Model Selection and Hyperparameter Tuning In Maps Query Auto-Completion Ranking

Bhagirath Addepalli
bhagiraa@microsoft.com
Microsoft
Bellevue, WA, USA

Hua Li
hli@microsoft.com
Microsoft
Bellevue, WA, USA

Delbert Dueck
ddueck@microsoft.com
Microsoft
Bellevue, WA, USA

ABSTRACT

In this article we consider the ranking problem on Maps Query Auto-Complete, and share insights on making ranking improvements. Query auto-completion (QAC) on Maps differs from traditional auto-completion in that it involves both query- and entity retrieval and ranking. The QAC system on a Maps product is typically tasked with recommending up to 5 business, place, or address entities, and / or completed query-suggestions, to user-typed query prefixes.

Several ideas were considered for ranking improvements. For conciseness, we examine the impact and the relative importance of the following factors: a) ranking problem formulation, b) training data generation, size, and instance, c) model-type and ensembling, d) new and high-value features, e) hyperparameter optimization, and f) data freshness and distribution drift. The performance of these factors was evaluated relative to a simple baseline model, trained using Bing query-logs. Incorporating the factors helped achieve approximately 37% of the maximum possible gain. The intuition behind the performance of the various factors, along with the practical aspects and constraints of model development are detailed.

KEYWORDS

stratified sampling, hyperparameter optimization, boosted trees, data freshness, distributional drift

ACM Reference Format:
Bhagirath Addepalli, Hua Li, and Delbert Dueck. 2019. Model Selection and Hyperparameter Tuning In Maps Query Auto-Completion Ranking. In *Proceedings of ACM Conference (CIKM MoST-Rec Workshop).* November 2019, Beijing, China, 12 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

In this article we consider the ranking problem in the context of Query Auto-Completion (QAC) on Maps search, and share practical insights on making ranking improvements. The QAC service is responsible for recommending relevant query-completions and entities as users type query prefixes into the Maps search box. Up to 5 results are usually shown for every character typed by a user into the search box. There are several user, system, and business-level advantages to having a high-quality QAC service within a Maps product offering. Some of the benefits include: a) improvement in a users search experience by recommending entities the user is looking for, which reduces the users' typing effort, b) provide users a window into entities other users consider relevant or important, c) proceed directly to result fulfillment if a user found a suggestion to be relevant and clicked on it (instead of routing the query through the main search stack), d) in addition to saving typing effort, a QAC service gives the user the first indication of the quality of a search engine (in addition to the quality of the base-map data on Maps), e) the QAC service could be monetized through an enterprise offering, etc.

The layout of the paper is as follows. Since the QAC ranking problem on Maps differs significantly from traditional QAC ranking (in Web search / SERP) and click prediction problems, we first introduce terminology that will be used through the rest of the paper. We then do a brief review of the existing literature on ranking in QAC systems. The metrics used to measure the quality of the QAC service are then described. Metrics provide an objective measure to quantify ranking improvements. Real-world ranking systems typically comprise of several models that complement each other and are optimized to different metrics. We briefly discuss the stack architecture of a QAC service and the specific ranking component this paper addresses. We then go over the training data generation process, model formulation, and the strategy, tactics, and tricks employed for ranking improvements. We finish by discussing some of the challenges inherent in the QAC ranking problem, and possible ways to work around them.

2 TERMINOLOGY

This section provides definitions for some of the terms used through the rest of the paper. An example usage of these terms in a search scenario on Bing Maps is shown in Figure 1. The scenario illustrated in Figure 1 is that of a user located in Kirkland, WA, with Mapview over Kirkland, WA, searching for a Thai restaurant named Wild Ginger, located in Bellevue, WA. In a QAC search scenario, the entity the user intends to search-for (and would click on) is referred to as the *ideal entity*.

For a given query-prefix, results returned by QAC on a typical Maps product belong to one of the following four entity-types: Business, Place, Address, and *Query-suggestion*. Businesses represent physical entities that can be contacted (via. an address or phone number). Places represent geographical regions (e.g. Seattle, WA) or points-of-interest (POI) (e.g. Golden Gate bridge). Addresses could be either roads (e.g. Main street) or specific point-addresses on the road (e.g. 123 Main street). *Query-suggestions* are not physical entities. They may be viewed as representations that wrap one or more physical entities, or query-completions mined from search logs or entity indexes. They could used to fulfill search intent when there could be multiple entities that could be relevant to the query-prefix - for example, for the query-prefix = {starb}, returning a *query-suggestion* Starbucks may be more useful, instead of showing specific Starbucks franchises (businesses) (especially if the business

CIKM MoST-Rec Workshop, November 2019, Beijing, China
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

density of Starbucks outlets is high within the area of interest). Ranking between the various entity-types for a given prefix adds to the challenge of QAC ranking on Maps.

In Figure 1, as the user types the query characters (conversation), the QAC service returns candidate suggestions. At the prefix = {wild}, the ideal entity shows-up in the result-list and it is assumed that the user clicks on the Wild Ginger restaurant in Bellevue. It should be noted that for prefix = {wild}, QAC returns a *Query-suggestion* as the top result (position 1), Businesses (restaurants) at positions 2,4, and 5, and a Place at position 3. In this example, since we assume that the user clicked on a suggested entity, the conversation is an unabandoned conversation.

For the purposes of this paper, we assume that for *clicked conversations* (conversations in which a user clicked on QAC result), the ideal entity (true user-intent) is always known. Note that this assumption ignores the possibility of accidental or noisy clicks. It follows that for *unclicked conversations* (includes abandoned and unabandoned conversations), the ideal entity is assumed to be unknown.

Figure 1: Terminology used to denote the various concepts in QAC on Maps.

3 RELATED WORK

There have been quite a few works (see for example, [3, 27, 33]) that have studied the relevance and ranking problem in QAC systems. These studies however view the problem from a Web search (SERP) standpoint. Also, most of these studies investigate specialized aspects of ranking, such as personalization [33], improving result diversity [7], leveraging session and other context information [3], improving performance on rare prefixes [27], etc. In the current work we focus exclusively on modeling the QAC ranking problem on Maps, and detail ways in which the ranking problem can be set up and solved, and improvements could be made.

The QAC ranking problem may also be viewed more broadly as a click prediction problem - predicting entities that a user is likely to click, and recommending those as the top search suggestions. Click prediction problems arise in several other areas of search and advertising (in addition to Web and Maps QAC). There are quite a few articles from these domains that discuss strategies, and novel and robust techniques to solve such problems (see for example, [2, 5, 15, 17, 24, 25, 34]). The main difference between the current work and the cited articles lies in the specific problem area (Maps), and its various nuances. Also, most of these works adopt a pointwise approach [17, 25] to solving the ranking problem (formulate the ranking problem as a classification problem), while we consider both pairwise and listwise approaches to ranking.

The current work draws upon some of the existing work, adapts it to the Maps domain, and suggests possible improvements. For instance, in the current work we use the training data labeling and generation strategy from [33] and [24], and demonstrate how extending it using stratified sampling could yield better results. Click logs were used to generate the training data in the works of [33] and [24]. Here, we study the impact of using click- versus human-labeled data on ranking accuracy.

