Web Appendix for "Search Personalization using Machine Learning"

Hema Yoganarasimhan [*]

University of Washington

A Table of Features

Table A.1: List of features.

Feature No.	Feature Name	Feature Classes
1	*Listings*(q, u, g)	F_G, F_Q, F_U
2	*CTR*(q, u, g)	F_G, F_Q, F_U
3	*ADT*(q, u, g)	F_G, F_Q, F_U
4	*AvgPosn*(q, u, g)	F_G, F_Q, F_U
5	*HDCTR*(q, u, g)	F_G, F_Q, F_U
6	*WCTR*(q, u, g)	F_G, F_Q, F_U
7	*PosnCTR*(q, u, g)	F_G, F_Q, F_U
8	*SkipRate*(q, u, g)	F_G, F_Q, F_U
9	*Listings*(q, d, g)	F_G, F_Q, F_D
10	*CTR*(q, d, g)	F_G, F_Q, F_D
11	*ADT*(q, d, g)	F_G, F_Q, F_D
12	*AvgPosn*(q, d, g)	F_G, F_Q, F_D
13	*HDCTR*(q, d, g)	F_G, F_Q, F_D
14	*WCTR*(q, d, g)	F_G, F_Q, F_D
15	*PosnCTR*(q, d, g)	F_G, F_Q, F_D
16-19	*Listings*$(l_{1..4}, u, g)$	F_G, F_T, F_U
20-23	*CTR*$(l_{1..4}, u, g)$	F_G, F_T, F_U
24-27	*ADT*$(l_{1..4}, u, g)$	F_G, F_T, F_U
28-31	*AvgPosn*$(l_{1..4}, u, g)$	F_G, F_T, F_U
32-35	*HDCTR*$(l_{1..4}, u, g)$	F_G, F_T, F_U
36-39	*WCTR*$(l_{1..4}, u, g)$	F_G, F_T, F_U
40-43	*PosnCTR*$(l_{1..4}, u, g)$	F_G, F_T, F_U
		Continued on next page

[*]Please address all correspondence to: hemay@uw.edu.

