Management Science

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

Search Personalization Using Machine Learning

Hema Yoganarasimhan

To cite this article:
Hema Yoganarasimhan (2020) Search Personalization Using Machine Learning. Management Science 66(3):1045-1070. https://doi.org/10.1287/mnsc.2018.3255

http://pubsonline.informs.org/journal/mnsc

MANAGEMENT SCIENCE
Vol. 66, No. 3, March 2020, pp. 1045–1070
ISSN 0025-1909 (print), ISSN 1526-5501 (online)

Search Personalization Using Machine Learning

Hema Yoganarasimhan[a]

[a] Foster School of Business, University of Washington, Seattle, Washington 98195
Contact: hemay@uw.edu, http://orcid.org/0000-0003-0703-5196 (HY)

Received: January 16, 2016
Revised: July 27, 2018; May 24, 2018
Accepted: August 8, 2018
Published Online in Articles in Advance: August 29, 2019

https://doi.org/10.1287/mnsc.2018.3255

Abstract. Firms typically use query-based search to help consumers find information/products on their websites. We consider the problem of optimally ranking a set of results shown in response to a query. We propose a personalized ranking mechanism based on a user's search and click history. Our machine-learning framework consists of three modules: (a) feature generation, (b) normalized discounted cumulative gain–based LambdaMART algorithm, and (c) feature selection wrapper. We deploy our framework on large-scale data from a leading search engine using Amazon EC2 servers and present results from a series of counterfactual analyses. We find that personalization improves clicks to the top position by 3.5% and reduces the average error in rank of a click by 9.43% over the baseline. Personalization based on short-term history or within-session behavior is shown to be less valuable than long-term or across-session personalization. We find that there is significant heterogeneity in returns to personalization as a function of user history and query type. The quality of personalized results increases monotonically with the length of a user's history. Queries can be classified based on user intent as transactional, informational, or navigational, and the former two benefit more from personalization. We also find that returns to personalization are negatively correlated with a query's past average performance. Finally, we demonstrate the scalability of our framework and derive the set of optimal features that maximizes accuracy while minimizing computing time.

History: Accepted by Juanjuan Zhang, marketing.
Funding: This research was supported by funding from Marketing Science Institute.
Supplemental Material: The online appendix is available at https://doi.org/10.1287/mnsc.2018.3255.

Keywords: marketing • online search • personalization • machine learning • search engines

1. Introduction

1.1. Online Search and Personalization

As the Internet has grown, the amount of information and products available in individual websites has increased exponentially. Today, Google indexes more than 130 trillion web pages (Google 2018), Amazon sells more than 562 million products (ScrapeHero 2018), and more than 400 hours of video are uploaded to YouTube every minute (Tran 2017). Although such large assortments give an abundance of choice to consumers, they also make it hard for them to locate the exact product or information they are looking for. Therefore, most businesses have adopted a query-based search model that helps consumers locate the product/information that best fits their needs. Consumers enter queries (or keywords) in a search box, and the firm presents a set of documents that is deemed most relevant to the query.

However, search is costly in both time and effort as documented by a large stream of research (Weitzman 1979, De los Santos et al. 2012, Koulayev 2014). Search costs not only affect how much a consumer searches before finding the information that she is looking for, but also whether the search is ultimately successful or not. Long and/or unsuccessful searches can have negative consequences for a firm because consumers may leave its website without clicking and/or making purchases. Firms have, therefore, long grappled with the question of how to improve the search process.

Broadly speaking, there are two ways to do this. The first is to provide a more relevant set of results. The second is to rank relevant results higher or to show them earlier in the search process so that consumers can avoid clicking on wrong results or scrolling all the way down to find them (and thereby avoid scroll and click costs). In this paper, we focus on the latter method of improving search. Indeed, the optimal ordering of results within a list is an important problem because recent research has shown that position effects have a significant impact on consumers' click behavior and firm profits (Narayanan and Kalyanam 2015, Ursu 2018).

At the first glance, this seems a trivial problem; given a set of documents, simply rank ordering them in decreasing order of relevance should ensure that consumers reach the relevant information with minimum cost. However, the difficulty lies in identifying the correct relevance value for each document for a given user and search-instance. If preferences are homogeneous across

users and search instances, then we have a globally valid rank ordering of relevance for all documents for a given query, and the ranking problem is straightforward. However, if user preferences are heterogeneous, that is, if two users using the same query are looking for different pieces of information, the problem becomes much more challenging (Mihalkova and Mooney 2009). This can be illustrated using the classic example of "java" (De Vrieze 2006). A user who searches for java can be looking to (a) get some coffee, (b) find information on the programming language Java, or (c) vacation in the Java islands. The relevance values of documents are therefore user-specific (e.g., a coffee shop is relevant to the user looking for coffee but not to the one looking for vacation packages), and the optimal rank ordering has to be individual and search-instance specific.

A potential solution to this problem is to personalize the rankings of search results. Indeed, all the major search engines have experimented with and/or deployed personalized search algorithms.[1] See Figure 1 for an example of how personalization can influence search results. However, the extent and effectiveness of search personalization has been a source of intense debate (Schwartz 2012, Hannak et al. 2013). Given the privacy implications of storing long user histories, both consumers and firms need to understand the returns to personalization and identify situations in which personalization helps.

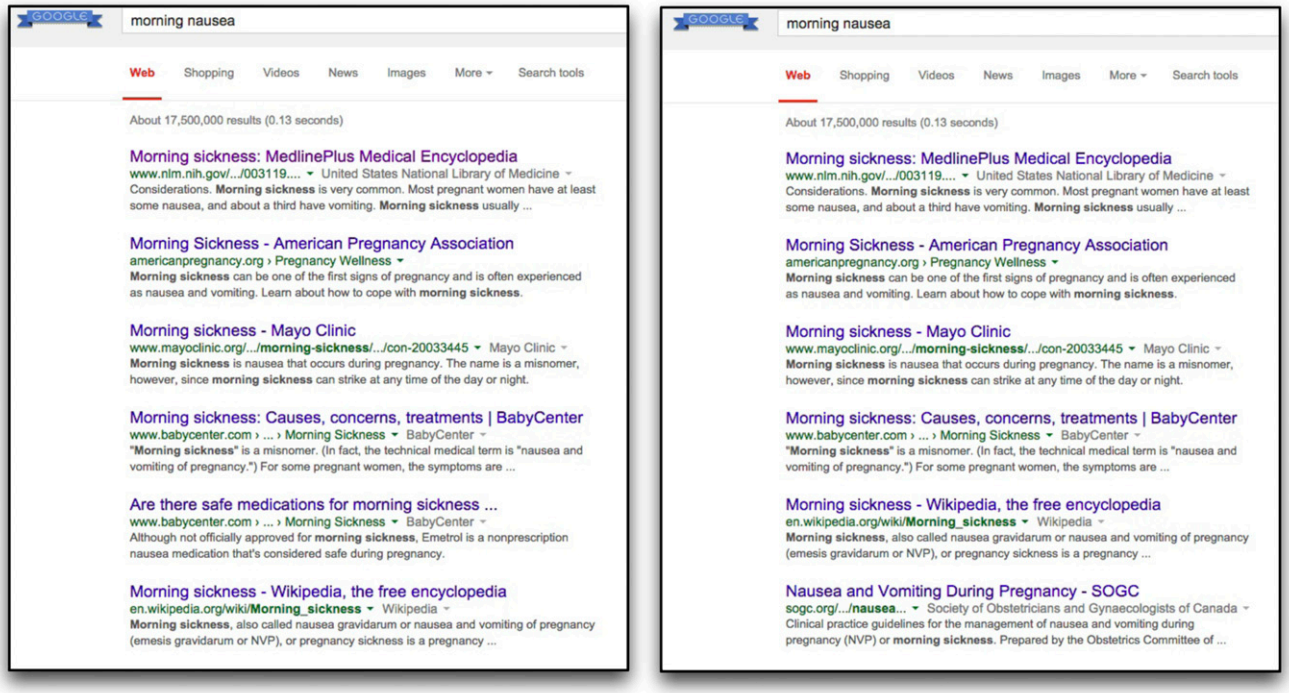
1.2. Research Agenda and Challenges

In this paper, we study personalization of search results through reranking, that is, the optimal search instance–specific rank-ordering of the first page of search results. Our goal is to build a scalable framework to implement personalized search and evaluate the returns to personalization. In the process, we seek to quantify the role of user and query-level heterogeneity on personalization.

This is a challenging problem for three reasons. First, we need a framework with extremely high out-of-sample predictive accuracy. Note that the search engine's problem is one of prediction: which result or document will a given user in a given search instance find most relevant? Formally, what is the user search–specific rank-ordering of the documents in decreasing order of relevance, such that the most relevant documents are at the top for each user search instance? Traditional econometric tools are not well suited to answer this question because they have been designed for causal inference (to derive the best unbiased estimates) and not for prediction. So they perform poorly in out-of-sample predictions. Indeed, the well-known bias–variance trade-off suggests that unbiased estimators are not always the best predictors because they tend to have high variance and vice versa (Hastie et al. 2009).

Second, we have to accommodate a large number of attributes and allow for complex interactions between

Figure 1. (Color online) An Example of Personalization



Notes. The right panel shows results with personalization turned off by using the Chrome browser in a fresh incognito session. The left panel shows personalized results. The fifth result in the left panel from the domain *babycenter.com* does not appear in the right panel. This is because the user in the left panel had previously visited this URL and domain multiple times in the past few days.

them. A commonly used modeling approach is to assume a fixed functional form to model an outcome variable, include a small set of explanatory variables, allow for a few interactions between them, and then infer the parameters associated with the prespecified functional form. However, this approach has poor predictive ability (as we later show). In our setting, we have a large number of features (close to 300), and we need to allow for nonlinear interactions between them. This problem is exacerbated by the fact that we don't know which of these variables and which interaction effects matter a priori. Indeed, it is not possible for the researcher to come up with the best nonlinear specification of the functional form either by manually eyeballing the data or by including all possible interactions. The latter would lead to more than 45,000 variables even if we restrict ourselves to pairwise interactions. Thus, we are faced with the problem of not only inferring the parameters of the model, but also inferring the optimal functional form of the model itself. So we need a framework that can search and identify the relevant set of features and all possible nonlinear interactions between them.

Finally, our approach has to be efficient and scalable. Efficiency refers to the idea that the runtime and deployment of the model should be as small as possible with minimal loss in accuracy. Thus, an efficient model may lose a small amount of accuracy but make huge gains on runtime. Scalability refers to the idea that X times increase in the data leads to no more than $2X$ (or linear) increase in the deployment time. Both scalability and efficiency are necessary to run the model in real time.

To address these challenges, we turn to machine-learning methods that are specifically designed to provide extremely high out-of-sample accuracy, work with a large number of data attributes, and perform efficiently at scale.

1.3. Our Framework and Findings

We present a general framework for search personalization that employs a three-pronged approach: (a) feature generation, (b) normalized discounted cumulative gain (NDCG)-based LambdaMART algorithm, and (c) feature selection using the wrapper method. The first module consists of a program that uses a series of functions to generate a large set of features (or attributes) that function as inputs to the subsequent modules. These features consist of both global and personalized metrics that characterize the aggregate and user-level data regarding relevance of search results, respectively. The second module consists of the LambdaMART ranking algorithm that maximizes search quality as measured by NDCG. This module employs the Lambda ranking algorithm in conjunction with the machine-learning algorithm called gradient-boosted regression trees to maximize the predictive ability of the framework. The boosted-tree algorithm performs automatic variable selection and infers both the optimal functional form (allowing for all types of interaction effects and nonlinearities) as well as the parameters of the model. It is not hampered by the large number of parameters and variables because it performs optimization efficiently in gradient space using a greedy algorithm. The third module focuses on scalability and efficiency; it consists of a wrapper algorithm that wraps around the LambdaMART step to derive the set of optimal features that maximize search quality with minimal loss in accuracy. The wrapper, thus, identifies the most efficient model that scales well.

We apply our framework to a large-scale data set from Yandex. It is the fourth largest search engine in the world with more than 60% market share in Russia and Eastern Europe (Clayton 2013, Pavliva 2013) and more than $300 million U.S. dollars in quarterly revenues (Kaggle 2013, Yandex 2016).[2] The data consists of information on anonymized user identifiers, queries, query terms, URLs, URL domains, and clicks. The data spans one month, consists of 5.7 million users, 21 million unique queries, 34 million search sessions, and more than 64 million clicks.

Because all three modules in our framework are memory- and computing-intensive, we estimate the model on the fastest servers available on Amazon EC2 with 32 cores and 256 GB of memory.

We evaluate our algorithm on the test data and present the following main findings:

1. We show that search personalization can be effective in the field. Search quality, as quantified by the average error in the rank of a click (AERC), improves by 9.43% with personalization. AERC is a position-based click metric similar in spirit to the metrics used in the recommendation-systems literature (Ricci et al. 2011). Further, personalization leads to a substantial increase in click-through rates (CTRs) at the top positions (more than 3.5% for the first position), which is a leading measure of user satisfaction.

2. Next, we quantify the value of the different attributes of the data or *features*. User-specific features generate more than 50% of the improvement from personalization (as measured by the optimization metric NDCG), click-related features generate more than 28% of the improvement, and domain- and URL-specific features contribute more than 10% each. Interestingly, session-specific features are not very useful; they contribute only 4% of the gain. Thus, across-session personalization based on user-level data is more valuable than within-session personalization. Thus, the low returns from within-session personalization may

not be sufficient to warrant the computing costs of real-time personalization.

3. We show that there is significant heterogeneity in the returns to personalization depending on the length of user history and the type of query used. Specifically,

• The quality of personalized results increases monotonically with the length of a user's search history. If we restrict the data to users with at least 50 past queries, AERC improves by more than 11%, and this number increases to 12.5% when we focus on users with at least 100 past queries. Given that storing long personal histories has privacy and storage costs, the firm can use our estimates to decide the optimal length of history to store.

• Queries that receive clicks to many URLs (across all users) benefit more from personalization. We expect the number of unique URLs clicked to be indicative of query type; navigational queries are likely to have low to no variety in the URLs clicked, and transactional and informational (e.g., java) queries are likely see a high variety in the URLs clicked. Our findings, thus, suggest that transactional and informational queries are more likely to benefit from personalization compared with navigational queries.

• Queries benefit differentially based on their past performance with poorly performing queries benefiting much more from personalization. Queries whose prior clicks, on average, occurred at positions six or lower gain more than 22% in AERC compared with the baseline.

4. We demonstrate the scalability of our method to large data sets. Going from 20% of the data to the full data set leads to a 2.38 times increase in computing time for feature generation, 20.8 times increase for training, and 11.6 times increase for testing/deployment of the final model. Thus, training and testing costs increase superlinearly, whereas feature-generation costs increase sublinearly. Crucially, none of the speed increases are exponential, and all of them provide substantial improvements in search quality.

5. Finally, to increase our framework's efficiency, we provide a forward-selection wrapper, which derives the optimal set of features that maximize search quality with minimal loss in accuracy. Of the 293 features provided by the feature-generation module, we optimally select 47 that provide 95% of the improvement in NDCG with significantly lower computing costs and speed. We find that training the pruned model takes only 15% of the time taken to train the unpruned model. Similarly, we find that deploying the pruned model is only 50% as expensive as deploying the full model.

Finally, we do caution that our results are to be interpreted cautiously because we have not evaluated our algorithm in the field. The gains can be lower if consumers react adversely to personalization. (Indeed they could also be higher if they react favorably to personalized search.)

In sum, our paper makes three main contributions to the marketing literature on personalization. First, it presents a comprehensive machine-learning framework to help researchers and managers improve personalized marketing decisions using data on consumers' past behavior. Second, it presents empirical evidence in support of the returns to personalization in the online search context. Third, it demonstrates how large-scale data can be leveraged to improve marketing outcomes without compromising the speed or real-time performance of digital applications.

2. Related Literature

Our paper relates to streams of literature in marketing and computer science.