Training on click logs alone poses several key challenges, including: a) it incentivizes the ranker to continue to do better on examples it already does well on, b) it creates a distribution mismatch between training and evaluation sets (since the ranker is evaluated on how well it does on (a random sample of) all conversations, and not just the clicked ones), c) it biases the training data generated and used for subsequent runs, etc. The work of [36] proposes addressing these issues using reinforcement learning techniques (explore-exploit strategies). In the current work we take a more practical approach, and illustrate how unclicked conversations could be used in training and evaluation to mitigate the above challenges.

The metrics employed to evaluate QAC effectiveness on Maps are an extension to those proposed in [21], and may be viewed as an enhanced version of the mean reciprocal rank (MRR) metrics used in [33]. For ranking, we consider gradient boosted decision tree models (GBDT) and deep neural networks (DNN), and extend the analysis by examining the impact of various implementations of GBDT's on prediction accuracy. For hyperparameter optimization, we assess the impact of the various model parameters and training set size (as was done in the cited works). Additionally, we demonstrate how the training data instance and the sampling scheme could be could be viewed as hyperparameters. Click prediction rankers typically suffer from concept-drift (see for instance, [17, 18]). In the current work we demonstrate that the same holds true for QAC on Maps, and characterize the extent to which model performance deteriorates over time. We finally discuss challenges inherent in real-world QAC systems such as optimizing for multiple metrics and the discrepancy between training and evaluation metrics.

4 STACK ARCHITECTURE

At a conceptual level, a typical QAC ranking stack comprises of three main layers. High-level details of these layers are provided below. It should be noted that for the purpose of building and improving a QAC ranking model, we consider the stack to be stateless, and do not consider conversation-level or session-level features.

4.1 Filter-Set Generation Layer

The role of this layer is to reduce the set of all possible entities that can be returned for a given query-prefix to a more manageable

number (referred to as *filter-set* from hereon). To compute the *filter-set*, this layer uses simple models and specialized *tries*, while taking into account mined grammars, contextual information such as user-location, mapview size, historical data, and other factors. The *filter set* generation step on its own does not rank entities. The *filter-set*, after featurization, is passed to the next ranking layer.

4.2 Ranking Layer

Several ranking models could be used in the ranking layer, either sequentially or in parallel. Each of these models could be optimized to different evaluation metrics. For instance, in domains such as Maps search, individual models could be trained and optimized for matching query-intent, returning popular results (overriding mapview and user-location considerations), predicting click propensities, etc.

In this article we focus on building and making improvements to a click prediction ranker. The click ranker is one of the most important components of the QAC ranking stack. The click ranker ranks the entities in the *filter-set* based on their likelihood of getting clicked by the user. For click propensity computation, the click ranker considers several distance and popularity signals, in addition some entity-prefix match signals, and other historical signals. The click ranker formulates the click prediction problem as a learning-to-rank (LETOR) problem [23, 26]. It uses either the pairwise or listwise approach to ranking, depending on the granularity of the training labels used.

4.3 Aggregation and Post-processing Layer

The aggregate ranker provides the final ranking of the entities in the *filter-set*. The final scores are computed by combining scores from the previous layer. The ranking model in the aggregation layer is usually determined by solving a multi-objective optimization problem, by trading-off several competing metrics. The top five entities returned by the aggregate model are usually displayed to the user, with minor modifications.

5 CLICK-LOGS SCHEMA AND FEATURES

In the present work we use logged QAC conversations from Bing Maps to build and improve the click ranker. For the purposes of this work, it is only assumed that the search engine logs ConversationId's, every prefix in a conversation and the results shown for that prefix, and the user-action that denotes the end of the conversation. A conversation typically ends with a user either selecting a QAC recommendation, or by submitting the query to Maps vertical, or by abandoning the conversation. Several features could be used in the ranking model. The features could be broadly categorized into historical and contextual features.

- **Historical features**: these features capture the historical importance of various entities, and are computed from the click logs and entity indexes. Examples of such features include: static ranks for the various entity-types, click through rates for the various entity types, average prefix lengths at click for the various entity-types, how frequently a particular entity was search in the last week, month, year, etc.
- **Contextual features**: these features are extracted based on the current prefix context, and by considering the various entities in the *filter-set*. Examples of such features include: prefix length, mapview size, distance between mapview and user-location, distance between mapview / user-location and the various entities in the *filter-set*, prefix-entity match, user language, day of the week, week of the month, month of the year, time of the day, etc.

6 METRICS

Several online and offline metrics are used to measure the quality of QAC ranking models. In this article we focus on an offline click prediction metric called the normalized cumulative gain (NCG). All the ranking improvements are evaluated against this metric. The NCG metric draws from the work of [21], and adapts it to the Maps scenario.

The basic idea behind offline click prediction metrics in general, and NCG in particular is the following. All conversations observed Maps QAC may be classified into one of *clicked* or *unclicked* (includes *abandoned*) conversations. For a *clicked* conversation, the ideal entity is assumed to be the one clicked by the user. For an *unclicked* conversation, we may be able to deduce the ideal entity through a human auditor or some other model. Given an ideal entity for a query-prefix, we could optimize the QAC service to try to return the ideal entity at as low a prefix length as possible. Figure 2 depicts this scenario. In the figure, query = {wild ginger}, and ideal entity = {Wild Ginger, Bellevue, WA} (denoted as WGB). The QAC service returns a maximum of up to five suggestions that are visible to the user. It could be argued, that an ideal QAC service, for this particular query, is one that returns the ideal entity at every prefix (the prefix-length dimension, *i*), and at a result depth of 1 (top-ranked result) (result-depth dimension, *j*).

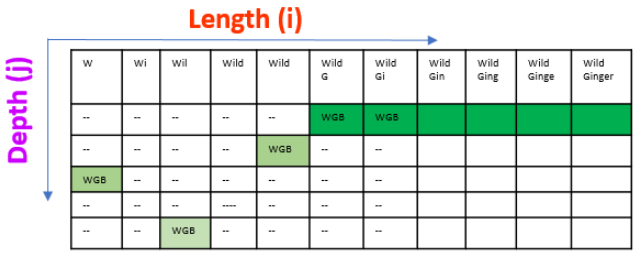

Figure 2: Schematic describing the key ideas behind the NCG metric.

It should be noted that click-through rates (CTR) on ranked lists usually decay exponentially with the result-depths. With a service such as QAC, one may consider the CTR distribution as a joint distribution over the prefix-length and result-depth dimensions. Figure 3 shows the CTR distribution along the prefix length (with a range of 1 to 30 characters) and result depth (top 5 results) dimensions, based on a sample of conversations from the month of May, 2018. In the plot, the green and red regions indicates areas with high and low click probabilities. The findings from the plot are similar in trend (with domain-specific differences) to those reported in [24, 28], for PC (SERP) and Mobile scenarios. From the figure it is seen that in general, query-prefixes on Maps tend to be longer than SERP, and users continue to use the QAC service at longer prefix lengths (which is in contrast to SERP, where queries tend to be shorter). The longer query-prefixes are due to the differences

in the query-intents on SERP and Maps, wherein on Maps, users usually search for specific Business, Place, and Address entities.

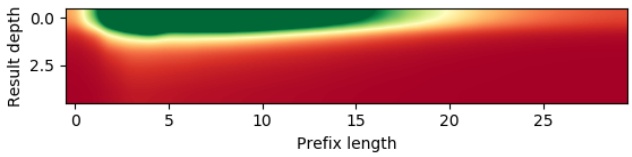

Figure 3: Click-through rate (CTR) distribution for QAC on Maps along the prefix length and result depth dimensions. The green and red regions represent areas of high and low CTR values.

