example, if a host cancels a reservation within 30 days of check-in, Airbnb provides support for finding a similar place, depending on availability at comparable pricing [1].

While Airbnb provides Customer Support (CS) and Aircover, ideally, the need for CS support is minimized in the first place. That led to the key question of this paper: whether CS support needs may be predicted. If they are, some of them might be preventable before they happen and there is less need to contact CS. When we started this work, we had some evidence that they are. It is known that matching a new guest with a new host is more likely to require CS support as both parties are unfamiliar with how Airbnb works. Another example is same day bookings where more responsive hosts may result in less CS support needed.

Based on the evidence, we set out to build predictive models on whether a booking may result in CS support needs. From offline evaluation, we confirmed that our models are able to predict it to some extent. By incorporating the model in search ranking as one of the many considerations, we promote more reliable matching between guests and hosts, thereby preventing CS support needs before they happen.

2 RELATED WORK

At Airbnb, search is the major interface where a guest is matched with a home given specific query parameters including location, dates and number of guests. During the match, a guest goes over a ranked list of homes and determines which one to book. Because of that, search ranking is one of the major levers for Airbnb to optimize business metrics. There are many different business metrics that can be optimized in search and recommendation systems. Ranking typically focuses on optimizing conversions. In Airbnb, we have been optimizing search ranking, either by directly predicting an uncancelled booking [9], or by adopting multi-task models to optimize multiple events through the guests' search journey [13].

Besides conversions, user satisfaction is another important business metric that is often optimized on search products for two sided marketplaces. User satisfaction is sometimes indirectly measured based on user engagement signals such as time between visits or retention rates. Alternatively, surveys are used to directly ask customers to rate their satisfaction. Search engines [7][8] make continuous updates to their ranking algorithms to reduce the amount of low quality contents and fake news in their search results. Social media platforms [11] combine multiple user actions (e.g. clicks and likes) in their ranking algorithms, and they additionally apply integrity-related scores in the final stage to remove harmful contents. Video recommendation systems [17][14] uses multi-task learning to optimize both user engagement and user satisfaction jointly to improve recommendation quality. E-commerce site [10] uses multi-objective optimization algorithms to balance between objectives such as product quality and purchase likelihood.

3 APPROACH

We start by formulating a binary classification problem. To train a classifier, we construct a data set from past bookings on our

platform. Lastly, the predicted likelihood from the classifier is incorporated into the ranking function in Airbnb homes search to help prevent CS support required.

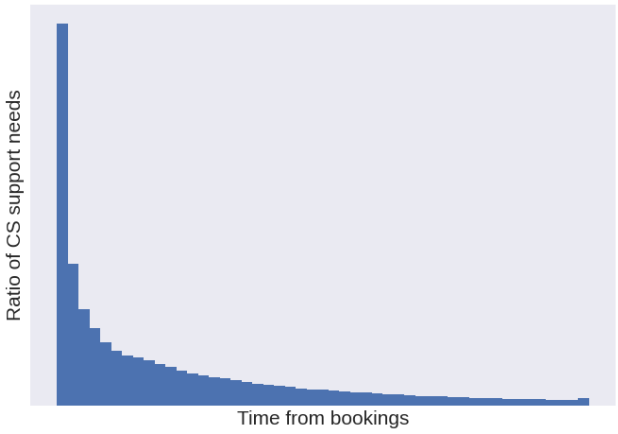

Figure 1: Time from bookings to CS support needs

3.1 Binary Classification

We formulate a binary classification problem for predicting

$$p(\text{CS support needs}|\text{booking})$$

. In words, our training examples are all the historical bookings, and positive examples are bookings that had CS support needs. Our models need to predict future outcomes that are significantly delayed; while CS support might be required anytime between booking and check outs (or even after that) — see Figure 1 for a distribution —, we are trying to predict the probability at the time of bookings.

Thus, in constructing our data set, one of the key considerations is determining an attribution window, i.e. number of days to wait from a booking to determine a label. To increase coverage, we need to use a longer attribution window. On the other hand, using a longer attribution window prevents us from using the most recent data and it also has implications on running online A/B experiments. When we run an online experiment with a new search ranking algorithm, the experiment (and the new algorithm) is applied for a fixed time period but its outcome — whether a booking made during the experiment led to CS support needs — should be tracked for an extended period of time (at least longer than the attribution window).