Table A.1 – continued from previous page

Feature No.	Feature Name	Feature Classes
44-47	*SkipRate*$(l_{1..4}, u, g)$	F_G, F_T, F_U
48-51	*Listings*$(l_{1..4}, d, g)$	F_G, F_T, F_D
52-55	*CTR*$(l_{1..4}, d, g)$	F_G, F_T, F_D
107-110	*HDCTR*$(l_{1..4}, u, it)$	F_P, F_T, F_U
111-114	*WCTR*$(l_{1..4}, u, it)$	F_P, F_T, F_U
115-118	*PosnCTR*$(l_{1..4}, u, it)$	F_P, F_T, F_U
119-122	*SkipRate*$(l_{1..4}, u, it)$	F_P, F_T, F_U
123-126	*Listings*$(l_{1..4}, d, it)$	F_P, F_T, F_D
127-130	*CTR*$(l_{1..4}, d, it)$	F_P, F_T, F_D
131-134	*ADT*$(l_{1..4}, d, it)$	F_P, F_T, F_D
135-138	*AvgPosn*$(l_{1..4}, d, it)$	F_P, F_T, F_D
139-142	*HDCTR*$(l_{1..4}, d, it)$	F_P, F_T, F_D
143-146	*WCTR*$(l_{1..4}, d, it)$	F_P, F_T, F_D
147-150	*PosnCTR*$(l_{1..4}, d, it)$	F_P, F_T, F_D
151	*Listings*(q, u, ijt)	F_S, F_Q, F_U
152	*CTR*(q, u, ijt)	F_S, F_Q, F_U
153	*ADT*(q, u, ijt)	F_S, F_Q, F_U
154	*AvgPosn*(q, u, ijt)	F_S, F_Q, F_U
155	*HDCTR*(q, u, ijt)	F_S, F_Q, F_U
156	*WCTR*(q, u, ijt)	F_S, F_Q, F_U
157	*PosnCTR*(q, u, ijt)	F_S, F_Q, F_U
158	*SkipRate*(q, u, ijt)	F_S, F_Q, F_U
159	*Listings*(q, d, ijt)	F_S, F_Q, F_D
160	*CTR*(q, d, ijt)	F_S, F_Q, F_D
161	*ADT*(q, d, ijt)	F_S, F_Q, F_D
162	*AvgPosn*(q, d, ijt)	F_S, F_Q, F_D
163	*HDCTR*(q, d, ijt)	F_S, F_Q, F_D
164	*WCTR*(q, d, ijt)	F_S, F_Q, F_D
165	*PosnCTR*(q, d, ijt)	F_S, F_Q, F_D
166-169	*Listings*$(l_{1..4}, u, ijt)$	F_S, F_T, F_U
170-173	*CTR*$(l_{1..4}, u, ijt)$	F_S, F_T, F_U
174-177	*ADT*$(l_{1..4}, u, ijt)$	F_S, F_T, F_U
178-181	*AvgPosn*$(l_{1..4}, u, ijt)$	F_S, F_T, F_U
182-185	*HDCTR*$(l_{1..4}, u, ijt)$	F_S, F_T, F_U
186-189	*WCTR*$(l_{1..4}, u, ijt)$	F_S, F_T, F_U
190-193	*PosnCTR*$(l_{1..4}, u, ijt)$	F_S, F_T, F_U
194-197	*SkipRate*$(l_{1..4}, u, ijt)$	F_S, F_T, F_U
198-201	*Listings*$(l_{1..4}, d, ijt)$	F_S, F_T, F_D
202-205	*CTR*$(l_{1..4}, d, ijt)$	F_S, F_T, F_D
206-209	*ADT*$(l_{1..4}, d, ijt)$	F_S, F_T, F_D

Continued on next page

Table A.1 – continued from previous page

Feature No.	Feature Name	Feature Classes
210-213	*AvgPosn*$(l_{1..4}, d, ijt)$	F_S, F_T, F_D
214-217	*HDCTR*$(l_{1..4}, d, ijt)$	F_S, F_T, F_D
218-221	*WCTR*$(l_{1..4}, d, ijt)$	F_S, F_T, F_D
222-225	*PosnCTR*$(l_{1..4}, d, ijt)$	F_S, F_T, F_D
226	*Listings*$(\emptyset, u, g)$	$F_G, \quad F_U$
227	*CTR*$(\emptyset, u, g)$	$F_G, \quad F_U$
228	*ADT*$(\emptyset, u, g)$	$F_G, \quad F_U$
229	*AvgPosn*$(\emptyset, u, g)$	$F_G, \quad F_U$
230	*HDCTR*$(\emptyset, u, g)$	$F_G, \quad F_U$
231	*WCTR*$(\emptyset, u, g)$	$F_G, \quad F_U$
232	*PosnCTR*$(\emptyset, u, g)$	$F_G, \quad F_U$
233	*SkipRate*$(\emptyset, u, g)$	$F_G, \quad F_U$
234	*Listings*$(\emptyset, d, g)$	$F_G, \quad F_D$
235	*CTR*$(\emptyset, d, g)$	$F_G, \quad F_D$
236	*ADT*$(\emptyset, d, g)$	$F_G, \quad F_D$
237	*AvgPosn*$(\emptyset, d, g)$	$F_G, \quad F_D$
238	*HDCTR*$(\emptyset, d, g)$	$F_G, \quad F_D$
239	*WCTR*$(\emptyset, d, g)$	$F_G, \quad F_D$
240	*PosnCTR*$(\emptyset, d, g)$	$F_G, \quad F_D$
241	*Listings*$(\emptyset, u, it)$	$F_P, \quad F_U$
242	*CTR*$(\emptyset, u, it)$	$F_P, \quad F_U$
243	*ADT*$(\emptyset, u, it)$	$F_P, \quad F_U$
244	*AvgPosn*$(\emptyset, u, it)$	$F_P, \quad F_U$
245	*HDCTR*$(\emptyset, u, it)$	$F_P, \quad F_U$
246	*WCTR*$(\emptyset, u, it)$	$F_P, \quad F_U$
247	*PosnCTR*$(\emptyset, u, it)$	$F_P, \quad F_U$
248	*SkipRate*$(\emptyset, u, it)$	$F_P, \quad F_U$
249	*Listings*$(\emptyset, d, it)$	$F_P, \quad F_D$
250	*CTR*$(\emptyset, d, it)$	$F_P, \quad F_D$
251	*ADT*$(\emptyset, d, it)$	$F_P, \quad F_D$
252	*AvgPosn*$(\emptyset, d, it)$	$F_P, \quad F_D$
253	*HDCTR*$(\emptyset, d, it)$	$F_P, \quad F_D$
254	*WCTR*$(\emptyset, d, it)$	$F_P, \quad F_D$
255	*PosnCTR*$(\emptyset, d, it)$	$F_P, \quad F_D$
256	*Listings*$(\emptyset, u, ijt)$	$F_S, \quad F_U$
257	*CTR*$(\emptyset, u, ijt)$	$F_S, \quad F_U$
258	*ADT*$(\emptyset, u, ijt)$	$F_S, \quad F_U$
259	*AvgPosn*$(\emptyset, u, ijt)$	$F_S, \quad F_U$
260	*HDCTR*$(\emptyset, u, ijt)$	$F_S, \quad F_U$
261	*WCTR*$(\emptyset, u, ijt)$	$F_S, \quad F_U$