6.1 Normalized Cumulative Gain (NCG):

The above discussion outlines a framework to formulate the NCG metric. For a given conversation, the NCG metric takes into account the result-set for every prefix, and rewards returning of the ideal entity in the result-set. The reward is higher if the ideal entity is returned at lower prefix lengths, and ranked at top positions with respect to the depth. Computation of such a metric would require definition of a weight- or expected-utility matrix $W_{i,j}$. The matrix $W_{i,j}$ encodes the utility or reward from returning an ideal entity (e^*) at a given prefix-length i and result-depth j. The metric score ($NCG(q)$) for a single conversation (query q) can be defined as shown in Equation 1 below. In Equation 1, $e_{i,j}$ denotes the entity returned at prefix-length i and result-depth j, and $I\{.\}$ is the indicator function. The weights $W_{i,j}$ are assumed to be normalized such that the metric score is equal to 1, when the ideal entity is returned for all prefix-lengths at a result-depth of 1. The weight matrix $\mathbf{W}$ is usually deduced from the query-logs, and by in itself provides an indication of the current quality of the QAC service. The matrix $\mathbf{W}$ may be slightly adjusted to guide the trajectory of the product.

$$NCG(q) := \Sigma_{i=1}^{|q|} \Sigma_{j=1}^{5} W_{i,j} I\{e_{i,j} = e^*\}. \quad (1)$$

A few characteristics of the NCG metric are worth noting: a) the NCG metric is similar in spirit to the MRR metric computed per prefix, and averaged across all the prefixes in the conversation. It uses custom discounting factors learned from query-logs, b) the weight matrix is calibrated per query-intent. This ensures that the final metric score is not biased by queries with a particular intent (e.g. queries for Address entities, which tend to be longer), c) in our experience, gains on the NCG metric correlate well online metric gains (improved user engagement).

7 EXPERIMENTAL SETUP

This section details the training data generation process, the ranking models considered, and the evaluation sets used to measure improvements. The training and evaluation sets were sampled from Bing query logs. The training set was sampled from data collected in February 2019, while the evaluation set was based on March 2019 data.

7.1 Training Data Generation

7.1.1 Human-Labeled Data. For training, we consider both human-labeled data and click logs. The human-labeled data was generated by sampling prefixes from query-logs, and asking judges to label candidate entities for the prefixes on a four point scale (Excellent, Fair, Bad, DSAT). There are two main disadvantages with human-labeled data: a) the training set cannot be too large, and b) query-intent ambiguity is higher at lower prefix lengths. This is due to the fact that on Maps, in addition to query-prefixes, judges also have to take into account user-location, mapview, mapview size, the distance between mapview and user-location, result density, and other geographical factors when labeling candidate entities. This results lower quality judgments at lower prefix lengths. The human-labeled training data comprised of 15,000 prefixes, with about 15 candidate entities per prefix.

7.1.2 Click Logs. Click logs are an inexpensive source of large training sets. For the experiments conducted, we only considered the *clicked* conversations. As stated previously, ignoring the *unclicked* conversations results in loss of valuable information, and creates distributional differences between training and evaluation sets. The main challenge with accounting for the *unclicked* conversations is that the ideal entity is unknown for these queries. A possible inexpensive workaround for this conundrum would be to consider results returned by Maps vertical (upon submission of the query (when the QAC is not used)) as candidate ideal entities. For training data generated from user clicks, we considered several variations of generating training data, the high-level details of which are given below.

- **Simple random sampling of prefixes at click**: in this scheme we sample prefixes at which users clicked on an entity, and use it for training.
- **Simple + Stratified sampling of prefixes at click**: in this scheme, we consider prefixes at click, but also selectively sample prefixes of varying lengths and entity-types (at click). The rationale behind this approach is to force the learning model to see more example of lower prefix length and of different entity-types (query-intents).
- **Synthesizing lower length prefixes**: this approach was adopted by [24, 33], wherein a prefix at click is first sampled, and then lower length prefixes are synthesized from this prefix. For synthesizing lower prefix lengths, we consider prefixes which are up to 5 characters shorter than the sampled prefix.

Several other combinations of the above schemes were also considered. The main difference between synthesizing prefixes versus performing stratified sampling is that the former approach assumes and enforces the clicked entity to be the ideal entity (which is what the NCG metric) does, while the latter considers lower prefix length entities on which users actually clicked. In our experience, with large training sets the synthesizing approach works out fine, but could be noisy (and hurt prediction accuracy) with smaller training sets. Also, when synthesizing prefixes, one of has the option of either letting each prefix (and its candidate entities) to be its own *query-group* during training, or have all the prefixes in the conversation to be a part of the same *query-group*. The latter approach seems more sensible, as it is more aligned with the evaluation metric. However, adopting it requires adjusting the labels such that the training and evaluation loss functions are in sync.

7.1.3 Ranking Models. For ranking, we considered four variants of gradient boosted decision trees (GBDT), and a deep neural network. The GBDT variants considered were: an in-house implementation of the LambdaMART model ([6]), LightGBM ([20]), XGBoost ([10]), and CatBoost ([11, 32]). Each model has its own merits. For instance, the LightGBM model is extremely fast to train, and can be accurate at the expense of overfitting at times. XGBoost has built-in regularization, which makes it robust to overfitting, but is slower to train in comparison to LightGBM. CatBoost handles categorical features in a more effective and efficient manner. In our experiments, LightGBM and XGBoost models did as well as (if not better than) the in-house LambdaMART model. For DNN's, we considered the ranking library from TensorFlow [31].

7.1.4 Evaluation Sets: The evaluation set was a random sample of 10,000 *clicked* conversations from March 2019. To capture effects such as concept-drift, evaluation sets of size 10,000 were sampled from different months of the year (additional details provided in the section on concept-drift). As mentioned previously, for better results, the evaluation set should be comprised of both *clicked* and *unclicked* conversations.

8 STRATEGY, TACTICS, AND TRICKS

As mentioned in the previous sections, in this article we detail the approach adopted to make systematic improvements to a click prediction ranker. The decision to work on click ranker improvements relies on the presupposition that there is a *significant* metric gap between the NCG metrics computed over the entire *filter-set* (assuming ideal ranking), and that computed using the top 5 suggestions. Assuming the search stack configuration to be fixed, the ideal NCG metric over the *filter-set* provides an upper-bound on the maximum realizable gain from the click ranker.