In terms of input features, we consider all information available in our search ranking system since the model will be scored inside the system. When a guest searches for homes on Airbnb, they typically set filters such as destination, check in/out dates and number of guests. Given the search filters, we retrieve all the available homes that match the filters. For each available home, we will estimate the likelihood of CS support needs if the home is booked by the searcher with the search filters. Thus, we formulate features about the searcher, home and its hosts. We also found that search filters play important roles in predicting CS support needs. For example, same day bookings tend to require more CS support than other types of bookings.

3.2 Maximizing Area under the ROC Curve

The area under an ROC curve (AUC) is an evaluation metric used in many classification applications [5]. One interpretation of AUC is that, given pairs of positive and negative examples, it computes a ratio of pairs where a positive example is assigned a higher score by a model than a negative example is (see Lemma 1 in Cortes et al. [5]). Because of that, AUC is often used as a metric for ranking applications and we also used it as our main evaluation metric of models.

During model training, there are many approaches for directly maximizing AUC [16]. One simple method is, for a loss function, using an approximation of AUC based on sigmoid functions [4]. Specifically, an i-th example x_i gets assigned a logit $f(x_i)$, where f is a model. Next, between positive and negative examples, pairwise logit differences are computed and the differences are fed into sigmoids to form a loss function. Essentially, the loss function we optimize is equivalent to a cross entropy loss with only positive examples where each positive example is a pair of original positive and negative examples. Formally, it is computed as:

$$-\sum_{x_i \in \text{positives}} \sum_{x_j \in \text{negatives}} \log \sigma(f(x_i) - f(x_j))$$

, where σ is a sigmoid function.

3.3 Neural Networks

There are different classes of models that can be used for classification and we choose to use neural networks [6] for our application. One advantage of neural networks is that they can learn representation of categorical features during the course of training, which reduces efforts on manual feature engineering. It also guarantees parity with the current architecture and infrastructure as our main ranking models are also neural networks [13].

To use neural networks, we normalized all continuous features so that each feature has zero mean and unit variance. For features with skewed distribution (e.g. number of past bookings), we also applied a logarithmic transformation before normalization. Each categorical feature is mapped to an embedding vector that is learned during the optimization process.

3.4 Ranking Function

Our current ranking function is a linear combination of predicted probabilities for many different events [3] including clicks on search results, sending booking requests and cancellations [13]. In the existing linear combination, we add one more term based on likelihood of CS support needs given a booking.

4 EXPERIMENTS

We optimized neural networks architectures and hyperparameters based on AUC metrics on a hold-out set. Another consideration was minimizing latency as these models will be computed for each search request at serving time. Finally, we choose one model and then run online A/B experiments to measure impacts on business metrics such as number of bookings and CS contacts.

4.1 Data Set

To train models, we collected historical bookings on Airbnb. Among the bookings that are past their attribution windows — thus they have fixed labels —, bookings made during the last three weeks were held out for evaluation, while those made in the year leading up to these three weeks were used for training.

4.2 Hyperparameter Search

We trained feed-forward neural networks with different depths and widths. For a loss function, either a cross entropy loss or the aforementioned approximation of AUC was used. We also varied other hyperparameters such as optimizers, regularizers and activation functions. Lastly, a learning rate scheduler was used to halve a learning rate when a plateau is reached on a validation set.

We list a few combinations and their AUC metrics in Table 1. Since AUC is a probability of correct ordering on pairs of positive and negative examples [5], a random guesser would achieve an AUC of 0.5. Our models achieved AUC over 0.73, meaning CS support needs are predictable to some extent at the time of bookings. As expected, directly maximizing AUC outperformed minimizing cross entropy loss. Since positive examples — bookings with CS support needs — are rare, we used a large batch size. Increasing the number of hidden layers enhances the validation AUC, although with diminishing returns.

As our models need to serve online traffic, latency should be minimized. We assume that latency from model inference would be roughly proportional to the number of multiplications in a model since we are using CPUs to serve these models. In Table 1, we computed the number of multiplications in each architecture by using input dimension of 209 and number of parameters in the feed-forward neural networks. To balance between validation AUC and latency, we decided to use the model from the first row in Table 1.