First, it relates to the empirical literature on personalization in marketing. In an influential paper, Rossi et al. (1996) quantify the benefits of one-to-one pricing using purchase history data and find that personalization improves profits by 7.6%. Similarly, Ansari and Mela (2003) find that content-targeted emails can increase the click-throughs up to 62%, and Arora and Henderson (2007) find that customized promotions in the form of "embedded premiums" or social causes associated with the product can improve profits. On the other hand, a series of recent papers question the returns to personalization in a variety of contexts, ranging from advertising to ranking of hotels in travel sites (Zhang and Wedel 2009, Lambrecht and Tucker 2013, Ghose et al. 2014). There are two main dimensions on which our paper differs from these: scalability and predictive performance. These earlier papers are applied to small(er) data sets, and the approaches used (e.g., Markov chain Monte Carlo) are limited in scalability and are not built to maximize predictive performance. In contrast, our approach leverages techniques, such as automatic feature selection, and is scalable to extremely large data sets without compromising performance or speed.

Our paper also relates to the computer science literature on personalization of search engine rankings. Qiu and Cho (2006) infer user interests from browsing history using experimental data on 10 subjects. Teevan et al. (2008) and Eickhoff et al. (2013) show that there could be returns to personalization for some specific types of searches involving ambiguous queries and atypical search sessions. Bennett et al. (2012) find that long-term history is useful early in the session, whereas short-term history is later in the session. There are three main differences between these papers and ours. First, from a data perspective, unlike the previous papers, we suffer from the limitation of not knowing the context (i.e., information on the URLs and domains of the web pages, the hyperlink

structure between them, and the queries used in the search process). This data limitation makes our task significantly more challenging because we are not able to exploit ideas such as topic similarities across queries in our algorithm. Second, the previous papers work with much smaller data sets, which helps them avoid the scalability issues that we face. Third, we provide a new set of substantive results on the heterogeneity in the returns to personalization. Specifically, none of the previous papers establish the returns to personalization as a function of the length of user history, the prior performance of the query, or the user intent associated with the query (as discussed in Section 1.3 earlier).

Our paper leverages the machine-learning techniques related to boosting and regression trees. Boosting was first introduced in the 1990s by Schapire and Freund, who theoretically proved it is possible to generate a strong learner using a combination of weak learners (Schapire 1990, Freund 1995, Freund and Schapire 1996). In an important paper, Breiman (1998) showed that boosting can be interpreted as gradient descent in function space, a viewpoint that was expanded by Friedman (2001), who showed how boosting can be applied to a large set of loss functions using any underlying weak learner (e.g., CART, logistic functions, regressions). Since then, boosted regression trees have been used to solve a variety of prediction problems. In the marketing literature, Lemmens and Croux (2006) were the first to use boosted regression trees for a prediction task, and they showed that boosting significantly improves churn prediction. Neslin et al. (2006) conducted a meta-analysis of submissions to the Teradata challenge to identify methods that perform well and found that tree-based methods are good at predicting churn. Indeed, the winning entry in the competition used boosted regression trees.

More broadly, our paper is relevant to the growing literature on machine-learning methodologies for learning-to-rank methods. Many information-retrieval problems in business settings often boil down to ranking. Hence, ranking methods have been studied in many contexts, such as collaborative filtering (Harrington 2003), question answering (Surdeanu et al. 2008, Banerjee et al. 2009), text mining (Metzler and Kanungo 2008), and digital advertising (Ciaramita et al. 2008). We refer interested readers to Liu (2011) for details.

Our paper also contributes to the small but growing literature on machine-learning applications in marketing. Please see Dzyabura and Hauser (2011), Netzer et al. (2012), and Huang and Luo (2015) for some recent applications of machine learning in marketing and Toubia et al. (2007), Dzyabura and Yoganarasimhan (2018) for an overview of machine learning applications in marketing.

3. Data

3.1. Data Description

We use the publicly available anonymized data set released by Yandex for its personalization challenge (Kaggle 2013). The data set is from 2011 (exact dates are not disclosed) and represents 27 days of search activity. The queries and users are sampled from a large city in Eastern Europe. The data are processed such that two types of sessions are removed: those containing queries with commercial intent (as detected by Yandex's proprietary classifier) and those containing one or more of the top-K most popular queries (where K is not disclosed). This minimizes risks of reverse engineering of user identities and business interests. The current rankings shown in the data are based on Yandex's proprietary nonpersonalized algorithm.

Table 1 shows an example of a session from the data. The data contains information on the following anonymized variables:

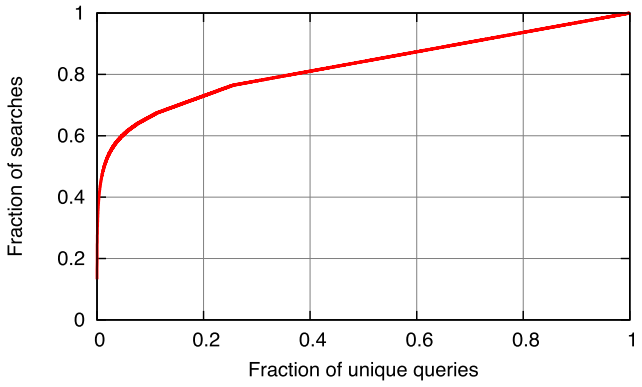
- User: A user, indexed by i, is an agent who uses the search engine for information retrieval. Users are tracked over time through a combination of cookies, IP addresses, and logged-in accounts. There are 5,659,229 unique users in the data.
- Query: A query is composed of one or more words, in response to which the search engine returns a page of results. Queries are indexed by q. There are a total of 65,172,853 queries in the data, of which 20,588,928 are unique.[3] Figure 2 shows the distribution of queries in the data, which follows a long-tail pattern: 1% of queries account for 47% of the searches and 5% for 60%.
- Term: Each query consists of one or more terms, and terms are indexed by l. For example, the query "pictures of richard armitage" has three terms: "pictures," "richard," and "armitage." There are 4,701,603 unique terms in the data, and we present the distribution of the number of terms in all the queries seen in the data in Table 2.[4]
- Session: A search session, indexed by j, is a continuous block of time during which a user is repeatedly issuing queries, browsing, and clicking on results to obtain information on a topic. Search engines use proprietary algorithms to detect session boundaries (Göker and He 2000), and the exact method used by Yandex is not publicly revealed. According to Yandex, there are 34,573,630 sessions in the data.
- URL: After a user issues a query, the search engine returns a set of 10 results in the first page. These results are URLs of web pages relevant to the query. There are 69,134,718 unique URLs in the data, and URLs are indexed by u. Yandex only provides data for the first search engine results page (SERP) for a given query because a vast majority of users never

Table 1. Example of a Session

20	M	6	5			
20	0	Q	0	6716819	2635270,4603267	70100440,5165392 17705992,1877254 41393646,3685579 46243617,3959876 26213624,2597528 29601078,2867232 53702486,4362178 11708750,1163891 53485090,4357824 4643491,616472
20	13	C	0	70100440		
20	495	C	0	29601078		

Notes. The data are in a log format describing a stream of user actions with each line representing a session, query, or a click. In all lines, the first column represents the session (SessionID = 20 in this example). The first row from a session contains the session-level metadata and is indicated by the letter "M." It has information on the day the session happened (6) and the UserID (5). Rows with the letter "Q" denote a query, and those with C denote a click. The user submitted the query with QueryID 6716819, which contains terms with TermIDs 2635270, 4603267. The URL with URLID 70100440 and DomainID 5165392 is the top result on a results page. The rest of the nine results follow similarly. Thirteen units of time into the session, the user clicked on the result with URLID 70100440 (ranked first in the list), and 495 units of time into the session, the user clicked on the result with URLID 29601078.

Figure 2. (Color online) Cumulative Distribution Function of the Number of Unique Queries: The Fraction of Overall Searches Corresponding to Unique Queries, Sorted by Popularity



go beyond the first page, and click data for subsequent pages is very sparse. So, from an implementation perspective, it is difficult to personalize later SERPs, and from a managerial perspective, the returns to personalizing them is low because most users never see them.

Table 2. Distribution of Number of Terms in a Query

Number of terms per query	Percentage of queries
1	22.32
2	24.85
3	20.63
4	14.12
5	8.18
6	4.67
7	2.38
8	1.27
9	0.64
10 or more	1.27

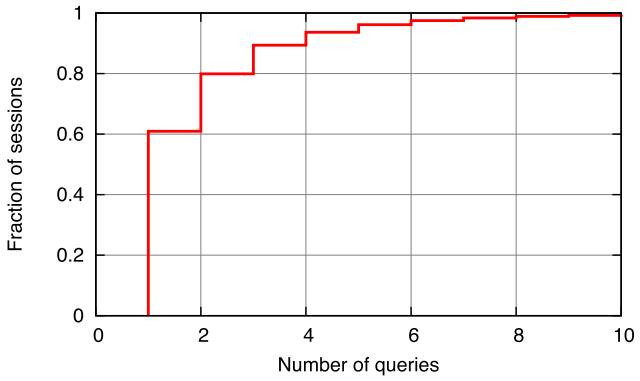
Note. (Min, max) = (1, 82).

- URL domain: This is the domain to which a given URL belongs. For example, www.imdb.com is a domain, and http://www.imdb.com/name/nm0035514/ and http://www.imdb.com/name/nm0185819/ are URLs or web pages hosted at this domain. There are 5,199,927 unique domains in the data.
- Click: After issuing a query, users can click on one or all the results on the SERP. Clicking a URL is an indication of user interest. We observe 64,693,054 clicks in the data. So, on average, a user clicks 1.007 times following a query (including repeat clicks to the same result). Table 3 shows the probability of click by position or rank of the result. Note the steep drop off in click probability with position: the first document is clicked 44.51% of times, whereas the fifth one is clicked a mere 5.56% of times.
- Time: Yandex gives us the timeline of each session. However, to preserve anonymity, it does not disclose how many milliseconds are in one unit of time. Each action in a session comes with a time stamp that tells us how long into the session the action took place. Using these time stamps, Yandex calculates "dwell time" for each click, which is the time between a click and the next click or the next query. Dwell time is informative of how much time a user spent with a document.

Table 3. Click Probability Based on Position

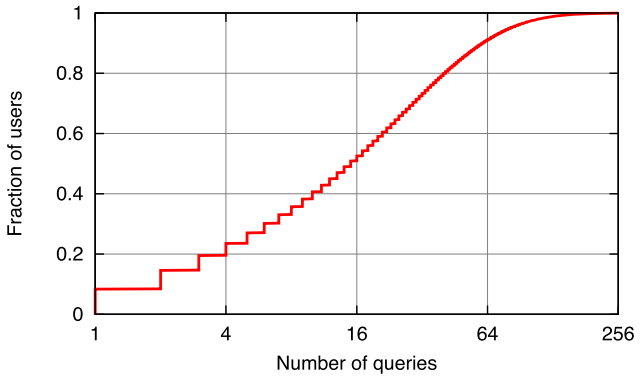
Position	Probability of click
1	0.4451
2	0.1659
3	0.1067
4	0.0748
5	0.0556
6	0.0419
7	0.0320
8	0.0258
9	0.0221
10	0.0206

Figure 3. (Color online) Cumulative Distribution Function of the Number of Queries in a Session



Note. Min, max = 1, 280.

Figure 4. (Color online) Cumulative Distribution Function of Number of Queries per User



Notes. Observed in days 26, 27. Min, max = 1, 260.

- Relevance: A result is relevant to a user if the user finds it useful and irrelevant otherwise. Search engines attribute relevance scores to search results based on user behavior: clicks and dwell time. It is well known that dwell time is correlated with the probability that the user satisfied the user's information needs with the clicked document. The labeling is done automatically and, hence, different for each query issued by the user. Yandex uses three grades of relevance:
  - Irrelevant: $r = 0$. A result gets an irrelevant grade if it doesn't get a click or if it receives a click with dwell time strictly less than 50 units of time.
  - Relevant: $r = 1$. Results with dwell times between 50 and 399 time units receive a relevant grade.
  - Highly relevant: $r = 2$. Highly relevant is given to two types of documents: (a) those that received clicks with a dwell time of 400 units or more and (b) if the document was clicked and the click was the last action of the session (irrespective of dwell time). The last click of a session is considered to be highly relvevant because a user is unlikely to have terminated the search if the user is not satisfied.
- In cases in which a document is clicked multiple times after a query, the maximum dwell time is used to calculate the document's relevance.

3.2. Model-Free Analysis

For personalization to be effective, we need to observe three types of patterns in the data. First, we need sufficient data on user history to provide personalized results. Second, there has to be significant heterogeneity in users' preferences. Third, users have to exhibit some persistence in their behavior. We examine the extent to which search and click patterns satisfy these criteria.

First, consider user history metrics. We find that 60% of the sessions have only one query (Figure 3). This implies that within-session personalization is not possible for such sessions. However, 20% of sessions have two or more queries, making them amenable to within-session personalization. Next, we examine the feasibility of across-session personalization for users observed in days 26 and 27 by considering the number of queries they have issued in the past.[5] We find that the median user has issued 16 queries, and more than 25% of the users have issued at least 32 queries (Figure 4). Together, these patterns suggest that both short- and long-term personalization are feasible in this context.

Figure 5 presents the distribution of the average number of clicks per query for the users in the data. As we can see, there is considerable heterogeneity in users' propensity to click. For example, more than 15% of users don't click at all, whereas 20% of them click on 1.5 results or more for each query they issue. To ensure that these patterns are not driven by one-time or infrequent users, we present the same analysis for users who have issued at least 10 queries in Figure 6, which shows similar click patterns. Together, these two figures establish the presence of significant user-specific heterogeneity in clicking behavior.

Figure 5. (Color online) Cumulative Distribution Function of Average Number of Clicks per Query for a Given User

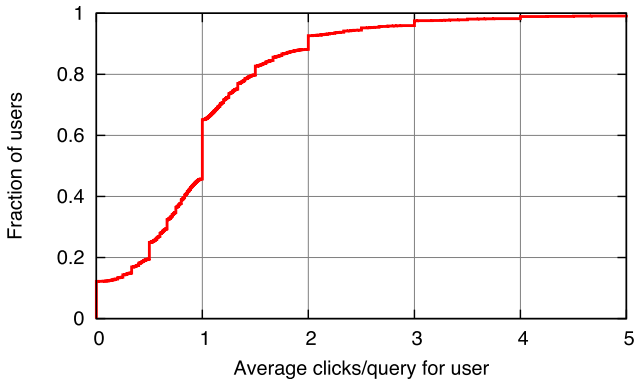


Figure 6. (Color online) Cumulative Distribution Function of Average Clicks per Query for Users Who Have Issued 10 or More Queries

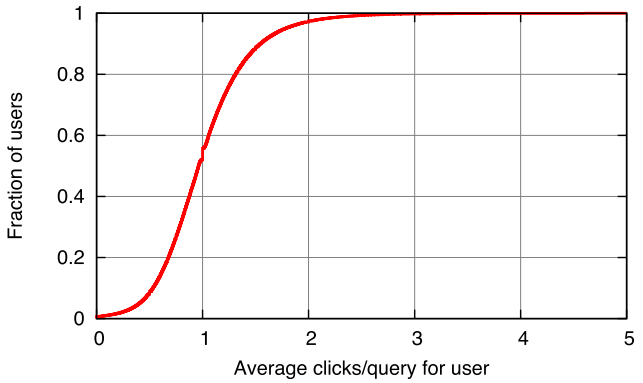
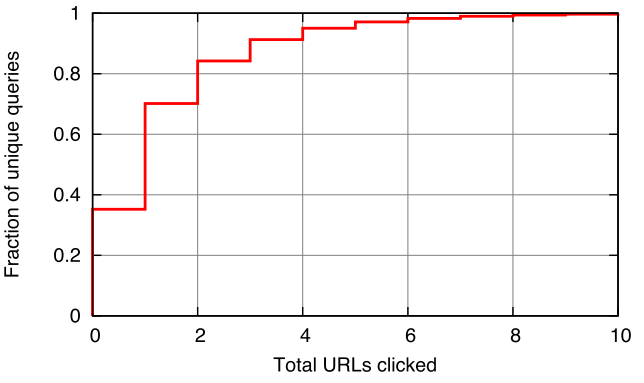


Figure 7. (Color online) Cumulative Distribution Function of Number of Unique URLs Clicked for a Given Query in the 27-Day Period



Next, we examine whether there is variation or entropy in the URLs that are clicked for a given query. Figure 7 presents the cumulative distribution of the number of unique URLs clicked for a given query through the time period of the data. Specifically, 37% of queries receive no clicks at all, and close to another 37% receive a click to only one unique URL. These are likely to be navigational queries; for example, almost everyone who searches for "cnn" will click on www.cnn.com. However, close to 30% of queries receive clicks on two or more URLs. This suggests that users are clicking on multiple results. This could be either the result of heterogeneity in users' preferences for different results or because the results themselves change a lot for the query during the observation period (high churn in the results is likely for news-related queries, such as "earthquake"). To examine if the last explanation drives all the variation in Figure 7, we plot the number of unique URLs and domains we see for a given query through the course of the data in Figure 8. We find that more than 90% of queries have no churn in the URLs shown in the results page. Hence, it is unlikely that the entropy in clicked URLs is driven by churn in search engine results. Rather, it is more likely to be driven by heterogeneity in users' tastes.