Stated another way, if an *ideal entity* corresponding to a query-prefix in the evaluation set does not belong to the *filter-set* for that prefix, then, the ranking step cannot return it in the top 5 positions. For an evaluation set Q of size n, with queries $\{q_1, ..., q_n\}$, if M represents the maximum number of entities in a *filter-set* (for any prefix that the system can allow), then the mean NCG score for Q over the *filter-set* is defined as shown in Equation 2.

$$NCG_M(Q) := \frac{1}{n}\Sigma_{k=1}^{n}\Sigma_{i=1}^{|q_k|}\Sigma_{j=1}^{M} W_{i,j} I\{e_{i,j} = e_k^*\}. \quad (2)$$

It follows from Equation 2 that, $NCG_M(Q) <= NCG_5(Q)$. Also, the maximum value of $NCG_M(Q)$ occurs if the *ideal entity* e^* is returned at depth $j = 1$ (for prefixes where e^* belongs to the *filter-set*). Denoting this maximum value by $NCG_M^{max}(Q)$, starting with a baseline ranker configuration, the maximum metric gain achievable through ranking improvements (ranking potential) is as shown in Equation 3. The difference between the maximum and the baseline NCG scores is referred to as the *filter-set gap* from hereon.

$$\textit{filter-set gap or ranking potential} := NCG_M^{max}(Q) - NCG_5(Q). \quad (3)$$

Due to the proprietary nature of the data sets used, only percentage improvements made relative to the *filter-set gap* are reported. For instance, performing hyperparameter optimization on a baseline ranker could resulting in metric gap reduction, while some other unsuccessful strategies could lead to poorer models, thereby resulting in gap increase relative to the baseline.

To demonstrate improvements, a baseline ranker based on GBDT was considered. It was trained using 200,000 randomly sampled clicked examples. The pairwise loss function was used for training. Addition details on the ranker configuration are given in the subsequent sections. Given the baseline ranker, the goal of the ranker improvements would then be to reduce the *filter-set gap*. To realize some of potential metric gain, the following approach was devised:

(1) **Appropriate loss function**: the first step was to ensure that the choice of the loss function was not posing an inherent limitation on the prediction accuracy. Ranking problems are typically solved using one of pointwise, pairwise, or listwise loss functions ([23]). The difference between these lies in the way in which they make use of the group structure of the ranking problem. The pointwise approach converts the ranking problem into a classification problem, thereby totally ignoring the group structure (ordering of entities based on some ground-truth label). The pairwise approach transforms the ranking problem into a pairwise classification problem, which preserves some of the group structure. The listwise approach takes the full result ordering into account. It has been shown ([23, 31]), and is fairly intuitive, that the pairwise and listwise approaches outperform the pointwise approach.
When training with human-labeled data, we used the listwise approach. For training data comprised of clicks, we chose to go with the pairwise approach. When using click logs, since the goal of ranking would be to rank the clicked entity above all the unclicked entities, the pairwise and listwise approaches should give similar results. This is because the entities for a given prefix can only take 2 possible labels, *clicked* and *unclicked*, and there is no ordering specified between the unclicked entities.
(2) **Bias:** the next step was to address the *Bias* or under-fitting problem. For the baseline click ranker, the model-type and features were first fixed. Hyperparameter optimization was performed to identify the model configuration that maximized NCG metric gain. It should be noted that assuming a simple baseline configuration, the optimized version is almost always more complex than the baseline (for instance, the baseline configuration may use 50 boosted trees, while the optimum configuration may use 5000).
(3) **Variance**: to ensure that some of the gains from hyperparameter optimization were not lost due to over-fitting, we erred on the side of having large training sets (when using clicks for training), than what the baseline configuration was using. Prediction accuracy and training time were the factors that needed to be balanced when determining the training set size.
(4) **Training data sampling:** training data generated using simple random sampling comes with the inherent limitation that it captures the current state of the system. For instance, if the system is not performing well on Business queries, then this would be reflected in the CTR for the Business segment. In this case, the proportion of Business examples in

Table 1: The main hyperparameters of the baseline ranker configuration

.

Hyperparameter	Value
Number of trees	50
Number of leaves	50
Min. docs. per leaf	50
Learning rate	0.06
Number of features	34

the training set would be lower, due to the low CTR. Experimenting with different training data generation schemes (for example, the proportion of queries of various types, lengths, geographical locations, etc.) can potentially mitigate this issue. In the present work, the various aspects related to training data generation were treated as hyperparameters and tuned.

(5) **New, Improved, and Advanced features:** after reaching the point where with a fixed model, and with a large and well-represented training set, under-fitting was no longer the primary impediment, focus was shifted to adding new, improved, and advanced features. Improved features here refers to improvements made to the current set of features used, such as static ranks. The above cycle was repeated after these changes.
(6) **Data freshness:** since every shipped improvement influences future training data, in our experience, training with newer data resulted in small gains and improved ranking stability.
(7) **Distribution drift:** in QAC systems, distribution drift between the training data and the data the ranker is evaluated on (once put in production) causes a steady metric drop over time. Re-training once in a while can itself produce metric gains. More on this is discussed in the later sections.

9 RESULTS AND DISCUSSION

In this section results from the experiments conducted are shared. Most of the improvements reported are with respect to a GDBT-based baseline ranker, trained using 200,000 randomly sampled clicked examples, using a pairwise loss function. For consistency, and to isolate the effects of concept-drift, the baseline ranker was re-trained with Feb 2019 data. The ranker configuration is shown in Table 1 below. Improvements reported are relative to the *filter-set* gap of this ranker (Equation 3).

9.1 Pointwise vs. Pairwise Loss Functions

To characterize the effect of loss function choice on prediction accuracy, the baseline ranker was trained using the pointwise loss function (classification problem). Training the baseline ranker with a pointwise loss function caused the gap with the *filter-set* (relative to the baseline gap) to go up by approximately 24%. Thus, it is seen that choosing an appropriate loss function has a significant impact on the ranker performance.

9.2 Human-labeled vs. Click Training Data

To better understand the impact of using human-labeled versus click logs, the baseline GBDT ranker was trained with human-labeled data, with the pairwise loss function. The metrics produced by using clicks versus human labels were then compared. From the experiments it was seen that using human-labeled data caused a regression in the gap with the *filter-set* by about 15%. The regression observed may be attributed to the size of the training-set used, as well as the noise-level in the training data. As mentioned previously, human auditors find labeling candidate entities returned at lower prefix lengths (on Maps) to be extremely challenging.

9.3 GBDT's vs. DNN's

To compare the performance of GBDT's versus DNN's, the NCG metric score of the baseline ranker was compared against a DNN trained on the same (click logs-based) data. The DNN configuration comprised of a single neural network with 5 hidden layers. The number of neurons in layers 1 through 5 were 1024, 512, 256, 128, and 64 respectively. The training batch size was set to 256 examples, and ReLU activation functions were used. The DNN was trained for 5000 epochs with the pairwise loss function. All of these parameters were determined through hyperparameter tuning. From the results it was seen that using a DNN caused the *filter-set* gap to increase by about 3%. The results obtained are consistent with other studies where the performance of DNN's and GBDT' were compared on ranking tasks (see for example, [1, 25]).

9.4 Pairwise vs. Listwise Loss Functions

To evaluate the impact of pairwise and listwise loss functions, a DNN was trained with these loss functions using human-labeled data. A DNN was used for these experiments as the TensorFlow ranking library supports both the pairwise and listwise loss functions. The hyperparameter values used for the DNN were the same those described in the previous section. From the experiments it was found that using the listwise loss function yielded better results (as expected). However, the NCG *filter-set* gap as result of using human-labeled data was higher (even when using a listwise loss function), than when click data was used with a pairwise loss function.

From the above experiments the following conclusions were made: a) training data based on clicks seemed to perform better than human-labeled data, given the volume of click-logs, as well as the noise in human-labeled sets, b) since click training data comprises of clicked vs. unclicked labels, a listwise loss function does not add much additional value, and c) GBDT's perform slightly better than standalone DNN's for QAC ranking. Also, all of the above experiments only resulted in metric losses relative to the baseline. In the sections from hereon, we focus on using GBDT's with click training data to produce ranking improvements.