Continued on next page

Table A.1 – continued from previous page

Feature No.	Feature Name	Feature Classes
262	*PosnCTR*$(\emptyset, u, ijt)$	F_S, F_U
263	*SkipRate*$(\emptyset, u, ijt)$	F_S, F_U
264	*Listings*$(\emptyset, d, ijt)$	F_S, F_D
265	*CTR*$(\emptyset, d, ijt)$	F_S, F_D
266	*ADT*$(\emptyset, d, ijt)$	F_S, F_D
267	*AvgPosn*$(\emptyset, d, ijt)$	F_S, F_D
268	*HDCTR*$(\emptyset, d, ijt)$	F_S, F_D
269	*WCTR*$(\emptyset, d, ijt)$	F_S, F_D
270	*PosnCTR*$(\emptyset, d, ijt)$	F_S, F_D
271	*AggListings*(it)	F_P
272	*AggCTR*(it)	F_P
273	*AggHDCTR*(it)	F_P
274	*AggADT*(it)	F_P
275	*AggListings*(ijt)	F_S
276	*AggCTR*(ijt)	F_S
277	*AggHDCTR*(ijt)	F_S
278	*AggADT*(ijt)	F_S
279-288	*ClickProb*(i, p, t)	F_P
289	*NumSearchResults*(q)	F_G, F_Q
290	*NumClickedURLs*(q)	F_G, F_Q
291	*Day*(i, j, t)	F_S, F_U
292	*QueryTerms*(q)	F_Q
293	*SERPRank*(u, i, j, k)	

B Gradient Boosted Regression Trees

Gradient boosted regression 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). In this section, we introduce the concepts of Classification and Regression Tree (CART) and boosted CART. We present the high level overview of these models here and refer interested readers to Murphy (2012) for details.