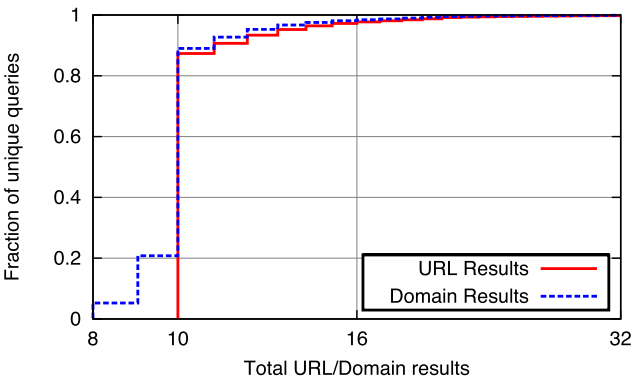
In sum, the data patterns point to significant heterogeneity in users' preferences and persistence in users' clicking behavior, both of which hint at the possibility of positive returns to personalization.

3.3. Data Sets: Training, Validation, and Testing

Supervised machine learning algorithms typically require three data sets: training, validation, and testing. The training data set is a set of preclassified data used for learning/estimating the model and inferring the model parameters. However, if we simply optimize the model on the training data set, we end up picking a model with great in-sample performance but poor out-of-sample performance. Validation data help avoid this problem. At each step of the optimization (which is done on the training data), the model's performance is evaluated on the validation data. In our analysis, we optimize the model on the training data and stop the optimization only when we see no improvement in its performance on the validation data for at least 100 steps. Then we go back and pick the model that offered the best performance on the validation data. Because both the training and validation data sets are used in model selection, we need a new data set for testing the predictive accuracy of the model. This separate holdout sample is labeled as the test data. The results from the final model on the test data are considered the final predictive accuracy achieved by our framework.[6]

Before constructing the three data sets (training, validation, and test) described, we clean the data to accommodate the test objective. Testing whether personalization is effective or not is only possible if there is at least one click. So following Yandex's recommendation, we filter the data sets in two ways. First, for each user observed in days 26–27, we only

Figure 8. (Color online) Cumulative Distribution Function of Number of Unique URLs and Domains Shown for a Given Query in the 27-Day Period



include all the user's queries from the test period with at least one click with a dwell time of 50 units or more (so that there is at least one relevant or highly relevant document). Second, if a query in this period does not have any short-term context (i.e., it is the first one in the session) and the user is new (i.e., the user has no search sessions in the training period), then we do not consider the query because it has no information for personalization.

The features described in Section 4.1 can be broadly categorized into two sets: (a) global features, which are common to all the users in the data, and (b) user-specific features that are specific to a user (could be within- or across-session features). In real business settings, managers typically have data on both global and user-level features at any given point in time. Global features are constructed by aggregating the behavior of all the users in the system. An example of a global feature is the average click-through rate for a URL u for query q across all users. (See Section 4.1 for details.) Because this feature requires aggregation over millions of users and data stored across multiple servers and geographies, it is not feasible to keep this up to date on a real-time basis. So firms usually update this data on a regular (weekly or biweekly) basis. Further, global data are useful only when they have been aggregated over a sufficiently long period of time. In our setting, we cannot generate reliable global features for the first few days of observation because of insufficient history. In contrast, user-specific features have to be generated on a real-time basis and need to include all the past activity of the user, including earlier activity in the same session. This is necessary to make personalized recommendations for each user. An example of a user-specific feature is the average click-through rate for a URL u shown in response to query q for user i until this search instance. Given these constraints and considerations, we now explain how we partition the data and generate these features.

Recall that we have a snapshot of 27 days of data from Yandex. Figure 9 presents a schema of how the three data sets (training, validation, test) are generated from the 27 days of data. First, using the entire data set from the first 25 days, we generate the global features for all three data sets. These are features that represent aggregate or population-level information (see Section 4.2 for details).

Next, we split all the users observed in the last two days (days 26–27) into three random samples (training, validation, and testing).[7] Then, for each observation of a user, all the data up until that observation is used to generate the user/session level features. For example, sessions A1, A2, B1, B2, and C1 are used to generate global features for all three data sets. Thus, the global features for all three data sets do not include information from sessions in days 26–27 (sessions A3, C2, and C3). Next, for user A in the training data, the user-level features at any given point in Session A3 are calculated based on the user's entire history starting from session A1 (when the user is first observed) until the user's last observed action in session A3. User B is not observed in the last two days. So user B's sessions are only used to generate aggregate global features; that is, the user does not appear at the user-level in the training, validation, or test data sets. For user C, the user-level features at any given point in sessions C2 or C3 are calculated based on user C's entire history starting from session C1.

4. Feature Generation

We now present the first module of our three-pronged empirical strategy: feature generation. In this module, we conceptualize and define an exhaustive feature set that captures both global and individual-level attributes that help predict the relevancy of a document for a specific search. The framework is general enough to be extrapolated to other personalization problems.

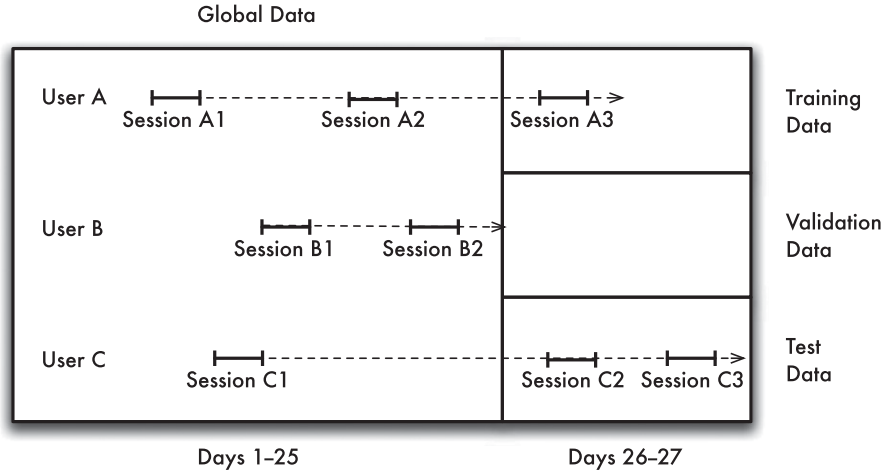
Our framework consists of a series of functions that can be used to generate a large and varied set of features succinctly. The functions typically are of the form $F(\theta_1, \theta_2, \theta_3)$, where the three inputs are associated with (1) search input (query, terms in query), (2) search results (URL, domain), and (3) the extent of personalization (global, user, user-session).

- $\theta_1 \in \{q, l_1, l_2, l_3, l_4, \emptyset\}$ contains information on the keywords used by the user in the search and can take six possible values: (1) q, the query used in the search; (2) l_1, l_2, l_3, l_4, the first, second, third, and fourth terms of the query used in the search;[8] (3) $\emptyset$, which implies that the function is aggregated over all possible queries.
- $\theta_2 \in \{u, d\}$ refers to the URL/domain of a search result. $\theta_2 = u$ implies that the function is specified over URL u, and $\theta_2 = d$ implies that it is specified over the domain d.
- $\theta_3 = \{g, it, ijt\}$ captures the granularity at which the feature is computed. $\theta_3 = g$ denotes that the function is calculated over the global data (i.e., aggregated over all the data in days 1–25; see Figure 9), and the resulting feature is a global feature. When $\theta_3 = it$, the function is calculated for user i up until the point in time t, resulting in a user-specific, real-time feature. For $\theta_3 = ijt$, function is calculated for the jth session of user i using information up until time t. This gives rise to a user-session feature that can be updated in real time.

4.1. Functions for Feature Generation

We now present a series of functions that are used to compute features. The full list of features generated using these functions are listed in Table A.1 in the online appendix.

Figure 9. Data-Generation Framework



1. $Listings(\theta_1, \theta_2, \theta_3)$: This function captures the number of times a result appears in a specific slice of the data. We give a few examples of the different invocations of this function.

- $Listings(q, u, ijt)$ calculates the number of times a URL u appeared in a SERP for searches issued by a user i for a query q in session j before a given time t.
- The same function, when invoked for $\theta_3 = g$, $Listings(q, u, g)$, computes the number of occurrences of URL u in a SERP for query q in global data g.
- If we supply the term l_1 as the input for θ_1 instead of a query, then the invocation $Listings(l_1, u, ijt)$ counts the number of times URL u appeared in a SERP for searches issued by a user i when the user used l_1 as the first term in the query in session j before a given time t.
- $Listings(\emptyset, u, g)$ counts the number of times URL u appears in the SERP in the global data g irrespective of the query or search term used.
- If we supply domain d as the input to θ_2, then $Listings(q, d, g)$ counts the number of times results from domain d appeared in the SERP in searches for query q in global data g.

These are just a few examples of the possible features that the function $Listings(\theta_1, \theta_2, \theta_3)$ can generate. In total, based on the number of combinations of different inputs, we can generate $\text{Dim}(\theta_1) \times \text{Dim}(\theta_2) \times \text{Dim}(\theta_3) = 6 \times 2 \times 3 = 36$ features.

We now list seven more features that take the same (or similar) inputs. To avoid repetition, we do not provide detailed examples of possible features for each of these functions. Instead, we refer readers to Table A.1 for the complete list of features.

2. $AvgPosn(\theta_1, \theta_2, \theta_3)$: This function calculates the average rank of a result within SERPs that list it for a given $\theta_1, \theta_2, \theta_3$. For example, $AvgPosn(q, u, ijt)$ denotes the average rank of URL u in response to searches for query q by user i in session j before time t. It provides information over and above the *Listings* function as it characterizes the average ranking of the result as opposed to simply noting its presence in the SERP.

3. $Clicks(\theta_1, \theta_2, \theta_3)$: This function calculates the number of times a click occurred for a given $\theta_1, \theta_2, \theta_3$. For example, $Clicks(q, u, ijt)$counts the number of times user i clicked on URL u after seeing it on a SERP provided in response to searches for query q by user i in session j before time t. *Clicks* can be used in conjunction with *Listings* to compute CTRs as follows:

$$CTR(\theta_1, \theta_2, \theta_3) = \frac{Clicks(\theta_1, \theta_2, \theta_3)}{Listings(\theta_1, \theta_2, \theta_3)}. \quad (1)$$

4. $HDClicks\ (\theta_1, \theta_2, \theta_3)$: This function calculates the number of times a high-duration click occurred for the given inputs. *HDClicks* can then be used to compute high-duration click-through rates as follows:

$$HDCTR(\theta_1, \theta_2, \theta_3) = \frac{HDClicks(\theta_1, \theta_2, \theta_3)}{Listings(\theta_1, \theta_2, \theta_3)}. \quad (2)$$

HDClicks and *HDCTR* have information over and beyond *Clicks* and *CTR* because they consider clicks of high relevance.

5. $WClicks(\theta_1, \theta_2, \theta_3)$: When consumers click on multiple documents with a SERP, then there is information in the order of the clicks because documents clicked earlier tend to be more relevant. $WClicks(\theta_1, \theta_2, \theta_3)$ is the weighted count of all clicks for this set of inputs, and the weight associated with a click is the reciprocal of the click order for the SERP. That is, the weight is one if it was the first click, 1/2 if it was the second, 1/3 if it was the third, and so on. *WClicks* is used to compute weighted CTRs (*WCTR*s) as follows:

$$WCTR(\theta_1, \theta_2, \theta_3) = \frac{WClicks(\theta_1, \theta_2, \theta_3)}{Listings(\theta_1, \theta_2, \theta_3)}. \quad (3)$$

If URL u is always clicked first when it is displayed, $WCTR(\theta_1, u, \theta_3) = CTR(\theta_1, u, \theta_3)$; if, instead, it is always clicked second, then $WCTR\ (\theta_1, u, \theta_3) = \frac{1}{2} CTR(\theta_1, u, \theta_3)$.

6. $Dwells(\theta_1, \theta_2, \theta_3)$: This function calculates the average dwell time for a search result for a set of inputs. *Dwells* can be used to compute the average dwell time (*ADT*), which has additional information on the relevance of URL u over and above CTR.

$$ADT(\theta_1, \theta_2, \theta_3) = \frac{Dwells(\theta_1, \theta_2, \theta_3)}{Clicks(\theta_1, \theta_2, \theta_3)}. \quad (4)$$

7. $Skips(\theta_1, \theta_2, \theta_3)$: Previous research suggests that users usually go over the list of results in a SERP sequentially from top to bottom (Craswell et al. 2008). Thus, if the user skips a top URL and clicks on one in a lower position, it is significant because the decision to skip the top URL is a conscious decision. Thus, if a user clicks on position 2 after skipping position 1, the implications for the URLs in positions 1 and 3 are different. The URL in position 1 has been examined and deemed irrelevant, whereas the URL in position 3 may not have been clicked because the user never reached it. Thus, the lack of a click is more damning for the skipped URL in position 1 compared with the one in position 3.

The function *Skips* captures the number of times a URL/domain has been skipped. For instance, consider a case in which a user clicks on the URL in position 5 followed by that in position 3. Then URLs in positions 1 and 2 were skipped twice, the URLs in positions 3 and 4 were skipped once, and those in positions 6–10 were skipped zero times.

Using *Skips*, we can compute *SkipRate* as follows:

$$SkipRate(\theta_1, \theta_2, \theta_3) = \frac{Skips(\theta_1, \theta_2, \theta_3)}{Listings(\theta_1, \theta_2, \theta_3)}. \quad (5)$$

8. $PosnClicks(\theta_1, \theta_2, \theta_3, p)$. The fourth input variable p refers to the position of the URL/domain in the SERP. Then

$$PosnClicks(\theta_1, \theta_2, \theta_3, p) = \frac{p}{10} \times Clicks(\theta_1, \theta_2, \theta_3). \quad (6)$$

If a URL in a low position gets clicked, it is likely to be more relevant than one that is in a high position and gets clicked. This stems from the idea that users usually go through the list of URLs from top to bottom and that they only click on URLs in lower positions if they are especially relevant. Using this, we can calculate the position-weighted CTR as follows:

$$PosnCTR(\theta_1, \theta_2, \theta_3, p) = \frac{PosnClicks(\theta_1, \theta_2, \theta_3, p)}{Listings(\theta_1, \theta_2, \theta_3)}. \quad (7)$$

As in the case of *Listings*, all these functions can be invoked for different combinations of $\{\theta_1, \theta_2, \theta_3\}$ giving us 36 features each for functions 1–6. For the seventh function, *SkipRate*, we did not find domain-specific skip features to be useful. So we only have 18 URL-specific *Skip* features.

In addition, we also include *Listings*, *CTR*, *HDCTR*, and *ADT* at the user level and user-session level, aggregated over all URLs and domains as follows:

9. $AggListings(\theta_3)$
10. $AggCTR(\theta_3)$
11. $AggHDCTR(\theta_3)$
12. $AggADT(\theta_3)$.

Here $\theta_3 = \{it, ijt\}$. This gives us another eight features that capture information on individual user behavior across and within a session. See features 271–278 in Table A.1.

Finally, we also use the following functions to generate a few additional stand-alone features.

13. $QueryTerms\ (q)$ is the total number of terms in a given query.

14. $Day(i, j)$ is the day of the week (Monday, Tuesday, etc.) in which user i conducts the search session j.