9.5 Fraction of Negative Training Data

The training set is unbalanced, as for every clicked entity in a prefix (query) group (in LETOR problems), there could be several unclicked entities (if one were to consider all the entities in the *filter-set*). The fraction of negative examples per query group has

a considerable influence on the prediction accuracy. Based on experiments conducted, tuning the positive to negative example ratio to approximately 1:20 yielded the best results. Using such a ratio caused the NCG gap with the *filter-set* to reduce by approximately 1%.

9.6 Hyperparameter Tuning

The predictive performance of boosted decision trees strongly depends on careful tuning of the various model hyperparameters. To demonstrate the impact of hyperparameter tuning, hyperaparameters of the baseline model (Table 1) were tuned. The bounds for the hyperparameter space were based partly on model complexity (latency) considerations. To ensure hyperparameter tuning did not result in overfitting, a large training set comprising of about 3 million clicked examples was used (with about 20 unclicked entities per clicked entity in a query-group).

Figures 4- 7 show the effect of the hyperparameter tuning on the reduction in the *filter-set* gap. These plots are based on a subset of the experiments conducted. From the figures it can noticed that the number of leaves and learning-rate have a significant positive effect on the NCG metric. The number of features and minimum documents per leaf (which controls tree complexity and acts as a regularizer) do not significantly affect the metric.

The results obtained are intuitive, and can be explained by considering how boosted trees work. The minimum documents per leaf hyperparameter affects the tree complexity. This factor becomes especially important if the amount of training data is limited. For hyperparameter tuning, since a large training set was chosen, this feature has minimal impact on the metric.

The number of features hyperparameter was varied in the interval [30, 50]. Hyperparameter values in this interval do not significantly influence the metric value. This gives us an indication that the top 30 features considered are probably sufficient (carry the necessary explanatory power) to build a reasonably accurate ranker. More details on the choice of number of features are given in section 9.6.2 on feature importance.

The number of leaves hyperparameter was varied over the interval [20, 100]. Over this interval, we see gap reduction by about 6.67%. The figure suggests that the prediction accuracy increases almost linearly with increasing number of leaves. This behavior can be explained by recognizing that a tree with more leaves represents a more complex learning model. An increase in the number of leaves resulting in metric gain (on the test set) is indicative of the underlying under-fitting problem, which can be addressed through hyperparameter optimization. The following aspects about the number of leaves parameter should however be noted.

(1) One may obtain similar metric gains by either increasing the number of leaves (with a fixed number of trees), or increasing the number of trees (with a fixed number of leaves), or both.
(2) Best-practices cited ([8]) in the literature recommend using a large number of moderately complex trees (instead of a few extremely complex trees, which can lead to over-fitting).

The learning rate also seems to have significant effect on the test set metric. From Figure 7 it is seen that good values for the learning rate for the click ranker seem to be greater than 0.02. The highest *filter-set* gap reduction was observed at a learning rate of approximately 0.04. In all of the above experiments, the number of trees were fixed at 500. Increasing the number of trees beyond this number did not have a significant impact on the metric, and was not done considering the latency constraints.

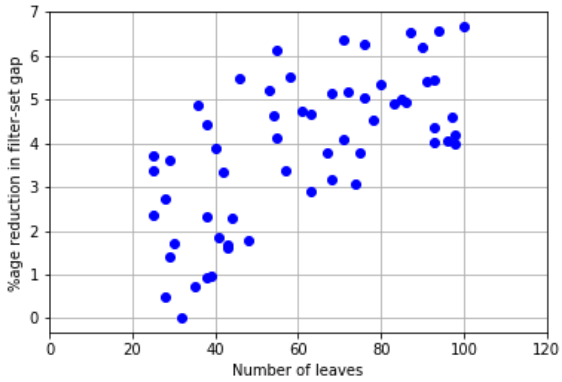

Figure 4: Effect of the number of leaves hyperparameter on *filter-set* gap reduction.

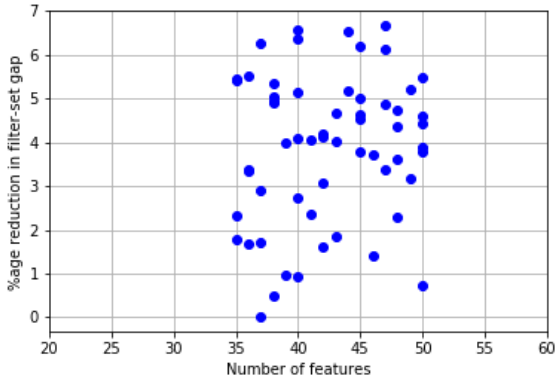

Figure 5: Effect of the number of features chosen in the click ranker on *filter-set* gap reduction.

9.6.1 Effect of Number of Trees. To characterize the effect of the number of trees hyperparameter, the number of trees were varied in the interval [50, 500], and the training and test set errors were measured. NDCG was used the evaluation metric on the training set. Shown in Figure 8 are the normalized NDCG@1 values as the number of trees were varied from 1 to 500. The NDCG@1 values were normalized by the maximum NDCG@1 value (at the end of the training). The other hyperaparameters were held constant for these experiments. From the figure, as expected, it is seen that the training error improves with increasing number of trees. We also however see diminishing returns in metric gain with increasing number of trees.

9.6.2 Feature Importance. The importance of a feature can be captured in terms of its contribution to the overall training error reduction (across all the trees). To study feature importance, approximately 250 features were considered. Figure 9 depicts the marginal

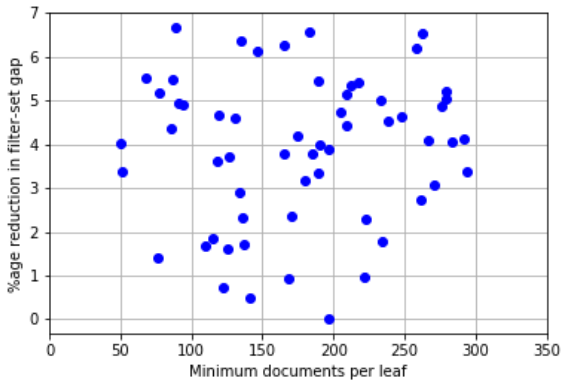

Figure 6: Effect of the minimum documents per leaf hyperparameter on *filter-set* gap reduction.

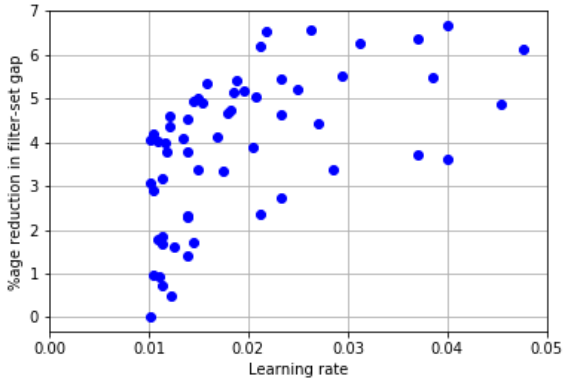

Figure 7: Effect of the learning rate on *filter-set* gap reduction.