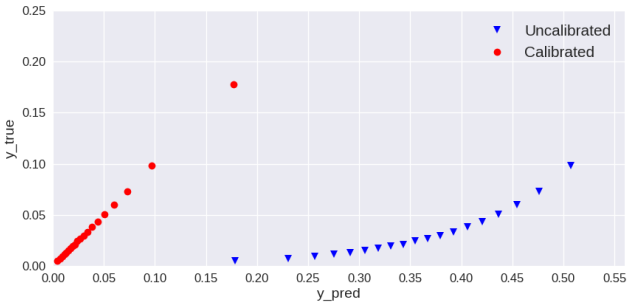

Figure 2: Calibration of model score using a Platt scaler. The line for calibrated score (red circle) is close to a diagonal line, which means the scaler provides well-calibrated probabilities.

4.3 Offline Tuning of Ranking Function

Since we train our models to maximize AUC, a resulting model does not provide calibrated probabilities. A Platt scaler [12] is used to calibrate model scores into conditional probabilities. The procedure

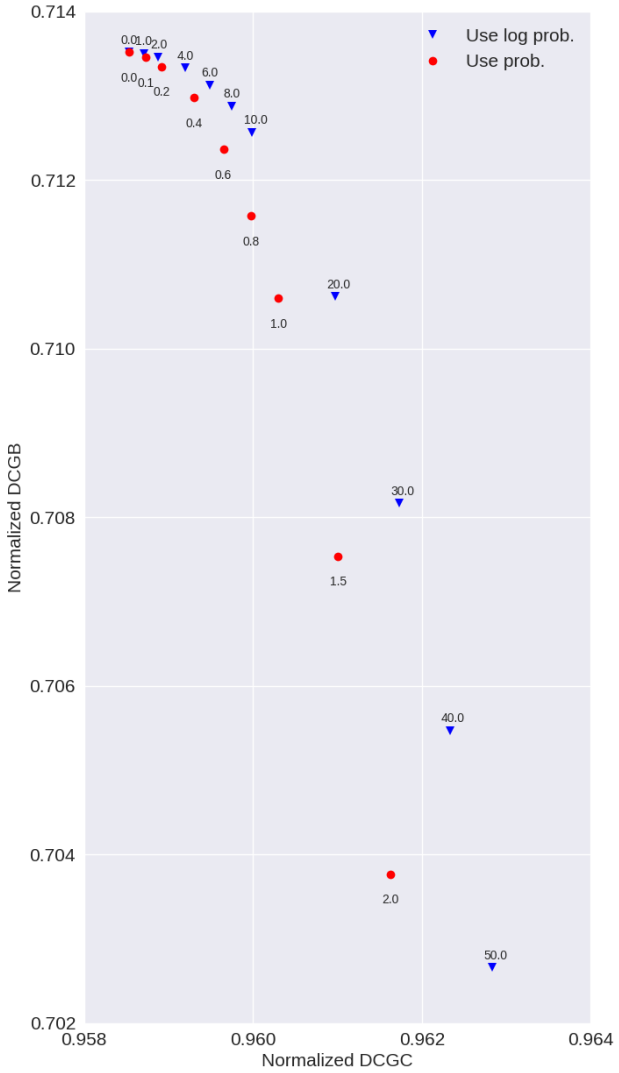

Figure 3: Trade off between normalized DCGB and normalized DCGC as a multiplier α is varied. We need to increase both of the metrics. For the details of metrics, please refer to section 4.3.

is akin to training a logistic regressor with a single feature where the single feature is a logit from the underlying model $f(x)$ as:

$$p(\text{CS support needs}|\text{booking}) = \frac{1}{1+\exp(-wf(x)-b)}$$

. Figure 2 demonstrates the effectiveness of a Platt scaler for our application.

As mentioned in the previous section, our current ranking function is a linear combination of multiple conditional probabilities of different events in the conversion funnel. In the linear combination, we add one more term as:

$$(\text{current ranking function}) + \alpha \log\left(1 - p(\text{CS support needs}|\text{booking})\right)$$

Table 1: A few representative combinations of loss function, architectures and hyperparameters. '64-32' means that there are two hidden layers with 64 and 32 units respectively. The model from the first row is used in our online experiment.