15. $ClickProb(i, p, t)$ is the probability that a user i clicks on a search result that appears at position p within a SERP before a given instance of time t. This function captures the heterogeneity in users' preferences for clicking on results appearing at different positions.

16. $NumSearchResults(q, t)$ is the number of unique search results appearing on any SERP associated with query q from day 1 until time t. It is an entropy metric on the extent to which search results associated with a query vary over time.

17. $NumClickedURLs(q, t)$ is the total number of unique URLs clicked on any SERP associated with q before time t. It is a measure of the heterogeneity in users' tastes for information/results within a query. For instance, for navigational queries, this number is likely to be very low, whereas for do-know queries, it is likely to be higher.

18. $SERPRank(q, u, i, j, k)$: Finally, we include the position/rank of a result for the kth search in session j by user i for query q based on Yandex's algorithm (which did not take personalized history into account). *SERPRank* is the same as the position p of the URL in the results page for this specific search. However, we define it as a separate feature because the position of a URL can vary across observations.

Recall that many aspects of the query, URL, and context are not visible to us because of anonymization. Such features play an important role in ranking algorithms, and not having information on them is a drawback (Qin et al. 2010). By using *SERPRank* as one of the features, we are able to fold in this information into a personalized ranking algorithm. Although this is not perfect (because there is information loss as a result of discretization and because we cannot interact the underlying features with the user-specific ones generated here), it is the best data we have.

All the features generated from these functions are listed in Table A.1 in the online appendix.

4.2. Categorization of Features

The total number of features that we generate from these functions is large (293 to be specific). To aid exposition and experiment with the impact of including or excluding certain sets of features, we group the set of features based on the three inputs used to generate them, $\{\theta_1, \theta_2, \theta_3\}$, which are associated with (a) search input (query, terms in query), (b) search results (URL, domain), and (c) the extent of personalization (global, user, user-session). This gives us the following classifications:

1. Query features F_Q: based on the query used for the search; that is, $\theta_1 = q$.
2. Term features F_T: based on the terms in the search query; that is, $\theta_1 = \{l_1, l_2, l_3, l_4\}$.
3. URL features F_U: based on the URL of the search result; that is, $\theta_2 = u$.
4. Domain features F_D: based on the domain name of the search result; that is, $\theta_2 = d$.
5. Global features F_G: features that use global data; that is, $\theta_3 = g$.
6. User features F_P: calculated at the person or user level (but not session level); that is, $\theta_3 = it$.
7. Session features F_S: features that are calculated at the user-session level; that is, $\theta_3 = ijt$.

The third column of Table A.1 in the online appendix provides a mapping from features to feature classes.

5. Ranking Methodology

We now present the second module of our empirical framework, which consists of a methodology for reranking the top P results for a given query by a given user at a given point in time. The goal of reranking is to promote URLs that are more likely to be relevant to the user and demote those that are less likely to be relevant. Relevance is data-driven and defined based on clicks and dwell times.

5.1. The Reranking Problem

We now present the general structure of the reranking problem. Let r_{ijk}^{qu} denote the relevance of a URL u for the kth search in the jth session of user i in response to query q. Note that the relevance labels for the same documents/URLs can vary with queries, users, and across and within the session, hence the superscripting by q and subscripting by i, j, and k.

Let $\mathscr{L}(P, i, j, k, q)$ be a list of P documents shown in response to a query q for the kth search in the jth session of a user i. Let the position of a document u in list $\mathscr{L}(P, i, j, k, q)$ be $p(u, \mathscr{L}(P, i, j, k, q))$.

Definition 1 (Reranking Problem). Consider a list of P documents $\mathscr{L}(P, i, j, k, q)$. The optimal reranking of this set of documents is a list $\mathscr{L}^*(P, i, j, k, q)$ such that the relevance of ordering of documents is nonincreasing. That is, for two documents m and n in $\mathscr{L}^*(P, i, j, k, q)$, if $r_{ijk}^{qm} > r_{ijk}^{qn}$, then the position of $m <$ position of n; that is, $p(m, \mathscr{L}^*(P, i, j, k, q)) < p(n, \mathscr{L}^*(P, i, j, k, q))$.

When a list is optimally reranked, a more relevant document is never displayed below a less relevant document. Note that this implies that we are indifferent between the rank-orderings of two equally relevant documents. For example, if two documents m and n both have relevance scores of two, then we are indifferent whether m is ranked above n or vice versa. Thus, the reranking problem has more than one solution when we have duplicate relevance scores in the list. This is certainly be the case when the number of positions is greater than the number of relevance scores (such as our case, in which we have 10 positions and three relevance scores).

Our goal is to come up with a reranking algorithm that takes as inputs a set of features and a current ordering of documents $\mathscr{L}$ and generates a new rank-ordering of documents that is as close as possible to the optimal list, $\mathscr{L}^*$, as the output.

5.2. Optimization Metric

To evaluate whether a specific reranking algorithm improves the rank-ordering of the results or not, we first need to define a metric on which to optimize.

First, consider a strawman metric: number of clicks until position p or C@p (or when averaged over many searches, CTR@p). In a reranking problem with 10 positions, C@10 is a constant for all possible orderings of documents in a list and, hence, cannot be optimized on. This is because the total number of clicks observed across all documents is the same irrespective of how we reorder them. Next, consider the total number of clicks at top k positions, say, C@3. This is problematic for a number of reasons. First, it is indifferent to the actual position of the click in the top three positions; for example, clicks to positions 1 and 2 ≡ clicks to positions 2 and 3 although the former is preferable from a user's perspective. Second, it ignores the ranking of relevant documents after the third position; for example, it is indifferent between clicks to positions 4 and 10 even though the former is better. Thus, the information from the lower positions is completely ignored. Third, it treats all clicks as the same and, thus, ignores the extra information on the relative relevance of clicks (high-relevance clicks with $r = 2$ and low-relevance clicks with $r = 1$). For these reasons, clicks or CTR is never used directly as an optimization metric in the information-retrieval literature. Nevertheless, changes in clicks to the top p positions can still be used as an outcome measure of interest after we optimize the model on more

appropriate metrics. In Section 6.1.2, we present results on this dimension.

A good optimization metric for reranking a list should ideally satisfy the following two properties (Liu 2009). First, it should be evaluated over a "list" of documents, such that it penalizes mistakes at the top of the list more heavily than those at the bottom. This mimics the response of human responders (Cao et al. 2007). Second, it should be able to handle multiple levels of relevance and not just binary relevance levels (in our case, $r \in \{0, 1, 2\}$).

A metric that satisfies both these properties is NDCG, a list-level, graded relevance metric. NDCG is both theoretically well grounded and practically popular. It is used extensively by search engines (including Yandex) and is the metric of choice in the recommendation-system literature (Wang et al. 2013). Intuitively, NDCG measures the usefulness, or gain, of a document based on its position in the result list.

We start by defining the simpler metric, discounted cumulative gain, DCG (Järvelin and Kekäläinen 2000, 2002). DCG at truncation position P is defined as

$$\mathrm{DCG}_P = \sum_{p=1}^{P} \frac{2^{r_p} - 1}{\log_2(p+1)}, \tag{8}$$

where r_p is the relevance rating of the document (or URL) in position p. A problem with DCG is that the contribution of each query to the overall DCG score of the data varies. For example, a search that retrieves four ordered documents with relevance scores $2, 2, 1, 1$ will have higher DCG scores compared with a different search that retrieves four ordered documents with relevance scores $1, 1, 0, 0$. This is problematic, especially for reranking algorithms, with which we only care about the relative ordering of documents.

To address this issue, a normalized version of DCG is used:

$$\mathrm{NDCG}_P = \frac{\mathrm{DCG}_P}{\mathrm{IDCG}_P}, \tag{9}$$

where IDCG_P is the ideal DCG at truncation position P. The IDCG_P for a list is obtained by rank-ordering all the documents in the list in decreasing order of relevance. NDCG always takes values between zero and one. Because NDCG is computed for each search in the training data and then averaged over all queries in the training set, no matter how poorly the documents associated with a particular query are ranked, it will not dominate the evaluation process because each query contributes equally to the average measure. In our analysis, we use NDCG_{10}, that is, NDCG at position 10 (henceforth referred to as NDCG for notational simplicity) as our optimization metric.

In Online Appendix C, we present model comparisons with other optimization metrics and in Online Appendix C.3, we consider truncation values other than 10 for P in NDCG_P.

5.3. The Learning-to-Rank Algorithm

We now describe the learning-to-rank algorithm used to optimize NDCG. To fix ideas and aid exposition, we first start by describing a much simpler problem: that of optimizing the ranking of pairs of documents.

5.3.1. Pairwise Learning. In pairwise evaluation, let $\mathscr{U}_{ijk}^{qm} \lhd \mathscr{U}_{ijk}^{qn}$ denote the event that URL m should be ranked higher than n for query q for user i in the kth search of the jth session. For example, if m has been labeled highly relevant ($r_{ijk}^{qm} = 2$) and n has been labeled irrelevant ($r_{ijk}^{qn} = 0$), then $\mathscr{U}_{ijk}^{qm} \lhd \mathscr{U}_{ijk}^{qn}$. Let $x_{ijk}^{qu} \in \mathscr{R}^F$ denote a set of F features for document/URL u in response to query q for the kth search in the jth session of user i. The function $f(\cdot)$ maps the feature vector to a score s_{ijk}^{qu} such that $s_{ijk}^{qu} = f(x_{ijk}^{qu}; w)$, where w is a set of parameters that define the function $f(\cdot)$. s_{ijk}^{qu} can be interpreted as the gain or value associated with document u for query q for user i in the kth search in the jth session.

Given two documents m and n associated with a search, we can write out the probability that document m is more relevant than n using a logistic function as follows:

$$P_{ijk}^{qmn} = P(\mathscr{U}_{ijk}^{qm} \lhd \mathscr{U}_{ijk}^{qn}) = \frac{1}{1 + e^{-\sigma(s_{ijk}^{qm} - s_{ijk}^{qn})}}, \tag{10}$$

where σ determines the shape of this sigmoid function. Using these probabilities, we can define the log-likelihood of observing the data or the cross entropy cost function, $\mathscr{C}$, as follows. This is interpreted as the penalty imposed for deviations of model-predicted probabilities from the desired probabilities.

$$\mathscr{C}_{ijk}^{qmn} = -\bar{P}_{ijk}^{qmn} \log(P_{ijk}^{qmn}) - (1 - \bar{P}_{ijk}^{qmn}) \log(1 - P_{ijk}^{qmn}), \tag{11}$$

$$\mathscr{C} = \sum_i \sum_j \sum_k \sum_{\forall m,n} \mathscr{C}_{ijk}^{qmn}, \tag{12}$$

where $\bar{P}_{ijk}^{qmn}$ is the observed probability (in data) that m is more relevant than n; that is, $\mathscr{U}_{ijk}^{qm} \lhd \mathscr{U}_{ijk}^{qn}$ or $r_{ijk}^{qm} > r_{ijk}^{qn}$. The summation in Equation (12) is taken in this order: first, over all pairs of URLs ($\{m, n\}$) in the consideration set for the kth search in session j for user i, and next, over all the searches within session j, then over all the sessions for user i, and finally over all users.

If the function $f(\cdot)$ is assumed to be known, the cost $\mathscr{C}$ can be minimized using commonly available optimizers such as Newton–Raphson or BFGS for the assumed functional form. This approach can be interpreted as the traditional pairwise maximum likelihood used in the marketing literature. It works well when the

researcher has a good model of consumers' behavior and the goal is causal inference. However, assuming the functional form $f(\cdot)$ is detrimental when the goal is prediction because it is impossible to a priori determine the correct functional form of $f(\cdot)$ for complex decision processes. Manual experimentation to arrive at an optimal functional form becomes exponentially difficult as the number of features expands. For example, with 293 features, we can have a total of $2^{293}-1$ possible interactions, and these would expand even more if we want to allow for nonlinearities for each feature. To address these challenges, we turn to solutions from the machine-learning literature that infer both the optimal parameters w and function $f(\cdot)$ from the data.[9]

In the context of pairwise evaluation, Burges et al. (2005) suggest a gradient descent formulation to minimize the cross-entropy cost function defined in Equations (11) and (12) as follows.[10] Let $S_{ijk}^{qmn} \in \{0,1,-1\}$, where $S_{ijk}^{qmn}=1$ if m is more relevant than n, -1 if n is more relevant than m, and zero if they are equally relevant. Thus, we have $\bar{P}_{ijk}^{qmn}=\frac{1}{2}(1+S_{ijk}^{qmn})$, and we can rewrite $\mathscr{C}_{ijk}^{qmn}$ as

$$\mathscr{C}_{ijk}^{qmn}=\frac{1}{2}(1-S_{ijk}^{qmn})\sigma(s_{ijk}^{qm}-s_{ijk}^{qn})+\log(1+e^{-\sigma(s_{ijk}^{qm}-s_{ijk}^{qn})}). \tag{13}$$

Note that cross-entropy function is symmetric; it is invariant to swapping m and n and flipping the sign of S_{ijk}^{qmn}.[11] Further, we have

$$\begin{aligned}
\mathscr{C}_{ijk}^{qmn}&=\log(1+e^{-\sigma(s_{ijk}^{qm}-s_{ijk}^{qn})}) \quad \text{if} \quad S_{ijk}^{qmn}=1\\
\mathscr{C}_{ijk}^{qmn}&=\log(1+e^{-\sigma(s_{ijk}^{qn}-s_{ijk}^{qm})}) \quad \text{if} \quad S_{ijk}^{qmn}=-1\\
\mathscr{C}_{ijk}^{qmn}&=\frac{1}{2}\sigma(s_{ijk}^{qm}-s_{ijk}^{qn})+\log(1+e^{-\sigma(s_{ijk}^{qm}-s_{ijk}^{qn})}) \quad \text{if} \quad S_{ijk}^{qmn}=0.
\end{aligned} \tag{14}$$

The gradient of the cross-entropy cost function is

$$\frac{\partial \mathscr{C}_{ijk}^{qmn}}{\partial s_{ijk}^{qm}}=-\frac{\partial \mathscr{C}_{ijk}^{qnm}}{\partial s_{ijk}^{qn}}=\sigma\left(\frac{1}{2}(1-S_{ijk}^{qmn})-\frac{1}{1+e^{-\sigma(s_{ijk}^{qm}-s_{ijk}^{qn})}}\right). \tag{15}$$

We can now employ a stochastic gradient descent algorithm to update the parameters, w, of the model, as follows:

$$w \rightarrow w-\eta\frac{\partial \mathscr{C}_{ijk}^{qmn}}{\partial w}=w-\eta\left(\frac{\partial \mathscr{C}_{ijk}^{qmn}}{\partial s_{ijk}^{qm}}\frac{\partial s_{ijk}^{qm}}{\partial w}+\frac{\partial \mathscr{C}_{ijk}^{qnm}}{\partial s_{ijk}^{qn}}\frac{\partial s_{ijk}^{qn}}{\partial w}\right), \tag{16}$$

where η is a learning rate specified by the researcher. The advantage of splitting the gradient as shown in Equation (16) instead of working with $\partial \mathscr{C}_{ijk}^{qmn}/\partial w$ is that it speeds up the gradient calculation by allowing us to use the analytical formulations for the component partial derivatives.

In addition, factorization can be used to improve speed as follows:

$$\frac{\partial \mathscr{C}_{ijk}^{qmn}}{\partial w}=\lambda_{ijk}^{qmn}\left(\frac{\partial s_{ijk}^{qm}}{\partial w}-\frac{\partial s_{ijk}^{qn}}{\partial w}\right), \tag{17}$$

where

$$\lambda_{ijk}^{qmn}=\frac{\partial \mathscr{C}_{ijk}^{qmn}}{\partial s_{ijk}^{qm}}=\sigma\left(\frac{1}{2}(1-S_{ijk}^{qmn})-\frac{1}{1+e^{-\sigma(s_{ijk}^{qm}-s_{ijk}^{qn})}}\right). \tag{18}$$

Given this cost and gradient formulation, we can use any machine-learning model (regression trees, neural networks) to minimize the cost function and infer both the optimal $f(\cdot)$ and w parameters.