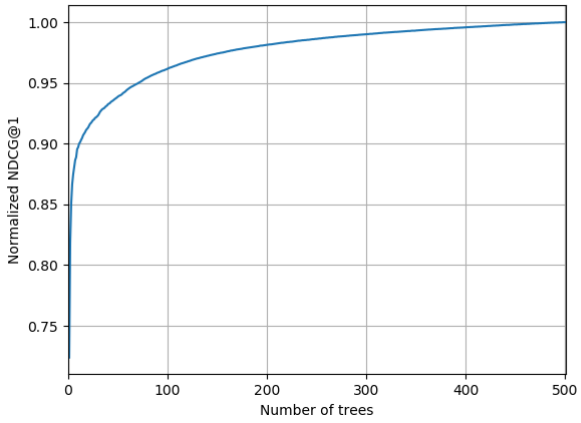

Figure 8: Effect of the number of trees on normalized NDCG@1 on the training set.

gains that are achieved as a result of including features with progressively lower explanatory power. From the figure it seen that the top 10 features contribute to approximately 93% of the training error reduction. There are diminishing returns with including more features (features with lower explanatory power). This data indicates that model complexity could be managed by lowering the number of features without taking a significant hit on the test-set metric.

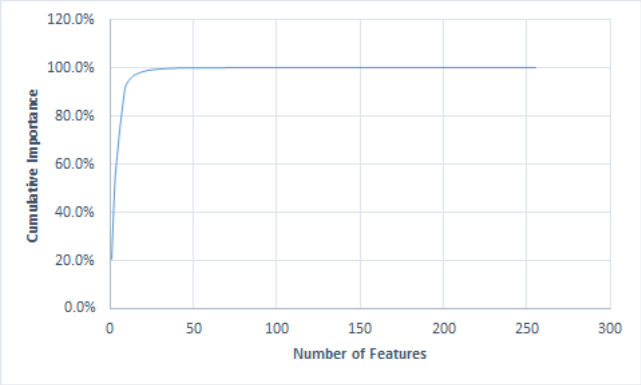

Figure 9: Effect of the number of features on the overall error reduction. The top 10 features contribute ≈ 93% to the reduction in training error.

9.7 Stratified sampling vs. Synthesizing prefixes

For conciseness we do not do a thorough comparison of using stratified sampling versus synthesizing prefixes in training data generation. In the experiments conducted, using stratified sampling in conjunction with simple random sampling reduced the *filter-set* gap by ≈ 2.5%. Synthesizing lower length prefixes and using them with the prefixes at click, or in a standalone manner, produced small metric gains (≈ 1% reduction in the metric gap with the *filter-set*). However, using this approach resulted in improvements on some and regressions on other query segments (for example, improvements on Business intent queries, and regression on Place intent queries). To counter this, one has to resort to either assigning training example weights, or using stratified sampling.

9.8 Performance of GBDT Variants

We next examined the prediction accuracy of the various publicly available GBDT implementations for ranking. The models considered were, LightGBM, XGBoost, and CatBoost. The training set to evaluate these models comprised of 200,000 clicked examples. For evaluation, the number of trees in these models was fixed at 500 and the learning rate was set to 0.3. The tree depth parameter was the only parameter that was varied in the interval [2, 12]. The CatBoost model was trained using the *YetiRankPairwise* loss function implemented in the library.

Figure 10 shows the prediction accuracy of these models as a function of tree depth. On the training-evaluation set pair and hyperparameter configuration considered, LightGBM produced the best NCG score. The NCG scores of the models were normalized using the best score, by computing the absolute deviations and dividing by the best score. From the figure it is seen that the prediction accuracies of LightGBM and XGBoost are comparable, with

the best NCG score occurring at a tree depth of 7. At higher tree depths, the prediction accuracy of LightGBM degrades much more rapidly than XGBoost. This trend may be attributed to the built-in regularization in XGBoost, which results in more robust models. LightGBM and XGBoost perform better than CatBoost in these experiments.

In terms of training time, the experiments were run on slightly different machines, and so quantitative statements cannot be made. However, qualitatively speaking, LightGBM was the fastest, while XGBoost was the slowest of the models considered.

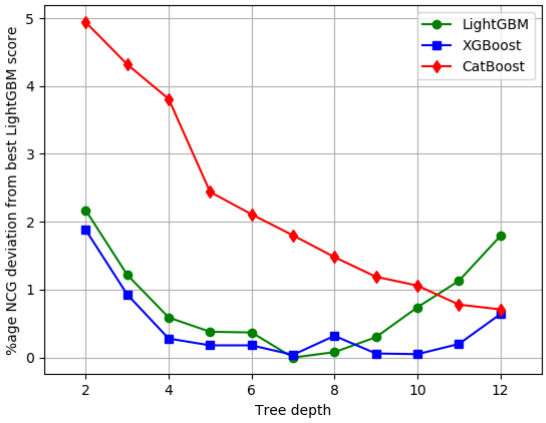

Figure 10: Prediction accuracy of LightGBM, XGBoost, and CatBoost as function of tree depth. The percentage deviations from the best LightGBM score are plotted on the *y*-axis.

9.9 Training Set Size

Figure 11 shows effect of the training set size on the (NCG) *filter-set* gap. In the plot, the percentage gains achieved relative to using a a training set with 200,000 clicked examples are shown. For these experiments, the LambdaMART model with a tree depth of 7 was used. From the figure it is seen that larger training sets result in improvement in test metrics (by $\approx 1.6\%$ in comparison to using a 200k set), with diminishing returns after about 3 million examples. From the figure it is also seen that the improvements are not a monotonic function of training set sizes. This behavior may be attributed to the sampling variance induced by random sampling of the training data.

9.10 Training Set Instance

The training set instance sampled could in itself be viewed as a hyperparameter. It may argued that, intuitively, the effect of the training set instance on metric gains should decrease with increasing set-sizes (since larger training sets generalize better). It could also be argued that even if a particular instance produces slightly better offline metrics (on a particular evaluation set), it may not translate to statistically significant online gains. As much as these arguments are intuitive, sometimes, some of such gains could actually translate to online improvements, and therefore it is beneficial to treat the training data instance as a hyperparameter.

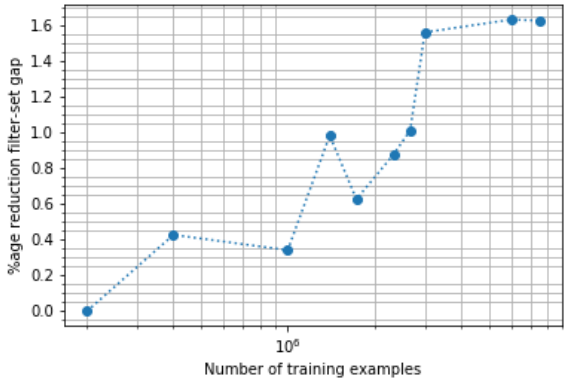

Figure 11: Effect of training set sizes on NCG gains relative to the filter-set and a training set of size 200,000.

To demonstrate the effect of training set instances, models were trained using ten sets each of size 100,000 (100k), 200k, and 300k. The evaluation set over which the models were evaluated was fixed, and was of size 10k. Shown in Figure 12 are the results from the experiments. In Figure 12, the percentage increases in *filter-set* gap, as a result of not using the best model, are reported. From the figure the following observations can be made: a) the model that performs the best occurs for a training set of size 200k, which is likely due to the small number of trials conducted, b) as hypothesized, an increase in the training set size results in slightly better metric scores, and also the variance in the scores is lower.