B.1 Classification and Regression Tree

CART methods are a popular class of prediction algorithms that recursively partition the input space corresponding to a set of explanatory variables into multiple regions and assign an output value for each region. This kind of partitioning can be represented by a tree structure, where each leaf of the tree represents an output region. Consider a dataset with two input variables $\{x_1, x_2\}$, which are used to predict or model an output variable y using a CART. An example tree with three leaves (or output regions) is shown in Figure B.1. This tree first asks if x_1 is less than or equal to a threshold t_1. If yes, it assigns the value of 1 to the output y. If not (*i.e.*, if $x_1 > t_1$), it then asks if x_2 is less than or equal to a threshold t_2. If yes, then it assigns $y = 2$ to this region. If not, it assigns the value $y = 3$ to this region. The chosen y

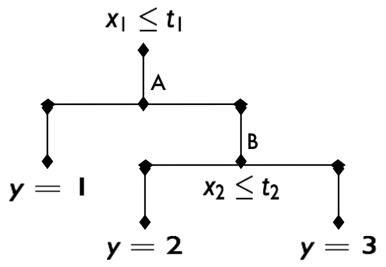

Figure B.1: Example of a CART model.

value for a region corresponds to the mean value of y in that region in the case of a continuous output and the dominant y in case of discrete outputs.

Trees are trained or grown using a pre-defined number of leaves and by specifying a cost function that is minimized at each step of the tree using a greedy algorithm. The greedy algorithm implies that at each split, the previous splits are taken as given, and the cost function is minimized going forward. For instance, at node B in Figure B.1, the algorithm does not revisit the split at node A. It however considers all possible splits on all the variables at each node. Thus, the split points at each node can be arbitrary, the tree can be highly unbalanced, and variables can potentially repeat at latter child nodes. All of this flexibility in tree construction can be used to capture a complex set of flexible interactions, which are not predefined but are learned using the data.

CART is popular in the machine learning literature because it is scalable, is easy to interpret, can handle a mixture of discrete and continuous inputs, is insensitive to monotone transformations, and performs automatic variable selection (Murphy, 2012). However, it has accuracy limitations because of its discontinuous nature and because it is trained using greedy algorithms. These drawbacks can be addressed (while preserving all the advantages) through boosting, which gives us boosted regression trees.

B.2 Boosting

Boosting is a technique that can be applied to any classification or prediction algorithm to improve its accuracy (Schapire, 1990). Applying the additive boosting technique to CART produces gradient boosted trees, which has now been shown empirically to be the best classifier available (Caruana and Niculescu-Mizil, 2006; Hastie et al., 2009). This method can be viewed as performing gradient descent in the function space using shallow regression trees (with a small number of leaves). Gradient boosting with regression trees works well because it combines the positive aspects of CART with those of boosting. CART, especially shallow regression trees, tend to have high bias, but have low variance. Boosting CART models addresses the bias problem while retaining the low variance. Thus, produces high quality classifiers.

A gradient boosted tree model can be interpreted as a weighted linear combination of a series of regression trees, each trained sequentially to improve the final output using a greedy algorithm. Let $L(x)$ be the output of a gradient boosted regression trees. Then, we can write $L(x)$ can be written as:

$$L_N(x) = \sum_{n=1}^{N} \alpha_n l_n(x, \beta_n) \tag{B.1}$$

where $l_n(x, \beta_n)$ is the function modeled by the n^{th} regression tree and α_n is the weight associated with the n^{th} tree. Both $l(\cdot)$s and αs are learned during the training or estimation. MARTs are also trained using greedy algorithms. $l_n(x, \beta_n)$ is chosen so as to minimize a pre-specified cost function, which is usually the least-squared error in the case of regressions and an entropy or logit loss function in the case of classification or discrete choice models.

Optimization Metric	AERC Gain (%)	NDCG Gain (%)
DCG	9.17	2.26
MAP	9.31	2.29
RR	9.33	2.29
ERR	9.19	2.26
NDCG	9.43	2.32

Table C.1: Comparing AERC gains and NDCG gains under different optimization metrics.