5.3.2. LambdaRank: Optimizing for NDCG. Now we explain how this method can be extended to a more complex optimization metric: NDCG. Unlike the continuous cross-entropy cost function described in Equation (12), NDCG is discrete. This makes its optimization challenging. For instance, if we were to use an optimizer that uses continuous scores on NDCG, then we would find that the relative positions of two documents (based on NDCG) would not change unless there are significant changes in their rank scores. So we could obtain the same rank orderings for multiple parameter values of the model. The LambdaRank algorithm addresses this challenge (Burges et al. 2006, Wu et al. 2010).

The speed and versatility of the pairwise learner comes from its gradient descent formulation, which requires the loss or cost function to be twice differentiable in model parameters. Ideally, we would like to preserve the gradient descent formulation because of its attractive convergence properties while modifying it to work for a discrete optimization metric (such as NDCG).[12] The main insight of LambdaRank is that we can directly train a model without explicitly specifying a cost function as long as we have gradients of the cost and these gradients are consistent with the cost function that we wish to minimize.

The gradient functions, λ_{ijk}^{qmn}s in Equation (18), can be interpreted as directional forces that pull documents in the direction of increasing relevance. If a document m is more relevant than n, then m gets an upward pull of magnitude $|\lambda_{ijk}^{qmn}|$, and n gets a downward pull of the same magnitude. LambdaRank expands this idea of gradients functioning as directional forces by weighing them with the change in NDCG caused by swapping the positions of m and n as follows:

$$\lambda_{ijk}^{qmn}=\frac{-\sigma}{1+e^{-\sigma(s_{ijk}^{qm}-s_{ijk}^{qn})}}|\Delta\text{NDCG}_{ijk}^{qmn}|. \tag{19}$$

Note that $\Delta\text{NDCG}_{ijk}^{qmn} = \Delta\text{NDCG}_{ijk}^{qmn} = 0$ when m and n have the same relevance, so $\lambda_{ijk}^{qmn} = 0$ in those cases. (This is the reason why the term $\frac{\sigma}{2}(1 - S_{ijk}^{qmn})$ drops out.) Thus, the algorithm does not spend effort swapping two documents of the same relevance.

Given that we want to maximize NDCG, suppose that there exists a utility or gain function $\mathscr{C}_{ijk}^{qmn}$ that satisfies the preceding gradient equation. (We refer to $\mathscr{C}_{ijk}^{qmn}$ as a gain function in this context as opposed to a cost function because we are looking to maximize NDCG.) Then, we can continue to use gradient descent to update our parameter vector w as follows:

$$w \rightarrow w + \eta \frac{\partial \mathscr{C}_{ijk}^{qmn}}{\partial w}. \tag{20}$$

Using Poincare's lemma and a series of empirical results, Yue and Burges (2007) and Donmez et al. (2009) have shown that the gradient functions defined in Equation (19) correspond to a gain function that maximizes NDCG. Thus, we can use these gradient formulations and follow the same set of steps as we did for pairwise learner to maximize NDCG (or mean average precision [MAP] and other discrete optimization metrics).

LambdaRank can be used in conjunction with any machine-learning algorithm (e.g., neural nets, CART, gradient-boosted regression trees). The most successful implementation of LambdaRank uses stochastic gradient-boosted trees as the learning algorithm and is referred to as LambdaMART (Wu et al. 2010, Chapelle and Chang 2011).[13] Stochastic gradient boosting with trees is a machine-learning algorithm that models a dependent or output variable as a linear combination of a set of shallow regression trees (a process known as boosting). Gradient-boosted trees have many good properties. They are insensitive to monotone transforms in input variables, robust to outliers and missing data, and perform automatic variable selection (Breiman et al. 1984, Elith et al. 2008, Murphy 2012). They have been empirically shown to be the best classifier available (Caruana and Niculescu-Mizil 2006, Hastie et al. 2009). Please see Online Appendix B for a detailed discussion of the method.

5.4. Model Training and Tuning

The performance of machine-learning models depends on a set of tunable parameters. In the case of LambdaMART, the researcher has control over the number of regression trees used, the number of leaves in a given tree, and the type of normalization employed on the features, etc. It is important to ensure that the tuning parameters lead to a robust model. A nonrobust model usually leads to overfitting; that is, it leads to higher improvement in the training data (in-sample fit) but lower improvements in validation and test data sets (out-of-sample fits). We take two key steps to ensure that the final model is robust. First, we normalize the features using their z-scores to make the feature scales comparable. Second, we experimented with a large number of specifications for the number of trees and leaves in a tree and found that 1,000 trees and 20 leaves per tree was the most robust combination.

In our analysis, we use the *RankLib* implementation of LambdaMART (Dang 2011). *RankLib* is well parallelized and offers an exhaustive set of evaluation metrics and learning to rank algorithms, which allows us perform model comparisons. These properties make it ideal for our purpose. In future, researchers may want to also consider newer implementations of boosted regression trees, such as *XGBoost*, which implements a version of Newton boosting instead of stochastic gradient boosting (Chen and Guestrin 2016). Please see Rafieian and Yoganarasimhan (2019) for a marketing application that uses XGBoost.

6. Returns to Personalization

We apply our framework to data from Yandex and present a series of main results. In Section 6.1, we evaluate the returns from personalization on three metrics of search quality. Then in Section 6.2, we quantify the value of different feature classes. In Section 6.3, we demonstrate the robustness of our findings using a series of tests and present results on the scalability of our approach.

6.1. Evaluation of Gains

There are two broad ways to evaluate the gains from personalization. First, we could simply consider the improvement in model fit based on the optimization metric (NDCG) used to estimate the model. However, optimization metrics (e.g., NDCG, log-likelihood, and others that we consider in Section 6.3.1) are somewhat opaque when it comes to evaluating the substantive improvements in search experience from the perspective of a marketing manager. Therefore, in addition to NDCG, we also consider two managerially relevant evaluation metrics: (1) AERC and (2) change in click through rates by position (ΔCTR@p and ΔHDCTR@p).

6.1.1. Average Error in the Rank of a Click. Intuitively, in a well-optimized list, clicks occur higher up in the list, whereas in a poorly optimized list (which ranks relevant documents lower down) clicks occur lower in the list. Thus, measures based on the position of clicks within a list are often used to evaluate the performance of ranking algorithms in the recommendation-systems literature (Ricci et al. 2011). We define a similar metric for our context: AERC. This metric captures the extent of error we see in the ranking of clicked documents.

First, we define the "error in rank" or ERC for a clicked document u in list $\mathscr{L}(P, i, j, k, q)$ of P documents, which is generated in response to the kth search

in the jth session by user i when the user uses query q. ERC for u is the absolute difference between u's current position in list $\mathscr{L}(P,i,j,k,q)$ and its position in a completely optimized list $\mathscr{L}^*(P,i,j,k,q)$. Mathematically,

$$\mathrm{ERC}(u,\mathscr{L}(P,i,j,k,q)) = \mathrm{Ind}(r_{ijk}^{qu} \neq 0)\cdot|p(u,\mathscr{L}(P,i,j,k,q)) - p(u,\mathscr{L}^*(P,i,j,k,q)|, \tag{21}$$

where $\mathrm{Ind}(r_{ijk}^{qu} \neq 0)$ is an indicator for whether document u was clicked or not, that is, whether it had relevance greater than zero. Then, AERC for a list $\mathscr{L}$ of results in a SERP is the average of ERCs for all the clicked documents observed in that list. That is,

$$\mathrm{AERC}(\mathscr{L}(P,i,j,k,q)) = \frac{\sum_{u\in\mathscr{L}} \mathrm{ERC}(u,\mathscr{L}(P,i,j,k,q))}{\sum_{u\in\mathscr{L}} \mathrm{Ind}(r_{ijk}^{qu} \neq 0)}. \tag{22}$$

An illustration of AERC for two lists is shown in Figure 10. The AERC for the entire data are the average of $AERC(\mathscr{L})$ over all the lists observed in the data.

Improvement in AERC: With personalization, we find that AERC improves by 9.43% compared with the baseline. This is a significant gain and highlights the effectiveness of our personalization algorithm. It suggests that the positions of clicks migrate upward when ranking is personalized and that the average error in how documents are ranked reduces by a good amount.

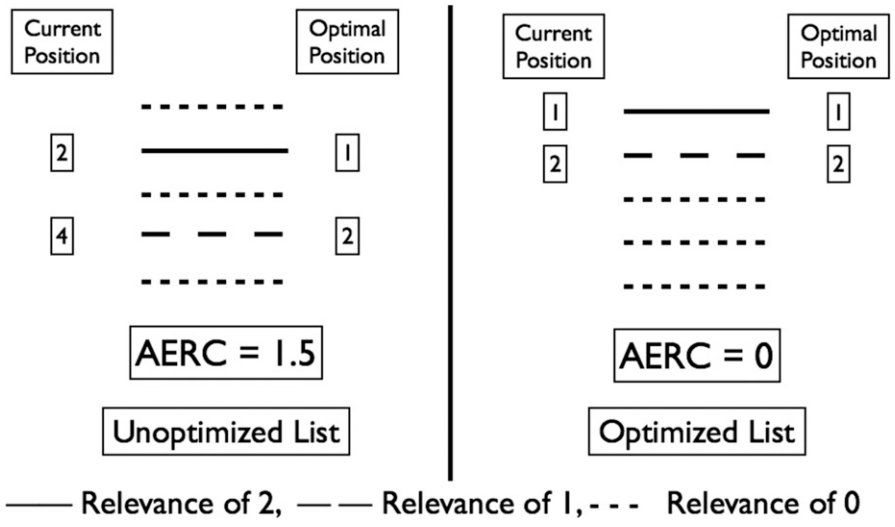
These improvements are significant in the search industry. Recall that the baseline ranking captures a large number of factors, including the relevance of page content, page rank, and the link structure of the web pages, which are not available to us. It is, therefore, notable that we are able to further improve on the ranking accuracy by incorporating personalized user-/session-specific features.

6.1.2. Change in Click-Through Rate by Position. Next, we consider another metric that is important from the firm's perspective: change in click-through rates. User satisfaction depends on the search engine's ability to serve relevant results at the top of a SERP. So an algorithm that increases CTRs for top positions is valuable. We, therefore, examine how position-based CTRs are expected to change with personalization compared with the current nonpersonalized ranking.

We consider two types of metrics within this context: (1) ΔCTR@p, change in CTR at position p, and (2) ΔHDCTR@p, change in high-relevance CTR (i.e., clicks with $r = 2$) at position p. Because we are solving a "reranking" or reordering problem, the total CTR or HDCTR for the entire list remains constant, and the sum of the changes in CTRs across all positions is zero (i.e., $\sum_{p=1}^{10} \Delta\mathrm{CTR@}p = 0$ and $\sum_{p=1}^{10} \Delta\mathrm{HDCTR@}p = 0$). However, with a better ranking algorithm, we should expect the documents to be reordered such that more relevant documents are displayed higher up. Thus, the CTRs for the topmost position should increase at the expense of lower positions.

Improvement in CTR and HDCTR: Figures 11 and 12 summarize our findings. The y-axis depicts the difference in CTRs (in percentage) after and before personalization, and the x-axis depicts the position. When we consider all clicks, CTR for position 1 increases by 3.5%, which is substantial. This increase mainly comes from positions 3 and 10. The patterns are similar for HDCTR with position 1 seeing an increase of more than 3.2%.

Figure 10. Example of AERC for Two Lists with the Same Set of Documents



Notes. The list on the left is unoptimized and has AERC = 1.5, whereas the right list is optimized (the documents are listed in the order of nonincreasing relevance) and has AERC = 0. For both lists, on the left, we show the current position of clicked documents and the optimal position on the right.

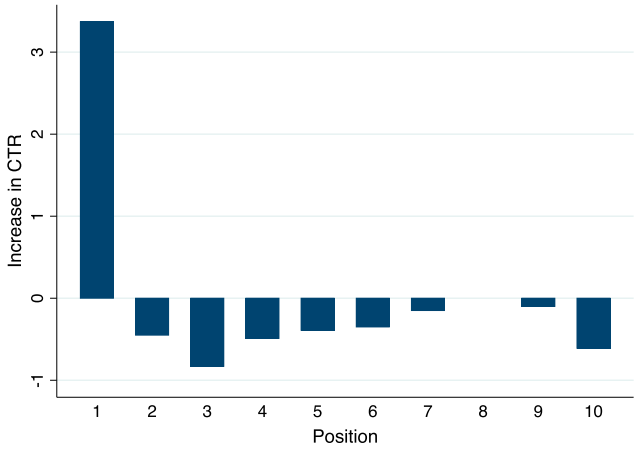
Overall, we find that deploying a personalization algorithm would lead to an increase in clicks for higher positions, thereby reducing the need for consumers to scroll further down the SERP for relevant results. This, in turn, can lead to higher user satisfaction and reduce switching, both of which are valuable outcomes from the search engine's perspective.

6.1.3. Change in NDCG. Model fit, as measured by our optimization metric NDCG, also improves with personalization. The baseline NDCG for the data is 0.7937. With personalized ranking, NDCG improves to 0.8121, which represents a 2.32% improvement over the baseline. Note that NDCG uses a log scale, which masks the full scale of improvement in rankings (which were more evident in the two metrics earlier).

Our overall improvement in NDCG is higher than that achieved by winning teams from the Kaggle contest (Kaggle 2014). This is in spite of the fact that the contestants used a longer data set for training their model.[14] We conjecture that this is due to our use of a comprehensive set of personalized features in conjunction with the parallelized LambdaMART algorithm that allows us to exploit the complete scale of the data. We refer interested readers to Masurel et al. (2014), Song (2014), and Volkovs (2014), the technical reports by the top three winning teams, which discuss their respective methodologies.

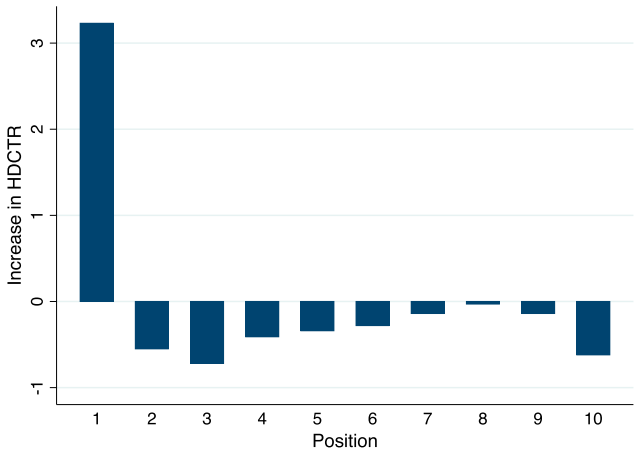
In sum, we find that personalization leads to a "better" ranking of documents—both on managerially relevant metrics, such as AERC, ΔCTR@p, and ΔHDCTR@p, and on optimization/model-fit measures, such as NDCG. Nevertheless, an important caveat here is that, if users react adversely to personalization (say because of privacy concerns) and reduce the extent to which they use the search engine and/or click on search results, the gains in the field can be lower than those documented in the paper. Of course, if users particularly like personalization and, therefore, choose to click more on the top results, the gains can be more positive than what we present here.

Figure 11. (Color online) Change in CTR as a Function of Position



Note. CTRs are expressed in percentages; therefore, ΔCTRs are also expressed in percentages.

Figure 12. (Color online) Change in HDCTR as a Function of Position



Note. HDCTRs are expressed in percentages; therefore, ΔHDCTRs are also expressed in percentages.

6.2. Value of Feature Classes

Next, we examine the source of the improvements discussed. Using a comparative analysis, we quantify the value of different types of features in improving search results. We start with the full model, then remove a specific category of features and evaluate the loss in NDCG prediction accuracy as a consequence of the removal. The percentage loss is defined as

$$PercentageLoss = \frac{NDCG_{full} - NDCG_{current}}{NDCG_{full} - NDCG_{base}} \times 100. \quad (23)$$

The results from this experiment are presented in Table 4. The first two rows show the NDCG for the full model and the baseline model.