Loss	Architecture	Hyperparameters	# Multiplications	Validation AUC	Train AUC
AUC	128-64-32 with Relu	batch size=5000, adam optimizer, l2 regularization=0.0005	36992	0.7321	0.7426
AUC	64-32 with Relu	same as the 1st row	15424	0.7314	0.7396
AUC	512-256-64-32 with Relu	same as the 1st row	256512	0.7326	0.7433
Cross entropy	same as the 1st row	same as the 1st row	36992	0.7225	0.7317
AUC	128-64-32 with Leaky Relu	same as the 1st row	36992	0.7314	0.7421

The multiplier α is determined by balancing two offline metrics on past search sessions. Each session consists of a query, a set of homes appeared in search results, and a label on which home is booked. In each session, a specific ranking function provides an ordering of the homes. For each ordering, we can compute a Discounted Cumulative Gain (DCG) [15] with a booking label as:

$$\text{DCGB} = \sum_{i=0}^{N} \frac{r_i}{\log(2.4+i)}$$

, where N is a number of homes and r_i is 1 if the i-th home is booked. Similarly, we compute a DCG with conditional probabilities of a booking **without** CS support needs for each home as:

$$\text{DCGC} = \sum_{i=0}^{N} \frac{1 - p(\text{CS support needs}|i\text{-th home is booked})}{\log(2.4+i)}$$

, where the conditional probabilities in numerator is from our model and Platt scaler.

On a data set of past search sessions from 7 days, we vary α and compute the two metrics DCGB and DCGC. The two metrics are normalized for each session so that they range between 0 and 1. The normalized metrics from a parameter sweep is presented in Figure 3. As we increase α, normalized DCGB declines, which means booked homes are ranked lower; but normalized DCGC improves, meaning that more reliable matchings are promoted in search results. It is also clear that using log probability provides a better trade-off than using the probability. We chose a few values for α to run online A/B experiments.

4.4 Online A/B Experiments

During our online experiments, we split searchers into multiple cohorts, and each cohort used a specific value of α, which controls the strength of our model in the ranking function. For each cohort, we measured business metrics including number of bookings and also number of bookings with CS support needs. As expected, higher α reduced both number of bookings and number of bookings with CS support needs. We were able to find a right value of α that significantly reduces bookings with CS support needs while not negatively impacting overall booking conversions. Specifically, with the launched value of α, we reduced bookings with CS support needs by 3.7% and also lowered host cancellations by 2.7% — host cancellations often lead to CS contacts. These metrics align with our expectation that a better matching between guests and hosts can reduce CS support needs and benefit both sides in the marketplace.

5 DISCUSSION

We started from an assumption that some CS support needs might be predicted at the time of bookings. The assumption led us to build a binary classifier to predict if a booking — a match between a guest and a home (and the host) — would lead to CS support needs or not. From our offline analysis, we confirmed that CS support needs are predictable to some extent at the time of bookings.

By incorporating our prediction model in search ranking, we achieved a significant reduction in CS support needs without reducing overall conversions. Furthermore, we believe that reducing CS support needs can improve satisfaction of our guests and hosts, helping our guests to book more on Airbnb and also assisting our hosts in growing their business.

For future work, we can extend this approach to other types of user feedback to further improve trip experiences. To enhance model performance, we could experiment with more advanced model architectures and different loss functions. Another direction is exploring more ways to combine multiple model scores in a ranking function to better balance different business goals.

ACKNOWLEDGMENTS

We thank our collaborators from the Relevance, Core Data Science and Community Support teams. Especially, support from Anna Matlin and Alex Deng was crucial in finding the right value of parameters. We also thank Airbnb's internal reviewers and anonymous reviewers for helpful comments.