C Robustness Checks and Model Comparisons

C.1 Model Comparisons Using Other Optimization Metrics

We now examine whether our results are robust to using other optimization metrics. We consider four other metrics:

- Discounted Cumulative Gain (DCG)

  This is the non-normalized version of NDCG, as defined in Equation (8). Unlike NDCG, DCG gives more weight to searches that recover more relevant documents. Please see (Järvelin and Kekäläinen, 2000; Järvelin and Kekäläinen, 2002) for details.

- Mean Average Precision (MAP)

  We start by defining the Precision-at-p ($P@p$), which is the fraction of relevant results at position p. For example, consider a list of five documents, with relevant documents at positions 1 and 3. Then: $P@1 = 1, P@2 = \frac{1}{2}, P@3 = \frac{2}{3}, P@4 = \frac{2}{4}, P@5 = \frac{2}{5}$. To calculate Average Precision AP for a list, we simply average the $P@p$ for all ps where there is a relevant document. For the example above, it would be: $AP = \frac{1}{2}\left(\frac{1}{1} + \frac{2}{3}\right) = .83$. When AP is averaged over all the searches in the data, we get MAP. Please see Baeza-Yates et al. (1999) for details.

  MAP is better than pairwise metrics like Pairwise Likelihood because it takes the entire list into account. However, it cannot handle multiple levels of relevance making it less efficient that NDCG.

- Reciprocal Rank (RR)

  Is the reciprocal of the rank at which the first relevant document was retrieved. $RR = 1$ if a relevant document was retrieved at rank 1, $RR = 0.5$ if not if the first relevant document was retrieved at rank 2 and so on.

  RR implicitly assumes that the user only wishes to see one relevant document and that she will scroll down until a relevant document is found. If that document is ranked at p, then $RR = p$. While this is reasonable for navigational (or do) queries, it may not capture the behavior of users in more general settings. See Craswell (2009) for details.

- Expected Reciprocal Rank (ERR)

  Is the expectation of the reciprocal of the position at which a user is satisfied with his search. It was introduced by Chapelle and Chang (2011) and depends on a cascade user model, as defined in their paper.

We re-run the LambdaMART model with each of the above as the optimization metric and present the results in Table C.1. We show the percentage improvement in both AERC and NDCG on the test data compared to the baseline. 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 optimize and is therefore a more objective evaluation criterion.

The main finding is that results are robust to the metric used. All four metrics – DCG, MAP, RR, and ERR, perform reasonably well. However, NDCG offers the best performance on AERC, suggesting that when we optimize on this metric, we are able to achieve the best performance on substantive measures of search quality.

C.2 Model Comparisons Using Other Training Algorithms

Method	Training parameters
Pairwise Maximum Likelihood	Logistic regression with linear combination of the 293 features and BFGS optimizer
Ranknet	1 hidden layer with 10 nodes; learning rate of 5×10^5
Rankboost	300 trees with 10 features each
Adarank	500 training rounds
Coordinate Ascent	25 iterations per dimension
LamdaRank with CART	Tree with 1000 leaves
LambdaRank Random Forests	300 trees with 100 leaves each

Table C.2: Details of training parameters used for each model in the comparative analysis.

Method	AERC Gain (%)	NDCG Gain (%)
Pairwise Maximum Likelihood	0.03	0.01
Ranknet	0.03	0.01
RankBoost	3.50	0.94
AdaRank	3.12	1.49
Coordinate Ascent	6.96	1.77
Lambda Rank with CART	2.60	1.50
LambdaRank with Random Forests	5.24	1.63
LambdaRank with MART (our approach)	9.43	2.32

Table C.3: Model comparisons. The percentage gain in AERC and NDCG are shown.

We now compare the performance of our approach to a number of other models. In all the comparisons, we present the percentage improvement in AERC as well as NDCG compared to the baseline. For all the models, the training and tuning parameters are shown in Table C.2, and the results are shown in Table C.3.