We first discuss the value of the feature sets described in Section 4.2. Eliminating global features (F_G) results in a loss of 33.15% of the improvement that we gain from using the full feature set. This suggests that the interaction effects between global and personalized features plays an important role in improving the accuracy of our algorithm. Next, we find that eliminating user-level features results in a loss of 53.26%. Thus, more than 50% of the improvement comes from the personalized user history. In contrast, eliminating within session–level features while retaining across-session user-level features results in a loss of only 3.81%. This implies that across-session features are more valuable than within-session features and that user behavior is stable over longer time periods. Thus, search engines can forgo real-time, session-level

Table 4. Loss in Prediction Accuracy When Different Feature Sets Are Removed from the Full Model

	NDCG	% loss
Full set (all features)	0.8121	
All features except SERP	0.7937	100
Global features (F_G)	0.8060	33.15
User features (F_P)	0.8023	53.26
Session features (F_S)	0.8114	3.80
URL features (F_U)	0.8101	10.87
Domain features (F_D)	0.8098	12.50
Query features (F_Q)	0.8106	8.15
Term features (F_T)	0.8119	1.09
Click features	0.8068	28.80
Dwell features	0.8112	4.89
Position features	0.8116	2.72
Skip features	0.8118	1.63
Posn-order-click features	0.8120	0.54

personalization (which is costly in computing time and deployment) without too much loss in prediction accuracy.

The elimination of features based on the domain name and URL also results in a significant loss in prediction accuracy. This is because users prefer search results from domains and URLs that they trust. Search engines can, thus, track user preferences over domains and use this information to optimize search ranking. Query-specific features are also quite informative (8.15% loss in accuracy).

Next, in the last five rows of Table 4, we examine the value of feature sets based on the function used to generate them. Here, we find click features are the most valuable; there is a 28.8% loss in accuracy when we exclude them. Click features constitute all features derived from the functions *CTR*, *HDCTR*, *WCTR*, *PosnCTR*, *ClickProb*, *NumSearchResults*, and *NumClickedURLs*. Click features provide information on the users' past clicking and dwelling behavior. The high loss from excluding them suggests that users exhibit persistence in clicking behavior—across both positions (some users may only click on the first two positions, and others may click on multiple positions) as well as URLs and domains. (For a given query, some users may exhibit persistent preference for clicking on certain URLs; e.g., some users might be using a query for navigational purposes, and others might be using it to get information/news.) Overall, this suggests that history on past clicking behavior can inform us of these proclivities and help us improve the results shown to different users.

Within click features, dwell features, computed based on the dwell times associated with user clicks, *HDCTR* and *ADT* are the most effective (leading to a loss of 4.89% in NDCG when dropped). Position features (computed from functions *PosnCTR* and *AvgPosn*) are marginally useful. Finally, skip features, computed using the function *SkipRate*, and click features computed based on position and/or order of click (using functions *WCTR* and *PosnCTR*) are the least effective.

In sum, these findings give us insight into the relative value of different types of features. This can help businesses decide which types of information to incorporate into their search algorithms and which types of information to store for longer periods of time.

6.3. Robustness Checks and Scalability

6.3.1. Robustness Checks. We perform a series of tests and analyses to confirm the robustness of our findings. We discuss them briefly here and refer readers to Online Appendix C for details.

First, we consider the robustness of our results to the optimization metric used. We consider four other commonly used metrics in the learning to rank literature: DCG, MAP, reciprocal rank, and expected reciprocal rank. Online Appendix C.1 discusses these metrics in detail, and Table C.1 shows the percentage improvement in both AERC and NDCG on the test data compared with the baseline for models using each of these metrics. The reason we include AERC is that one could argue that models that didn't optimize NDCG will perform poorly when evaluated on NDCG. However, AERC is a substantive metric that none of the models directly optimizes and is, therefore, a more objective evaluation criterion. Overall, we find that the model using NDCG offers the best improvement. The other metrics, however, come close, suggesting that our substantive results are not driven by the choice of the metric.

Second, we examine whether other learning algorithms offer better performance compared with LambdaMART. We consider both standard econometric methods as well as other machine-learning models. We consider seven other methods: (1) a logit model that optimizes the pairwise maximum likelihood; (2) Ranknet, a pairwise algorithm that uses neural networks for training; (3) RankBoost, a pairwise algorithm that uses boosted regression trees; (4) AdaRank and (5) coordinate ascent, two other commonly used learning-to-rank algorithms; (6) LambdaRank with CART; and (7) LambdaRank with random forests. The last two use the same LambdaRank algorithm that we use, but the former uses only one tree, whereas the latter uses random forests to learn the underlying $f(\cdot)$. We find that LambdaMART offers the best performance compared with all these models when compared on percentage gains in AERC and NDCG (see Table C.3 in the online appendix).

Third, we examine what happens if we assume that consumers only look at the top P positions, where $P < 10$. We train and test the model on positions $P = 3, 5, 8$ in addition to $P = 10$ and present the results in

Table C.4 in Online Appendix C.3. At a high level, we find that there are positive returns to personalization even if we constrain our analysis to only the top few positions. However, the extent of improvement is monotonically increasing in P because of two reasons. First, when we train only on the top P positions, we throw away the data (and information) in positions below P. Second, recall that personalization increases the CTR for position 1 by drawing from lower positions (see Figure 11). When we ignore positions below P, we forgo the possibility of migrating relevant documents from positions below P to the top P positions, which, in turn, reduces the returns to personalization. We refer readers to Online Appendix C.3 for a more detailed discussion.

6.3.2. Scalability. Finally, we consider the scalability of our framework. For the full data set, feature generation takes about 83 minutes on a single-core computer, training takes more than 11 days, and testing takes only 35 minutes. These numbers suggest that training is the primary bottleneck in implementing the algorithm, especially if the search engine wants to update the trained model on a regular basis. So we parallelize the training module over a 32-core Amazon EC2 server. This reduces the training time to about 8.7 hours. Overall, with sufficient parallelization, we find that the framework can be scaled to large data sets in reasonable time frames. Please see Online Appendix D for a detailed discussion. In Section 8, we focus on improving the scalability further using feature selection.

7. Heterogeneity in Returns to Personalization

We now examine the role of two types of heterogeneity in returns to personalization:

- User-level heterogeneity: In Section 7.1, we examine how the returns to personalization vary as a function of user history.
- Query-level heterogeneity: In Section 7.2, we examine how the returns to personalization vary with the type of the query used.

In both these sections, we quantify the returns to personalization as the percentage change in AERC, the substantive metric used to evaluate search quality defined in Section 6.1.1. The qualitative results remain the same if we instead show the results in terms of NDCG improvements.

7.1. Value of Users' Search History

What is the value of user history data in improving search quality? Firms are struggling with the decision of how much history to store, especially because storage of massive personal data sets can be quite costly. Moreover, there are privacy costs associated with storing and using user-level data. Although these are less tangible, they are nevertheless important to the firm's long-run relationship with its customers. By quantifying the improvement in search results for each additional piece of past history, we can provide firms with tools to decide whether the costs associated with storing longer histories are offset by the improvements in consumer experience.

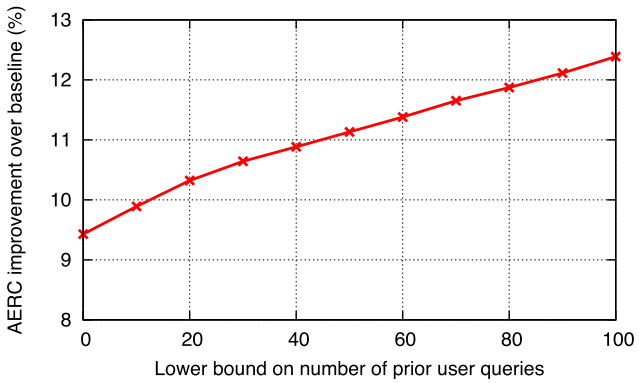
Even though we have less than 26 days of history, on average, the median query (by a user) in the test data set has 49 prior queries, and the 75th percentile has 100 prior queries.[15] We find that this accumulated history pays significant dividends in improving search quality. In Figure 13, when we set the lower bound for past queries to zero, we get the standard 9.43% improvement over the baseline as shown by the point on the y-axis. When we restrict the test data to only contain queries for which the user has issued at least 50 past queries (which is close to the median), we find that the AERC improves by more than 11%. When we increase the lower bound to 100 queries (the 75th percentile for our data), we see an even larger improvement in AERC (nearly 12.5% over the baseline).

In Figure 14, we present a similar analysis with upper bounds on the length of prior user history. When we set the upper bound on the number of past queries to 10, personalization leads to a 6.68% increase in AERC. This suggests that even small histories lead to significant improvements in personalized search quality. When we increase the upper bound to 50 queries, AERC improves by nearly 8%. Thus, similar to the lower-bound analysis, there is a monotonic increase in AERC as a function of the length of search history. This reaffirms the value of storing and using personalized browsing histories to improve users' search experience.

7.2. Query-Level Heterogeneity and Returns to Personalization

We consider two types of query-level heterogeneity: (1) heterogeneity in the intent of the query (as quantified

Figure 13. (Color online) Percentage Improvement in AERC for Different Lower Bounds on the Number of Prior Searches by the User



by the number of unique URLs clicked for a query) and (2) heterogeneity in the past performance of the query.

7.2.1. Impact of Query Intent on Personalization. Queries can be classified based on the number of unique URLs that are clicked on in response to a query. For example, if all users who search for "cnn" always click on cnn.com, then the number of unique URL clicks for this query would be one. Instead, if we observe clicks to cnn.com and as well the Wikipedia page on CNN, then the number of unique URLs clicked would be two. We now examine whether the returns to personalization vary across queries with different numbers of unique URLs clicked. For each search query in the test data, we first calculate the number of unique URL clicks that it received in the global data (from days 1–25). We then classify queries based on "number of unique prior clicks" and plot the gains in AERC for queries by this metric in Figure 15.

We find queries that have a larger number of unique URLs clicked on benefit more from personalization. When we restrict the data to queries that received clicks to at least one URL in the past, AERC improves by more than 14%.[16] As we increase the lower bound on the number of unique prior URL clicks, AERC increases monotonically, suggesting personalization. In fact, queries that receive clicks to 10 or more unique URLs see more than 18% increase in AERC. These findings are particularly interesting because the number of unique URL clicks metric reveals useful information about the type of query as described.

Search queries can be classified into three groups based on user intent: transactional, informational, and navigational (Jansen et al. 2008). In transactional queries, the user is looking to do something (e.g., "buy tea"); in informational queries, the user is looking to find information on a topic (e.g., "tea-growing regions"); and in navigational queries, the user is using the search engine to navigate to a website instead of typing in the URL directly (e.g., "numi tea"). Informally, this classification system is known as do-know-go. Because queries are anonymized in our data, we cannot directly classify them semantically. However, the number of unique URL clicks received metric can be interpreted as a measure of the extent to which a query is navigational, transactional, or informational—queries that receive clicks to just one unique URL or a few unique URLs are more likely to be navigational (e.g., "cnn"), transactional queries are likely to have a slightly larger set of unique URL clicks, and informational queries are likely to have the highest number of unique URLs clicked because users are often looking for different types of information even if they use the same keywords (as in the "java" example).

We, thus, find that navigational queries benefit the least from personalization, whereas informational queries benefit the most. More broadly, our results suggest that the search engine may want to weigh the costs and benefits of personalization through the lens of user intent.[17]

Figure 14. (Color online) Percentage Improvement in AERC for Different Upper Bounds on the Number of Prior Searches by the User

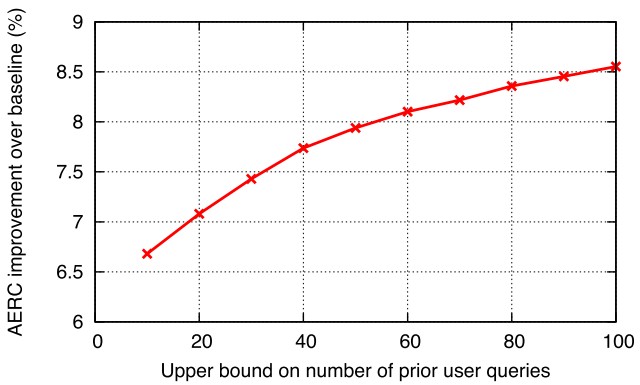
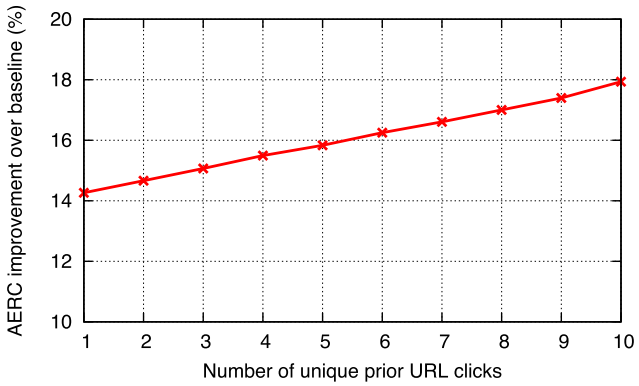


Figure 15. (Color online) Percentage Improvement in AERC for Different Lower Bounds on the Number of Prior Unique URL Clicks (in the Global Data) for the Query Used



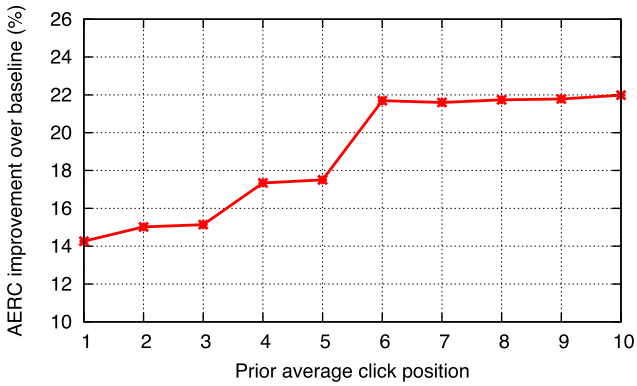
7.2.2. Impact of Query's Past Performance on Personalization. Queries can also be categorized based on their prior performance. For instance, queries that, on average, receive clicks to top-ranked documents can be interpreted as well-performing queries, whereas queries that receive clicks to lower-ranked documents are not well optimized. In this section, we examine the returns to personalization as a function of the query's past performance.

For each search query in the test data, we define the prior average click position as the average position of its clicks in the global data (from days 1 to 25). For example, if "buy tea" was searched for twice in the global data and it received two clicks at positions 1 and 2, respectively, this metric is 1.5. This metric quantifies the prior performance of the query.

We find that the value of personalization as a function of the query's past performance is initially increasing and then constant (see Figure 16). When we set the lower bound on the prior average click position to one, we find that AERC improves by more than 14%.[18] This number continues to increase until the prior average click position reaches six, at which point the AERC improvement is a substantial 22%. After that, for queries with a prior average click position of six or lower, the AERC improvement hovers around 22%.

In sum, we find that personalization, although generally useful, has considerable value for a subset of queries: informational queries and queries that are currently performing poorly. Therefore, even if personalization is too costly to apply broadly, the search engine can benefit by selectively personalizing certain subsets of searches.

Figure 16. (Color online) Percentage Improvement in AERC for Different Lower Bounds on Prior Average Click Position (in the Global Data) of the Query Used



8. Feature Selection

We now discuss the third and final step of our machine-learning approach. As discussed in Section 6.3.2, even when parallelized on 32-core servers, personalization algorithms can be very computing-intensive. Therefore, we now examine feature selection, by which we optimally choose a subset of features that maximize the predictive accuracy of the algorithm with minimal loss in accuracy. We start by describing the background literature and frameworks for feature selection and then discuss its application to our context, followed by the results.

Automatic feature selection is often used in supervised learning models because of two reasons (Guyon and Elisseeff 2003). First, it can provide a faster and cost-effective, or a more *efficient*, model by eliminating less-relevant features with minimal loss in accuracy. Efficiency is particularly relevant when training large data sets such as ours. Second, it provides more comprehensible models that offer a better understanding of the underlying data-generating process.[19]

The goal of feature selection is to find the smallest set of features that can provide a fixed predictive accuracy. In principle, this is a straightforward problem because it simply involves an exhaustive search of the feature space. However, even with a moderately large number of features, this is practically impossible. With F features, an exhaustive search requires 2^F runs of the algorithm, which is exponentially increasing in F. In fact, this problem is known to be NP-hard (Amaldi and Kann 1998).