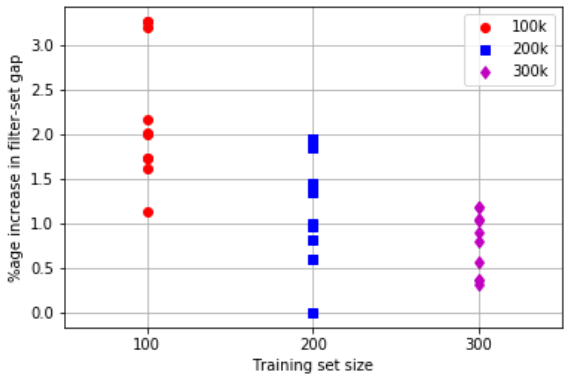

Figure 12: Percentage variation in filter-set gap relative to the best model. Models were trained using sets of sizes 100k, 200k, and 300k.

9.11 Ensembling Models

The work of [25] provides a good high-level description, and suggests approaches for ensembling models for click prediction problems that arise in sponsored search. In their work, they considered Bagging, Boosting, Stacking, and Cascading approaches for ensemble generation, and found boosting DNN's with GBDT to produce the best results. For the current work, only Bagging, Stacking, and Cascading were considered. To apply these approaches, different

models using GBDT's and DNN's were trained. The models differed in the training data, learning algorithms, and the loss functions used (for example, different GBDT models were trained using pointwise and pairwise loss functions, solving a regression problem, altering proportion of the various entity-types in the training data, etc.). For bagging, the normalized ranker scores were averaged. For stacking, a meta-ranker based on a GBDT was trained using model scores from the other models as the input features. For cascading, the model scores of the other models were used as input features in a particular GBDT model.

In the experiments conducted, none of the three approaches (bagging, stacking, cascading) could significantly outperform a well-optimized boosted decision tree. In terms of their relative performance, stacking produced the worst results. Cascading produced ensembles that were not much better than a well-optimized GBDT, and was the best of the three approaches. The poor performance of stacking could be attributed to the high correlation (≈ 0.93) between ranker scores of the various models, and such behavior is reported in [9, 22]. The potential benefit of using Boosting remains to be investigated. However, it should be noted that for services such as QAC, the latency budget for ranking is highly constrained (on the order of a few milliseconds). This poses practical limitations on the degree to which models can be ensembled.

9.12 Data freshness and Distribution drift

Data freshness is a critical factor for systems such as QAC for at least three reasons:

- Newer training data helps capture changing user-search patterns.
- If the training data distribution was altered in the previous ranker version to achieve a desired effect, then the newer data would capture the intended effect in real user conversations. Training on the newer data could then help either reinforce or take-forward the desired effect.
- As the relevance of the search suggestions improves over time, users may engage with the QAC service at lower prefix lengths (as they gain confidence (and build expectation) in the search suggestions). Training on newer data could therefore reduce the need to manually adjust the training set distribution, to help the system see more examples at lower prefix lengths.

In our experience, training on the latest logs almost always produced slightly better metrics than training on older logs. This could be due to the above-mentioned factors, as well as distribution drift. There are several ways to detect and quantify distribution drift. For a detailed discussion, the interested reader may refer to [35].

To quantify the impact of distribution drift in Maps QAC, rankers were trained and evaluated using monthly data between the time periods September 2017 through May 2019. The ranker trained using a particular training set (say, September 2018) was evaluated both forward and backward in time (using the monthly evaluation sets). The training and evaluation set sizes were 200,000 and 10,000 clicked examples respectively. For conciseness, the impact of using an year old ranker on the *filter-set* gap is shown in Table 2. For instance, in Table 2, the 9.91% *filter-set* gap increase for September 2018 evaluation set is reflective of the deterioration in the ranking quality as a result of using a September 2017 ranker (in September 2018). An average value of $\approx$ 7.87% per-year (and $\approx$ 0.66% per-month) indicates that, either online learning or ranker re-training could produce metric gains in comparison to using a stale ranker.

Table 2: Impact of using an year old ranker on the *filter-set* gap

.

Evaluation set	Filter-set gap increase
September 2018	9.91%
October 2018	11.62%
November 2018	9.66%
December 2018	6.93%
January 2019	6.80%
February 2019	8.39%
March 2019	5.16%
April 2019	7.36%
May 2019	5.04%
Average	7..87%

Table 3: Approximate impact of various factors on *filter-set* gap reduction.

.

Factor	Filter-set gains
Advanced static ranks	14.67%
Hyperparameter tuning	6.7%
Improved features	3.33%
Model type (LightGBM vs others)	3.00%
Training data generation scheme	2.67%
Training set size	2.00%
New (low explanatory power) features	1.67%
Training data instance	1.33%
Negative data sampling	1.00%
Distribution drift (per-month)	0.66%
Ensembling models (given latency constraints)	$\approx$ 0.0%

9.13 Overall Improvements

Table 3 summarizes at a very high-level, the approximate impact of the various factors considered on metric improvements, on a specific instance of a Maps QAC ranking problem. It is assumed that these factors are applied sequentially, starting from the baseline model listed in Table 1. From the table it is seen while new, improved, and advanced features contribute to $\approx$ 53.15% of the metric improvements achieved, if one were to consider the various aspects related to the training set as hyperparameters, then, model selection and hyperparameter tuning (model + data) contributes $\approx$ 45%. The table assumes that the baseline model comprises of features with reasonably good explanatory power.

10 CHALLENGES

This section outlines some of the challenges with the QAC click prediction problem.

10.1 Trading-off between metrics

Different metrics measure different aspects of the end-user experience, and different ranking components are optimized to different metrics. Therefore, quite often, singularly optimizing on offline click metric gains could cause regressions in other metrics. To mitigate this, typically, some of the offline click metric gains are traded to maintain other metrics at acceptable thresholds. This poses a challenge in automating the model selection and hyperparameter tuning process, as it may lead to unintended consequences.

10.2 Discrepancy between training and evaluation metrics

A key challenge with ranking is the difference between the evaluation measures on the training and test sets. Ranking models typically use NDCG as the evaluation measure on the training set, while specialized metrics such as NCG are used on the evaluation set. A possible way to mitigate this discrepancy could be through weighted loss functions, as suggested in [38]. This could introduce additional hyperparameters into the learning process.

11 CONCLUSIONS

In the work presented, insights are shared from ranker experiments performed in the context of Query Auto-Completion (QAC) on Maps. The prerequisite to work on ranker improvements is the presence of a significant gap between the metrics on *filter-set* and over the top 5 results. For Maps QAC ranking problem, coming up with new, improved, and advanced features (unsurprisingly) resulted in the highest metric gains. Deep neural networks (DNN's) and boosted decision trees (GBDT) were considered in ranking. In the experiments conducted, GBDT's performed slightly better than standalone DNN's. Of the various GBDT implementations considered, LightGBM had the highest prediction accuracy. It is seen that in addition to performing conventional model selection and hyperparameter tuning, treating the various aspects of the training data generation process as hyperparameters could lead to significant metric gains. It is also shown that the performance of a QAC ranking model steadily degrades over time due to *concept-drift*. This can be addressed through either regular re-training or online learning.