First, we consider the standard technique used in the marketing literature – Pairwise Maximum Likelihood with an assumed functional form for $f(\cdot)$. We find that Pairwise Maximum Likelihood with a score vector that is linear in features shows no improvement in test data compared to the single feature benchmark (with just *SERPRank*). The lackluster performance of the Maximum Likelihood model stems from two issues. First, because it is a pairwise gradient-based method it cannot optimize a list of documents well. Second, it cannot capture the unknown patterns in data because it assumes the functional form of the score vector. The next algorithm considered, RankNet, relaxes the second assumption using a neural network (Burges et al., 2005), and it also does not produce any improvement. The third pairwise algorithm, RankBoost, uses boosted regression trees (Freund et al., 2003) as the underlying learner. This performs better than the former two, i.e., boosted trees are better learners for our problem than both neural nets and fixed functional forms.

Next, we consider two other listwise algorithms that are commonly used in the learning to rank literature – Coordinate Ascent and AdaRank (Metzler and Croft, 2007; Xu and Li, 2007). In both these cases, we optimize for NDCG. While these two models are better than pairwise models, they still do not approach the performance of our method.

Finally, we consider methods that use LambdaRank but instead of MART use a different machine learning algorithm to learn $f(\cdot)$. We explore two other tree-based algorithms – CART (a single decision tree) and Random Forests (averaging with trees). In both cases, we use NDCG the optimization metric.

We find that LambdaMART continues to offer the best performance among all these models. Below we discuss a few interesting details from this comparative analysis. LambdaRank algorithm paired with CART (which consists of a single deep decision tree) only offers a small improvement in prediction (2.6% in AERC and 1.50% in NDCG). A key disadvantage of CART is that while increasing tree depth reduces prediction bias, it also increases prediction variance on test data, making it a weak predictor. The solution to this problem is to use a combination of trees. There are two methods that use this approach. The first one, MART, which we use in our main model, starts with shallow trees (that have high bias and low variance) and through the process of boosting adds trees to continually reduce the bias in different parts of the data. In contrast, Random Forest starts with deep trees (that have low bias and high variance) and averages over a large number of trees to reduce the variance. In general, boosting performs better than random forests because the latter is simply a process of bootstrapping or averaging over many samples whereas boosting fine-tunes the performance of the model in each additional step by focusing on the aspects of the data which the earlier steps do not explain well.[1] In our setting, while Random Forest performs better the CART by achieving improvements of 5.24% and 1.63% in AERC and NDCG respectively. However, it still falls short of MART. In sum, we find that our modeling approach offers better performance on both predictive accuracy as measured by NDCG and substantive search quality as measured by AERC.

C.3 NDCG at Different Positions

Position	With Personalization			Baseline	Percentage
	NDCG@p on Training Data	NDCG@p on Validation Data	NDCG@p on Test Data	NDCG@p on on Test Data	Gain on Test Data
$P = 3$	0.7762	0.7744	0.7745	0.7621	1.627%
$P = 5$	0.8034	0.8009	0.8010	0.7849	2.051%
$P = 8$	0.8126	0.8100	0.8105	0.7926	2.258%
$P = 10$	0.8141	0.8117	0.8121	0.7937	2.318%

Table C.4: Results for $NDCG_P$, where $P = 3, 5, 8, 10$.

Finally, we compare how our results change when we train and test the model on smaller subsets of data, specifically up to position P. In addition to position ten, we consider three other positions, $P = 3, 5, 8$. The results from this analysis are presented in Table C.4.

At a high level, we find that personalization helps even if we assume that consumers only look at the top $P < 10$ positions. However, the extent of improvement is monotonically reducing 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. This naturally reduces the accuracy of the trained model. For example, for $P = 3$, we are working with less than one-third of the data used in the full model. Second, when we ignore positions below P, we forgo the possibility of migrating relevant documents from positions below P to the top P positions. Recall that we showed earlier that personalization increases the CTR for position one by drawing from lower positions, mainly positions three and ten (see Figure 11). By focusing only on the top P positions, we are forgoing the ability to migrate the clicks to positions lower than P upwards. Intuitively, this suggests that if the consumers in fact look and click at positions at lower positions, then models that ignore lower positions will suffer.