REFERENCES

[1] Airbnb. 2022. AirCover for guests. https://www.airbnb.com/help/article/3218
[2] Airbnb. 2023. Airbnb Q4-2023 and full-year financial results. https://news.airbnb.com/airbnb-q4-2023-and-full-year-financial-results/
[3] Airbnb. 2024. How search results work. https://www.airbnb.com/help/article/39
[4] Toon Calders and Szymon Jaroszewicz. 2007. Efficient AUC Optimization for Classification. In *Knowledge Discovery in Databases: PKDD 2007*, Joost N. Kok, Jacek Koronacki, Ramon Lopez de Mantaras, Stan Matwin, Dunja Mladenič, and Andrzej Skowron (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 42–53.
[5] Corinna Cortes and Mehryar Mohri. 2003. AUC Optimization vs. Error Rate Minimization. In *Advances in Neural Information Processing Systems*, S. Thrun, L. Saul, and B. Schölkopf (Eds.), Vol. 16. MIT Press. https://proceedings.neurips.cc/paper_files/paper/2003/file/6ef80bb237adf4b6f77d0700e1255907-Paper.pdf
[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press. http://www.deeplearningbook.org.
[7] Google. 2017. Our latest quality improvements for Search. https://blog.google/products/search/our-latest-quality-improvements-search/
[8] Google. 2024. New ways we're tackling spammy, low-quality content on Search. https://blog.google/products/search/google-search-update-march-2024/
[9] Malay Haldar, Mustafa Abdool, Prashant Ramanathan, Tao Xu, Shulin Yang, Huizhong Duan, Qing Zhang, Nick Barrow-Williams, Bradley C. Turnbull, Brendan M. Collins, and Thomas Legrand. 2019. Applying Deep Learning

to Airbnb Search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery amp; Data Mining (KDD '19)*. ACM. https://doi.org/10.1145/3292500.3330658

[10] Debabrata Mahapatra, Chaosheng Dong, Yetian Chen, and Michinari Momma. 2023. Multi-label learning to rank through multi-objective optimization. In *KDD 2023*. https://www.amazon.science/publications/multi-label-learning-to-rank-through-multi-objective-optimization

[11] Meta. 2023. Scaling the Instagram Explore recommendations system. https://engineering.fb.com/2023/08/09/ml-applications/scaling-instagram-explore-recommendations-system/

[12] John Platt. 1999. Probabilistic Outputs for Support Vector Machines and Comparisons to Regularized Likelihood Methods. *Advances in Large Margin Classifiers* (1999).

[13] Chun How Tan, Austin Chan, Malay Haldar, Jie Tang, Xin Liu, Mustafa Abdool, Huiji Gao, Liwei He, and Sanjeev Katariya. 2023. Optimizing Airbnb Search Journey with Multi-task Learning. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Long Beach, CA, USA) *(KDD '23)*. Association for Computing Machinery, New York, NY, USA, 4872–4881. https://doi.org/10.1145/3580305.3599881

[14] Hongyan Tang, Junning Liu, Ming Zhao, and Xudong Gong. 2020. Progressive Layered Extraction (PLE): A Novel Multi-Task Learning (MTL) Model for Personalized Recommendations. In *Proceedings of the 14th ACM Conference on Recommender Systems* (Virtual Event, Brazil) *(RecSys '20)*. Association for Computing Machinery, New York, NY, USA, 269–278. https://doi.org/10.1145/3383313.3412236

[15] Yining Wang, Liwei Wang, Yuanzhi Li, Di He, and Tie-Yan Liu. 2013. A Theoretical Analysis of NDCG Type Ranking Measures. In *COLT 2013 - The 26th Annual Conference on Learning Theory, June 12-14, 2013, Princeton University, NJ, USA (JMLR Workshop and Conference Proceedings, Vol. 30)*, Shai Shalev-Shwartz and Ingo Steinwart (Eds.). JMLR.org, 25–54. http://proceedings.mlr.press/v30/Wang13.html

[16] Tianbao Yang and Yiming Ying. 2022. AUC Maximization in the Era of Big Data and AI: A Survey. *ACM Comput. Surv.* 55, 8, Article 172 (dec 2022), 37 pages.

[17] Zhe Zhao, Lichan Hong, Li Wei, Jilin Chen, Aniruddh Nath, Shawn Andrews, Aditee Kumthekar, Maheswaran Sathiamoorthy, Xinyang Yi, and Ed Chi. 2019. Recommending what video to watch next: a multitask ranking system. In *Proceedings of the 13th ACM Conference on Recommender Systems* (Copenhagen, Denmark) *(RecSys '19)*. Association for Computing Machinery, New York, NY, USA, 43–51. https://doi.org/10.1145/3298689.3346997