8.1. Wrapper Approach

The wrapper method addresses this problem by using a greedy algorithm. We describe it briefly here and refer interested readers to Kohavi and John (1997) for a detailed review. Wrappers can be categorized into two types: *forward selection* and *backward elimination*. In forward selection, features are progressively added until a desired prediction accuracy is reached or until the incremental improvement is very small. In contrast, a backward-elimination wrapper starts with all the features and sequentially eliminates the least valuable features. Both these wrappers are greedy in the sense that they do not revisit former decisions to include (in forward selection) or exclude features (in backward elimination). In our implementation, we employ a forward-selection wrapper.

To enable a wrapper algorithm, we also need to specify a selection as well as a stopping rule. We use the robust *best-first* selection rule (Ginsberg 1993), wherein the most promising node is selected at every decision point. For example, in a forward-selection algorithm with 10 features, at the first node, this algorithm considers 10 versions of the model (each with one of the features added) and then picks the feature whose addition offers the highest prediction accuracy. Further, we use the following stopping rule, which is quite conservative. Let $NDCG_{base}$ be the NDCG from the base model currently used by Yandex, which only uses *SERPRank*. Similarly, let $NDCG_{current}$ be the improved NDCG at the current iteration and $NDCG_{full}$ be the improvement that can be obtained from using a model with all the 293 features. Then our stopping rule is

$$PercentageLoss = \frac{NDCG_{current} - NDCG_{base}}{NDCG_{full} - NDCG_{base}} > 0.95. \tag{24}$$

According to this stopping rule, the wrapper algorithm stops adding features when it achieves 95% of the improvement offered by the full model.

Wrappers offer many advantages. First, they are agnostic to the underlying learning algorithm and the accuracy metric used for evaluating the predictor. Second, greedy wrappers have been shown to be

Table 5. Set of Features Added at Each Iteration of the Wrapper Algorithm and the Corresponding NDCGs

Iteration number	Number of features	Feature class added at iteration	NDCG
0	1	*SERPRank*	0.7937
1	9	*User–Query–URL*	0.8033
2	17	*Global–Query–URL*	0.8070
3	24	*User–Null–Domain*	0.8094
4	32	*Global–Null–URL*	0.8104
5	40	*User-session–Null–URL*	0.8108
6	47	*Global–Null–Domain*	0.8112

computationally advantageous and robust against overfitting (Reunanen 2003).

Wrappers are greedy algorithms that offer computational advantages. They reduce the number of runs of the training algorithm from 2^F to $\frac{F(F+1)}{2}$. Although this is significantly less than 2^F, it is still prohibitively expensive for large feature sets. For instance, in our case, this would amount to 42,778 runs of the training algorithm, which is not feasible. Hence, we use a *coarse wrapper* that runs on sets or classes of features as opposed to individual features. We use domain-specific knowledge to identify groups of features and perform the search process on these groups of features as opposed to individual ones. This allows us to limit the number of iterations required for the search process to a reasonable number.

8.2. Feature Classes for Wrapper

Recall that we have 293 features. We classify these into 39 classes as follows:

First, we use the 36 classes of features generated by the functional inputs θ_1, θ_2, and θ_3. (Recall that $\text{Dim}(\theta_1) \times \text{Dim}(\theta_2) \times \text{Dim}(\theta_3) = 6 \times 2 \times 3 = 36$.) The seven functions—*Listings*, *AvgPosn*, *CTR*, *HDCTR*, *WCTR*, *PosnCTR*, and *ADT*—generate $7 \times 36 = 252$ features, and *SkipRate* contributes 18 features (because it is only specified at the URL level). Thus, 36 classes cover the first 270 features listed in Table A.1 in the online appendix.

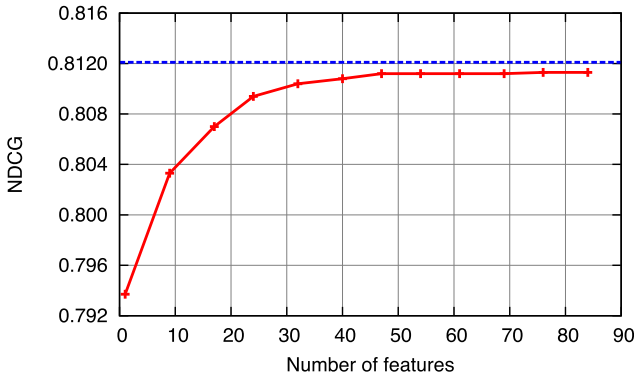
Next, we classify features 271–278, which consist of *AggListings*, *AggCTR*, *AggHDCTR*, and *AggADT* at user and session levels, aggregated over all URLs and domains, into the 37th class. The 38th class consists of user-specific click probabilities for all 10 positions, aggregated over the user history (features 279–288 in Table A.1 in the online appendix). The remaining features 289–292 form the 39th class. Note that *SERPRank* is not classified under any class because it is included in all models by default.

8.3. Results: Feature Selection

We use a search process that starts with a minimal singleton feature set that comprises just the *SERPRank* feature. This produces the baseline NDCG of 0.7937. Then we sequentially explore the feature classes using the first-best selection rule. Table 5 shows the results of this exploration process along with the class of features added in each iteration. We stop at iteration six (by which time, we have run the model 219 times), when the improvement in NDCG satisfies the condition laid out in Equation (24). The improvement in NDCG at each iteration is also shown in Figure 17 along with the maximum NDCG obtainable with a model that includes all the features. Although the condition for the stopping rule is satisfied at iteration 6, we show the results until iteration 11 in Figure 17 to demonstrate the fact that the improvement in NDCG after iteration 6 is minimal. Note that at iteration six, the pruned model with 47 features produces an NDCG of 0.8112, which equals 95% of the improvement offered by the full model. This small loss in accuracy brings a huge benefit in scalability as shown in Figures 18 and 19. When using the entire data set, the training time for the pruned model is ≈15% of the time it takes to train the full model. Moreover, the pruned model scales to larger data sets much better than the full model. For instance, the difference in training times for the full and pruned models is small when performed on 20% of the data set. However, as the data size grows, the training time for the full model increases more steeply than that for the pruned model. Although training time is the main time sink in this application, training is only done once. In contrast, testing runtimes have more consequences for the model's applicability because testing/prediction has to be done in real time for most applications. As with training times, we find that the pruned model runs significantly faster on the full data set (≈ 50%) as well as scaling better with larger data sets. Recall that we have 800,167 data points in the test data and the testing time for the full data is slightly more than 1,000 seconds. So, from an individual consumer's perspective, personalization leads to a mere 1.2-millisecond delay in seeing search results. This attests to the real-time scalability of our personalization framework.

Together, these findings suggest that feature selection is a valuable and important step in reducing runtimes and making personalization implementable in real-time applications for large data sets.

Figure 17. (Color online) Improvement in NDCG at Each Iteration of the Wrapper Algorithm



Note. The tick marks in the solid line refer to the iteration number, and the dotted line depicts the maximum NDCG obtained with all the features included.

9. Conclusion and Future Research

Personalization of digital services remains the holy grail of marketing. In this paper, we study an important personalization problem: that of search rankings in the context of online search.

We present a three-pronged machine-learning framework that improves the ranking of search results by incorporating a users' personal search history. We apply our framework to data from the premier Eastern European search engine, Yandex, and provide evidence of substantial improvements in search quality using personalization. We quantify the heterogeneity in returns to personalization as function of user history, query type (do-know-go), and query past performance. We also show that our framework can perform efficiently at scale, making it suitable for real-time deployment.

Our paper makes five key contributions to the marketing literature. First, it presents a general machine-learning framework that marketers can use to rank recommendations using personalized data in many settings. Second, it presents empirical evidence in support of the returns to personalization in the online search context. Third, it provides managerial insights on the role of heterogeneity in user history and query type on the returns to personalization. Fourth, it demonstrates how Big Data can be leveraged to improve marketing outcomes without compromising the speed or real-time performance of digital applications. Finally, it provides insights on how machine-learning methods can be adapted to solve important marketing problems that have been technically unsolvable so far (using traditional econometric or analytical models).

Figure 18. (Color online) Runtimes for Training the Full and Pruned Models on Different Percentages of the Data

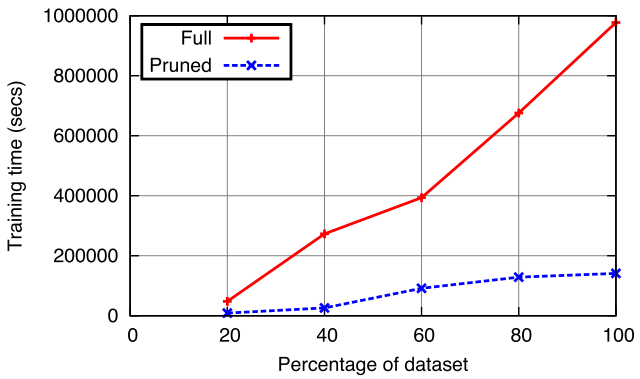
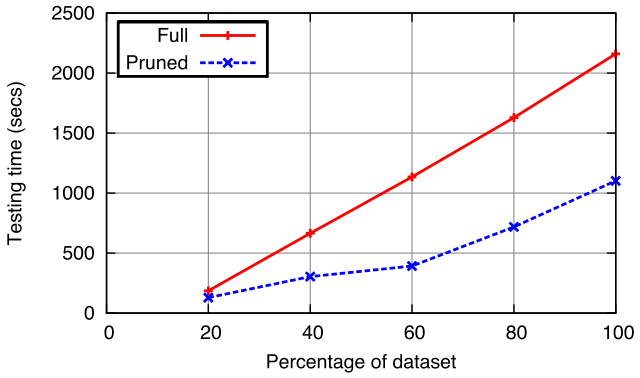


Figure 19. (Color online) Runtimes for Testing the Full and Pruned Models on Different Percentages of the Data



Nevertheless, our paper overlooks a bunch of issues that serve as excellent avenues for future research. First, the use of individual-level user data (albeit anonymized) raises questions regarding privacy. It is not clear how users trade off the loss in privacy with the gains from an improved search experience. Experimental or conjoint research that measures user preferences over these experiences can help managers decide the implications of deploying personalization algorithms. Second, our model's predictions are predicated on the assumption that consumers' click and search behaviors will continue to be the same even after we deploy personalized algorithms. However, this may not be the case. For example, Goldfarb and Tucker (2011) show that consumers alter their response rates when advertisements are perceived to intrude on privacy. Third, because we have not deployed the algorithm in the field, we cannot comment on long-term consumer satisfaction metrics and switching behavior. These limitations can be addressed by running a large-scale field experiment that shows personalized results to a sample of users and compares their behavior with that of users who are shown non-personalized results. We believe that such a study would be of immense value to the field. More broadly, we hope our work will spur research on the application of machine-learning methods to not only personalization-related issues, but also on a broad array of marketing issues.

Acknowledgments

The author thanks Yandex for providing the data used in this paper and Preyas Desai, Daria Dzyabura, Brett Gordon, Xiao Liu, Lan Luo, and Olivier Toubia for detailed comments on the paper. The author also thanks the participants of the following conferences: University of Texas at Dallas Frontiers of Research in Marketing Science (FORMS) 2014, Marketing Science 2014, Big Data and Marketing Analytics 2014, and Stanford Digital Marketing 2016; as well as the participants of the following seminars: University of Washington Marketing Camp 2015, Harvard Business School Marketing Seminar 2017, and Duke Marketing Seminar 2017, for constructive feedback.

Endnotes

[1] Google launched personalized search a few years back, customizing results based on a user's IP address and cookies (individual-level search and click data), using up to 180 days of history (Google Official Blog 2009). Bing and Yandex have also experimented with personalization (Sullivan 2011, Yandex 2013).

[2] The exact dollar amount can vary with the exchange rate. Yandex reported a 05.5 million revenue for Q3 in 2016 based on the exchange rates from October 27^{th} 2016.

[3] Queries are not the same as keywords, which is another term used frequently in connection with search engines. Please see Gabbert (2011) for a simple explanation of the difference between the two phrases.

[4] Search engines typically ignore common keywords and prepositions, such as "of'," "a," "in," used in searches because they tend to be uninformative. Such words are called "stop words."

[5] As explained in Section 3.3, users observed early in the data have no/very little past history because we have a data snapshot for fixed time period (27 days) and not a fixed history for each user. Hence, following Yandex's suggestion, we run the personalization algorithm for users observed in the last two days. In practice, of course, firms don't have to work with snapshots of data; rather, they are likely to have longer histories stored at the user level.

[6] K-fold cross-validation is another method to split data and perform model selection. (See Steckel and Vanhonacker (1993) for an example in marketing.) However, it suffers from two key drawbacks. First, as K increases, it becomes very computationally expensive. Second, because a given data point is used for both training and testing, the risk of overfitting is high. When a cross-validated model is tested on a completely new sample, it usually underperforms. So a three-way split is the right approach in data-rich situations, whereas K-fold cross-validation is better when the data are moderate/small. Because our data set is very large, we use the first-best approach of the three-way split. We refer interested readers to chapter 7 in Hastie et al. (2009) for a detailed discussion of this issue.

[7] We observe a total of 1,174,096 users in days 26–27, who participate in 2,327,766 sessions. Thus, each data set (training, validation, and testing) has approximately one third of these users and sessions in it.

[8] Although it is possible to consider a larger number of terms, we stop with four for a few reasons. First, more than 81.59% of the queries have four or fewer terms. So the first four terms are sufficient to capture most term-specific information in the data (see Table 2). Second, this allows us to keep the size of the feature set reasonable. Third, and most importantly, when we experimented with higher-order terms, we did not find any improvement in the model fit or results.

[9] In standard marketing models, ws are treated as structural parameters that define the utility associated with an option. So their directionality and magnitude are of interest to researchers. However, in this case, there is no structural interpretation of w. In fact, with complex trainers, such as boosted regression trees, the number of parameters is large and difficult to interpret.

[10] Versions of RankNet are used by commercial search engines, such as Bing (Liu 2011).

[11] Asymptotically, C_{ijk}^{qmn} becomes linear if the scores give the wrong ranking and zero if they give the correct ranking.

[12] This issue also exists for other discrete optimization metrics, such as DCG and MAP that we consider in Online Appendix C.1.

[13] MART (multiple additive regression trees) is a trademarked implementation of stochastic gradient-boosted trees by Salford Systems. However, the acronym is often used loosely to refer to any implementation of gradient-boosted trees as in the case of LambdaMART.

[14] The contestants used all the data from days 1–27 for building their models and had their models tested by Yandex on an unmarked data set (with no information on clicks made by users) from days 28–30. In contrast, we had to resort to using the data set from days 1–27 for both model building and testing. So the amount of past history we have for each user in the test data are smaller, putting us at a relative disadvantage (especially because we find that longer histories lead to higher NDCGs). Thus, it is noteworthy that our improvement is higher than that achieved by the winning teams.

[15] The number of past queries for a median query (50) is a different metric than the one discussed earlier, median of the number of queries per user (shown in Figure 4 as 16), for two reasons. First, this number is calculated only for queries that have one or more prior queries. Second, not all users submit the same number of queries. Indeed, a small subset of users account for most queries. So the average query has a larger median history than the average user.

[16] This number is different from 9.43% for the entire test data because it excludes queries that received no clicks in the global data and those that are new to the test data and, therefore, have zero unique prior URL clicks by default.

[17] We do not find any significant correlation between the query-level metrics that we consider: prior average click position and prior unique URLs clicked. When we regress the improvement in AERC on the two query-level metrics, both of them continue to have a significant and positive impact.

[18] This is different from 9.43% for the entire test data because it excludes queries that received no clicks in the global data and those that are new to the test data and, therefore, have been assigned a prior average click position of zero by default.