[1]We refer interested readers to the lecture notes by Trevor Hastie available at http://jessica2.msri.org/attachments/10778/10778-boost.pdf for a nice discussion of this issue.

D Scalability

We now present some measures on the scalability of our approach to large datasets. Note that there are three aspects of modeling – feature generation, testing, and training – all of which are computing and memory intensive. In general, feature generation is more memory intensive (requiring at least 140GB of memory for the full dataset), and hence it is not parallelized. The training module is more computing intensive. The *RankLib* package that we use for training and testing is parallelized and can run on multi-core servers, which makes it scalable. All the module (feature generation, training, and testing) were run on Amazon EC2 nodes since the computational load and memory requirements for these analyses cannot be handled by regular PCs. The specification of the compute server used on Amazon's EC2 node is – Intel(R) Xeon(R) CPU $E5 - 2670$ v2 @ 2.50GHz with 32 cores and 256GB of memory. This was the fastest server with the highest memory available at Amazon EC2.

We now discuss the scalability of each module – feature generation, training, and testing – one by one, using Figures D.1, D.2 and D.3. The runtimes are presented as the total CPU time added up over all the 32 cores. To construct the 20% dataset, we continue to use all the users and sessions from days $1 - 25$ to generate the global features, but use only 20% of the users from days $26 - 27$ to generate user and session specific features.

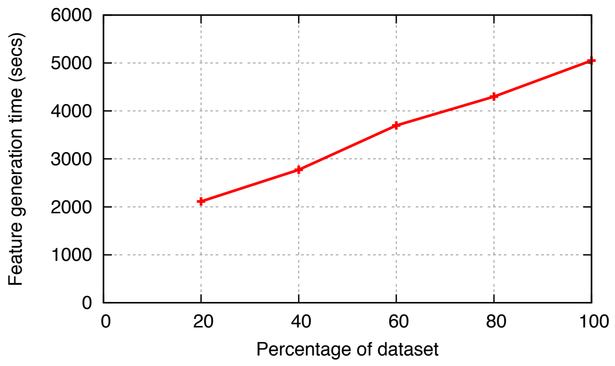

Figure D.1: Runtimes for generating the features the model on different percentages of the data.

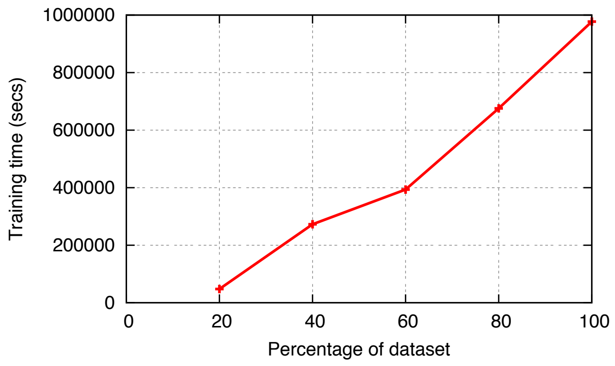

Figure D.2: Runtimes for training the model on different percentages of the data.

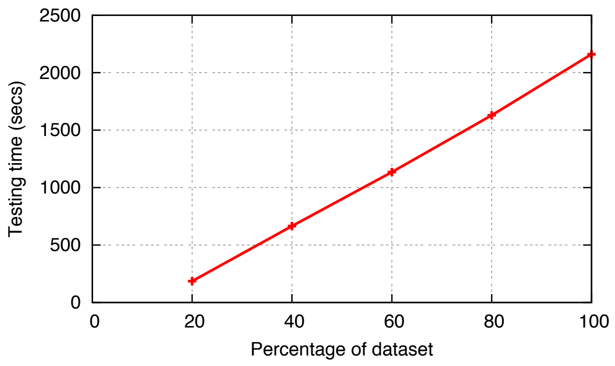

Figure D.3: Runtimes for testing the model on different percentages of the data.

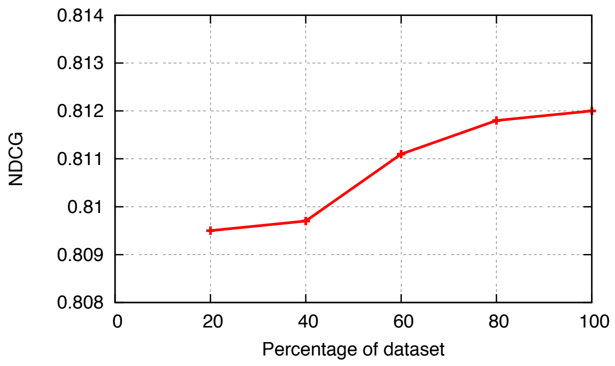

Figure D.4: NDCG improvements from using different percentages of the data.

For the full dataset, the feature generation module takes about 83 minutes on a single-core computer (see Figure D.1), the training module takes over 11 days (see Figure D.2), and the testing module takes only 35 minutes (see

Figure D.3). 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 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 datasets in reasonable time-frames.

Next, we compare the computing cost of going from 20% of the data to the full dataset. We find that it increases by 2.38 times for feature generation, by 20.8 times for training, and 11.6 times for testing. Thus, training and testing costs increase super-linearly whereas it is sub-linear for feature generation. These increases in computing costs however brings a significant improvement in model performance, as measured by NDCG (see Figure D.4). Overall, these measurements affirm the value of using large scale datasets for personalized recommendations, especially since the cost of increasing the size of the data is paid mostly at the training stage, which is easily parallelizable.

References

R. Baeza-Yates, B. Ribeiro-Neto, et al. *Modern Information Retrieval*, volume 463. 1999.

C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. Learning to Rank Using Gradient Descent. In *Proceedings of the 22nd International Conference on Machine Learning*, pages 89–96. ACM, 2005.

R. Caruana and A. Niculescu-Mizil. An Empirical Comparison of Supervised Learning Algorithms. In *Proceedings of the 23rd International Conference on Machine Learning*, pages 161–168. ACM, 2006.

O. Chapelle and Y. Chang. Yahoo! Learning to Rank Challenge Overview. *Journal of Machine Learning Research-Proceedings Track*, 14:1–24, 2011.

N. Craswell. *Mean Reciprocal Rank*, pages 1703–1703. Springer US, 2009. ISBN 978-0-387-39940-9. URL `http://dx.doi.org/10.1007/978-0-387-39940-9_488`.

Y. Freund, R. Iyer, R. E. Schapire, and Y. Singer. An Efficient Boosting Algorithm for Combining Preferences. *Journal of Machine Learning Research*, 4(Nov):933–969, 2003.

T. Hastie, R. Tibshirani, J. Friedman, T. Hastie, J. Friedman, and R. Tibshirani. *The Elements of Statistical Learning*, volume 2. Springer, 2009.

K. Järvelin and J. Kekäläinen. IR Evaluation Methods for Retrieving Highly Relevant Documents. In *Proceedings of the 23rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '00, pages 41–48. ACM, 2000.

K. Järvelin and J. Kekäläinen. Cumulated Gain-based Evaluation of IR Techniques. *ACM Transactions on Information Systems (TOIS)*, 20(4):422–446, 2002.

D. Metzler and W. B. Croft. Linear Feature-based Models for Information Retrieval. *Information Retrieval*, 10(3):257–274, 2007.

K. P. Murphy. *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012. ISBN 0262018020, 9780262018029.

R. E. Schapire. The Strength of Weak Learnability. *Machine learning*, 5(2):197–227, 1990.

J. Xu and H. Li. Adarank: A Boosting Algorithm for Information Retrieval. In *Proceedings of the 30th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 391–398. ACM, 2007.