[19] In cases in which the data sets are small and the number of features large, feature selection can actually improve the predictive accuracy of the model by eliminating irrelevant features whose inclusion often results in overfitting. Many machine-learning algorithms, including neural networks, decision trees, CART, and naive Bayes learners have been shown to have significantly worse accuracy when trained on small data sets with superfluous features (Duda and Hart 1973, Breiman et al. 1984, Aha et al. 1991, Quinlan 1993). In our setting, we have an extremely large data set, which prevents the algorithm from making spurious inferences in the first place. So feature selection is not accuracy improving in our case; however, it leads to significantly lower computing times.

References

Aha DW, Kibler D, Albert MK (1991) Instance-based learning algorithms. *Machine Learning* 6(1):37–66.

Amaldi E, Kann V (1998) On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems. *Theoretical Comput. Sci.* 209(1):237–260.

Ansari A, Mela CF (2003) E-customization. *J. Marketing Res.* 40(2):131–145.

Arora N, Henderson T (2007) Embedded premium promotion: Why it works and how to make it more effective. *Marketing Sci.* 26(4): 514–531.

Banerjee S, Chakrabarti S, Ramakrishnan G (2009) Learning to rank for quantity consensus queries. Allan J, Aslam J, Sanderson M,

Zhai C, Zobel J, eds. *Proc. 32nd Internat. ACM SIGIR Conf. Res. Development Inform. Retrieval* (ACM, New York), 243–250.

Bennett PN, White RW, Chu W, Dumais ST, Bailey P, Borisyuk F, Cui X (2012) Modeling the impact of short- and long-term behavior on search personalization. Hersh W, Callan J, Maarek Y, Sanderson M, eds. *Proc. 35th Internat. ACM SIGIR Conf. Res. Development Inform. Retrieval* (ACM, New York), 185–194.

Breiman L (1998) Arcing classifier. *Ann. Statist.* 26(3):801–849.

Breiman L, Friedman J, Stone C, Olshen R (1984) *Classification and Regression Trees*, The Wadsworth and Brooks-Cole Statistics-Probability Series (Springer, Cham, Switzerland).

Burges CJ, Ragno R, Le QV (2006) Learning to rank with nonsmooth cost functions. *Adv. Neural Inform. Processing Systems (NIPS)*, vol. 19 (Curran Associates, Red Hook, NY), 193–200.

Burges C, Shaked T, Renshaw E, Lazier A, Deeds M, Hamilton N, Hullender G (2005) Learning to rank using gradient descent. De Raedt L, Wrobel S, eds. *Proc. 22nd Internat. Conf. Machine Learn.* (ACM, New York), 89–96.

Cao Z, Qin T, Liu T-Y, Tsai M-F, Li H (2007) Learning to rank: From pairwise approach to listwise approach. Ghahramani Z, ed. *Proc. 24th Internat. Conf. Machine Learn.* (ACM, New York), 129–136.

Caruana R, Niculescu-Mizil A (2006) An empirical comparison of supervised learning algorithms. Cohen W, Moore A, eds. *Proc. 23rd Internat. Conf. Machine Learn.* (ACM, New York), 161–168.

Chapelle O, Chang Y (2011) Yahoo! Learning to rank challenge overview. *J. Machine Learn. Res. Proc. Track* 14:1–24.

Chen T, Guestrin C (2016) Xgboost: A scalable tree boosting system. Aggarwal C, Smola A, Rastogi R, Shen D, Krishnapuram B, Shah M, eds. *Proc. 22nd ACM Sigkdd Internat. Conf. Knowledge Discovery Data Mining* (ACM, New York), 785–794.

Ciaramita M, Murdock V, Plachouras V (2008) Online learning from click data for sponsored search. Huai J, Chen R, Hon S-W, Liu Y, Ma W-Y, Tomkins S, Zhang X, eds. *Proc. 17th Internat. Conf. World Wide Web* (ACM, New York), 227–236.

Clayton N (2013) Yandex overtakes Bing as world's fourth search engine. Accessed February 25, 2019, http://blogs.wsj.com/tech-europe/2013/02/11/yandex-overtakes-bing-as-worlds-fourth-search-engine/.

Craswell N, Zoeter O, Taylor M, Ramsey B (2008) An experimental comparison of click position-bias models. Najork M, Broder A, Chakrabarti S, eds. *Proc. 2008 Internat. Conf. Web Search Data Mining* (ACM, New York), 87–94.

Dang V (2011) RankLib. Online. Accessed March 14, 2019, https://sourceforge.net/p/lemur/wiki/RankLib/.

De los Santos B, Hortaçsu A, Wildenbeest MR (2012) Testing models of consumer search using data on web browsing and purchasing behavior. *Amer. Econom. Rev.* 102(6):2955–2980.

De Vrieze PT (2006) Fundamentals of adaptive personalisation. Dissertation, Netherlands Research School for Information and Knowledge Systems, Utrecht, Netherlands.

Donmez P, Svore KM, Burges CJ (2009) On the local optimality of LambdaRank. Allan J, Aslam J, Sanderson M, Zhai C, Zobel J, eds. *Proc. 32nd Internat. ACM SIGIR Conf. Res. Development Inform. Retrieval* (ACM, New York), 460–467.

Duda RO, Hart PE (1973) *Pattern Recognition and Scene Analysis*, vol. 3 (Wiley, New York).

Dzyabura D, Hauser JR (2011) Active machine learning for consideration heuristics. *Marketing Sci.* 30(5):801–819.

Dzyabura D, Yoganarasimhan H (2018) Machine learning and marketing. Mizik N, Hanssens DM, eds. *Handbook of Marketing Analytics* (Edward Elgar Publishing, Northampton, MA), 255–279.

Eickhoff C, Collins-Thompson K, Bennett PN, Dumais S (2013) Personalizing atypical web search sessions. Leonardi S, Panconesi A, Ferragina P, Gionis A, eds. *Proc. Sixth ACM Internat. Conf. Web Search Data Mining* (ACM, New York), 285–294.

Elith J, Leathwick JR, Hastie T (2008) A working guide to boosted regression trees. *J. Animal Ecology* 77(4):802–813.

Freund Y (1995) Boosting a weak learning algorithm by majority. *Inform. Comput.* 121(2):256–285.

Freund Y, Schapire RE (1996) Experiments with a new boosting algorithm. Saitta L, ed. *ICML '96 Proc. 13th Internat. Conf. Machine Learn.* (Morgan Kaufmann Publishers, San Francisco), 148–156.

Friedman JH (2001) Greedy function approximation: A gradient boosting machine. *Ann. Statist.* 29(5):1189–1232.

Gabbert E (2011) Keywords vs. search queries: What's the difference? Accessed February 25, 2019, http://www.wordstream.com/blog/ws/2011/05/25/keywords-vs-search-queries.

Ghose A, Ipeirotis PG, Li B (2014) Examining the impact of ranking on consumer behavior and search engine revenue. *Management Sci* 60(7):1632–1654.

Ginsberg M (1993) *Essentials of Artificial Intelligence* (Morgan Kaufmann Publishers, San Francisco).

Göker A, He D (2000) Analysing web search logs to determine session boundaries for user-oriented learning. Brusilovski P, Stock O, Strapparava C, eds. *Adaptive Hypermedia and Adaptive Web-Based Systems* (Springer-Verlag, Berlin, Heidelberg), 319–322.

Goldfarb A, Tucker C (2011) Online display advertising: Targeting and obtrusiveness. *Marketing Sci.* 30(3):389–404.

Google (2018) How search works. Accessed February 25, 2019, https://www.google.com/search/howsearchworks/crawling-indexing/.

Google Official Blog (2009) Personalized search for everyone. Accessed February 25, 2019, http://googleblog.blogspot.com/2009/12/personalized-search-for-everyone .html.

Guyon I, Elisseeff A (2003) An introduction to variable and feature selection. *J. Machine Learn. Res.* 3:1157–1182.

Hannak A, Sapiezynski P, Molavi Kakhki A, Krishnamurthy B, Lazer D, Mislove A, Wilson C (2013) Measuring personalization of web search. Schwabe D, Almeida V, Glaser H, Baeza-Yates R, Moon S, eds. *Proc. 22nd Internat. Conf. World Wide Web* (ACM, New York), 527–538.

Harrington EF (2003) Online ranking/collaborative filtering using the perceptron algorithm. Fawcett T, Mishra N, eds. *ICML '03 Proc. 20th Internat. Conf. Machine Learn.* (AAAI Press, Palo Alto, CA), 250–257.

Hastie T, Tibshirani R, Friedman J, Hastie T, Friedman J, Tibshirani R (2009) *The Elements of Statistical Learning*, vol. 2 (Springer-Verlag, New York).

Huang D, Luo L (2015) Consumer preference elicitation of complex products using fuzzy support vector machine active learning. *Marketing Sci* 35(3):445–464.

Jansen BJ, Booth DL, Spink A (2008) Determining the informational, navigational, and transactional intent of web queries. *Inform. Processing Management* 44(3):1251–1266.

Järvelin K, Kekäläinen J (2000) IR evaluation methods for retrieving highly relevant documents. *Proc. 23rd Annual Internat. ACM SIGIR Conf. Res. Development Inform. Retrieval, SIGIR '00* (ACM, New York), 41–48.

Järvelin K, Kekäläinen J (2002) Cumulated gain-based evaluation of IR techniques. *ACM Trans. Inform. Systems* 20(4):422–446.

Kaggle (2013) Personalized web search challenge. Accessed February 2, 2019, http://www.kaggle.com/c/yandex-personalized-web-search-challenge.

Kaggle (2014) Private leaderboard—Personalized web search challenge. Accessed February 2, 2019, https://www.kaggle.com/c/yandex-personalized-web-search-challenge/leaderboard.

Kohavi R, John GH (1997) Wrappers for feature subset selection. *Artificial Intelligence* 97(1):273–324.

Koulayev S (2014) Search for differentiated products: Identification and estimation. *RAND J. Econom.* 45(3):553–575.

Lambrecht A, Tucker C (2013) When does retargeting work? Information specificity in online advertising. *J. Marketing Res.* 50(5):561–576.

Lemmens A, Croux C (2006) Bagging and boosting classification trees to predict churn. *J. Marketing Res.* 43(2):276–286.

Liu T-Y (2009) Learning to rank for information retrieval. *Foundations Trends Inform. Retrieval* 3(3):225–331.

Liu TY (2011) *Learning to Rank for Information Retrieval* (Springer-Verlag, Berlin, Heidelberg).

Masurel P, Lefèvre-Hasegawa K, Bourguignat C, Scordia M (2014) Dataiku's solution to Yandex's personalized web search challenge. Technical report, Dataiku, New York.

Metzler D, Kanungo T (2008) Machine learned sentence selection strategies for query-biased summarization. *SIGIR Learn. Rank Workshop* (ACM, New York), 40–47.

Mihalkova L, Mooney R (2009) Learning to disambiguate search queries from short sessions. Buntine W, Grobelnik M, Mladenić D, Shawe-Taylor J, eds. *Joint Eur. Conf. Machine Learn. Knowledge Discovery Databases* (Springer, Berlin, Heidelberg), 111–127.

Murphy KP (2012) *Machine Learning: A Probabilistic Perspective* (MIT Press, Cambridge, MA).

Narayanan S, Kalyanam K (2015) Position effects in search advertising and their moderators: A regression discontinuity approach. *Marketing Sci.* 34(3):388–407.

Neslin SA, Gupta S, Kamakura W, Lu J, Mason CH (2006) Defection detection: Measuring and understanding the predictive accuracy of customer churn models. *J. Marketing Res.* 43(2):204–211.

Netzer O, Feldman R, Goldenberg J, Fresko M (2012) Mine your own business: Market-structure surveillance through text-mining. *Marketing Sci.* 31(3):521–543.

Pavliva H (2013) Google-beater Yandex winning over Wall Street on ad view. Accessed March 14, 2019, http://www.bloomberg.com/news/2013-04-25/google-tamer-yandex-amasses-buy-ratings-russia-overnight.html.

Qin T, Liu T-Y, Xu J, Li H (2010) LETOR: A benchmark collection for research on learning to rank for information retrieval. *Inform. Retrieval* 13(4):346–374.

Qiu F, Cho J (2006) Automatic identification of user interest for personalized search. Carr L, De Roure D, Iyengar A, Goble C, Dahlin M, eds. *Proc. 15th Internat. Conf. World Wide Web* (ACM, New York), 727–736.

Quinlan JR (1993) *C4. 5: Programs for Machine Learning*, vol. 1 (Morgan Kaufmann Publishing, San Francisco).

Rafieian O, Yoganarasimhan H (2019) Targeting and privacy in mobile advertising. Working paper, University of Washington, Seattle.

Reunanen J (2003) Overfitting in making comparisons between variable selection methods. *J. Machine Learn. Res.* 3:1371–1382.

Ricci F, Rokach L, Shapira B (2011) Introduction to recommender systems handbook. Ricci F, Rokach L, Shapira B, Kantor PB, eds. *Recommender Systems Handbook* (Springer, New York), 1–35.

Rossi PE, McCulloch RE, Allenby GM (1996) The value of purchase history data in target marketing. *Marketing Sci.* 15(4):321–340.

Schapire RE (1990) The strength of weak learnability. *Machine Learn.* 5(2):197–227.

Schwartz B (2012) Google: Previous query used on 0.3% of searches. Accessed February 25, 2019, http://www.seroundtable.com/google-previous-search-15924.html.

ScrapeHero (2018) How many products does Amazon sell? January 2018. Accessed February 25, 2019, https://www.scrapehero.com/many-products-amazon-sell-january-2018/.

Song G (2014) Point-wise approach for Yandex personalized web search challenge. Technical report, The Institute of Electrical and Electronics Engineers, Piscataway, NJ.

Steckel JH, Vanhonacker WR (1993) Cross-validating regression models in marketing research. *Marketing Sci.* 12(4):415–427.

Sullivan D (2011) Bing results get localized & personalized. Accessed February 25, 2019, http://searchengineland.com/bing-results-get-localized-personalized-642 84.

Surdeanu M, Ciaramita M, Zaragoza H (2008) Learning to rank answers on large online QA collections. *Proc. 46th Annual Meeting Assoc. Comput. Linguistics, Columbus, Ohio*, 719–727.

Teevan J, Dumais ST, Liebling DJ (2008) To personalize or not to personalize: Modeling queries with variation in user intent. Chua T-S, Leong M-K, Myaeng SH, Oard DW, Sebastiani F, eds. *Proc. 31st Annual Internat. ACM SIGIR Conf. Res. Development Inform. Retrieval* (ACM, New York), 163–170.

Toubia O, Evgeniou T, Hauser J (2007) Optimization-based and machine-learning methods for conjoint analysis: Estimation and question design. Gustafsson A, Herrmann A, Huber F, eds. *Conjoint Measurement: Methods and Applications* (Springer-Verlag, Berlin, Heidelberg), 231.

Tran K (2017) Viewers find objectionable content on YouTube kids. *Business Insider* (November 7), https://www.businessinsider.com/viewers-find-objectionable-content-on-youtube-kids-2017-11.

Ursu RM (2018) The power of rankings: Quantifying the effect of rankings on online consumer search and purchase decisions. *Marketing Sci.* 37(4):530–552.

Volkovs M (2014) Context models for web search personalization. Technical report, University of Toronto, Toronto.

Wang Y, Wang L, Li Y, He D, Chen W, Liu T-Y (2013) A theoretical analysis of NDCG ranking measures. *PMLR* 30:25–54.

Weitzman ML (1979) Optimal search for the best alternative. *Econometrica* 47(3):641–54.

Wu Q, Burges CJ, Svore KM, Gao J (2010) Adapting boosting for information retrieval measures. *Inform. Retrieval* 13(3):254–270.

Yandex (2013) It may get really personal we have rolled out our second–generation personalised search program. Accessed February 2, 2019, http://company.yandex.com/press_center/blog/entry.xml?pid=20.

Yandex (2016) Yandex announces third quarter 2016 financial results. Accessed February 2, 2019, http://ir.yandex.com/releasedetail.cfm?ReleaseID=995776.

Yue Y, Burges C (2007) On using simultaneous perturbation stochastic approximation for IR measures, and the empirical optimality of LambdaRank. Working paper, Centre for The Initiation of Talent and Industrial Training (CITRA), Kuala Lumpur, Malaysia.

Zhang J, Wedel M (2009) The effectiveness of customized promotions in online and offline stores. *J. Marketing Res.* 46(2):190–206.