REFERENCES

[1] Qingyao Ai, Xuanhui Wang, Nadav Golbandi, Michael Bendersky, and Marc Najork. 2018. Learning Groupwise Scoring Functions Using Deep Neural Networks. *CoRR* abs/1811.04415 (2018). arXiv:1811.04415

[2] Afroze Ibrahim Baqapuri and Ilya Trofimov. 2014. Using Neural Networks for Click Prediction of Sponsored Search. *CoRR* abs/1412.6601 (2014). arXiv:1412.6601

[3] Ziv Bar-Yossef and Naama Kraus. 2011. Context-sensitive Query Auto-completion. In *Proceedings of the 20th International Conference on World Wide Web (WWW '11)*. ACM, New York, NY, USA, 107–116.

[4] C.M. Bishop. 2013. *Pattern Recognition and Machine Learning*. Springer (India) Private Limited.

[5] Andrei Z. Broder, Peter Ciccolo, Marcus Fontoura, Evgeniy Gabrilovich, Vanja Josifovski, and Lance Riedel. 2008. Search Advertising Using Web Relevance Feedback. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management (CIKM '08)*. ACM, New York, NY, USA, 1013–1022.

[6] Christopher J. C. Burges, Krysta M. Svore, Paul N. Bennett, Andrzej Pastusiak, and Qiang Wu. 2010. Learning to Rank Using an Ensemble of Lambda-gradient Models. In *Proceedings of the 2010 International Conference on Yahoo! Learning to Rank Challenge - Volume 14 (YLRC'10)*. JMLR.org, 25–35.

[7] Fei Cai, Ridho Reinanda, and Maarten De Rijke. 2016. Diversifying Query Auto-Completion. *ACM Trans. Inf. Syst.* 34, 4, Article 25 (June 2016), 33 pages.

[8] Rich Caruana, Nikos Karampatziakis, and Ainur Yessenalina. 2008. An Empirical Evaluation of Supervised Learning in High Dimensions. In *Proceedings of the 25th International Conference on Machine Learning (ICML '08)*. ACM, New York, NY, USA, 96–103.

[9] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. 2004. Ensemble Selection from Libraries of Models. In *Proceedings of the Twenty-first International Conference on Machine Learning (ICML '04)*. ACM, New York, NY, USA, 18–.

[10] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. ACM, New York, NY, USA, 785–794.

[11] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. 2018. CatBoost: gradient boosting with categorical features support. *CoRR* abs/1810.11363 (2018). arXiv:1810.11363

[12] Anna Veronika Dorogush, Andrey Gulin, Gleb Gusev, Nikita Kazeev, Liudmila Ostroumova Prokhorenkova, and Aleksandr Vorobev. 2017. Fighting biases with dynamic boosting. *CoRR* abs/1706.09516 (2017). arXiv:1706.09516

[13] William Fithian and Trevor Hastie. 2014. Local case-control sampling: Efficient subsampling in imbalanced data sets. *Ann. Statist.* 42, 5 (10 2014), 1693–1724.

[14] Thore Graepel, Joaquin Quiñonero Candela, Thomas Borchert, and Ralf Herbrich. 2010. Web-scale Bayesian Click-through Rate Prediction for Sponsored Search Advertising in Microsoft's Bing Search Engine. In *Proceedings of the 27th International Conference on International Conference on Machine Learning (ICML'10)*. Omnipress, USA, 13–20.

[15] Thore Graepel, Joaquin Quiñonero Candela, Thomas Borchert, and Ralf Herbrich. 2010. Web-Scale Bayesian Click-Through Rate Prediction for Sponsored Search Advertising in Microsoft's Bing Search Engine. In *Proceedings of the 27th International Conference on Machine Learning ICML 2010, Invited Applications Track (unreviewed, to appear)* (proceedings of the 27th international conference on machine learning icml 2010, invited applications track (unreviewed, to appear) ed.). Invited Applications Track.

[16] T. Hastie, R. Tibshirani, and J.H. Friedman. 2009. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer.

[17] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, and Joaquin Quiñonero Candela. 2014. Practical Lessons from Predicting Clicks on Ads at Facebook. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising (ADKDD'14)*. ACM, New York, NY, USA, Article 5, 9 pages.

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

[19] G. James, D. Witten, T. Hastie, and R. Tibshirani. 2014. *An Introduction to Statistical Learning: with Applications in R*. Springer New York.

[20] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Advances in Neural Information Processing Systems 30*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.). Curran Associates, Inc., 3146–3154.

[21] Eugene Kharitonov, Craig Macdonald, Pavel Serdyukov, and Iadh Ounis. 2013. User Model-based Metrics for Offline Query Suggestion Evaluation. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '13)*. ACM, New York, NY, USA, 633–642.

[22] L.I. Kuncheva. 2004. *Combining Pattern Classifiers: Methods and Algorithms*. Wiley.

[23] H. Li. 2014. *Learning to Rank for Information Retrieval and Natural Language Processing: Second Edition.* Morgan & Claypool Publishers.

[24] Yanen Li, Anlei Dong, Hongning Wang, Hongbo Deng, Yi Chang, and ChengXiang Zhai. 2014. A Two-dimensional Click Model for Query Auto-completion. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '14).* ACM, New York, NY, USA, 455–464.

[25] Xiaoliang Ling, Weiwei Deng, Chen Gu, Hucheng Zhou, Cui Li, and Feng Sun. 2017. Model Ensemble for Click Prediction in Bing Search Ads. In *Proceedings of the 26th International Conference on World Wide Web Companion (WWW '17 Companion).* International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 689–698.

[26] Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Found. Trends Inf. Retr.* 3, 3 (March 2009), 225–331.

[27] Bhaskar Mitra and Nick Craswell. 2015. Query Auto-Completion for Rare Prefixes. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management (CIKM '15).* ACM, New York, NY, USA, 1755–1758.

[28] Bhaskar Mitra, Milad Shokouhi, Filip Radlinski, and Katja Hofmann. 2014. On User Interactions with Query Auto-completion. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '14).* ACM, New York, NY, USA, 1055–1058.

[29] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. 2012. *Foundations of Machine Learning.* The MIT Press.

[30] K.P. Murphy. 2012. *Machine Learning: A Probabilistic Perspective.* MIT Press.

[31] Rama Kumar Pasumarthi, Xuanhui Wang, Cheng Li, Sebastian Bruch, Michael Bendersky, Marc Najork, Jan Pfeifer, Nadav Golbandi, Rohan Anil, and Stephan Wolf. 2018. TF-Ranking: Scalable TensorFlow Library for Learning-to-Rank. *CoRR* abs/1812.00073 (2018). arXiv:1812.00073

[32] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. 2018. CatBoost: unbiased boosting with categorical features. In *Advances in Neural Information Processing Systems 31*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (Eds.). Curran Associates, Inc., 6638–6648.

[33] Milad Shokouhi. 2013. Learning to Personalize Query Auto-completion. In *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '13).* ACM, New York, NY, USA, 103–112.

[34] Ilya Trofimov, Anna Kornetova, and Valery Topinskiy. 2012. Using Boosted Trees for Click-through Rate Prediction for Sponsored Search. In *Proceedings of the Sixth International Workshop on Data Mining for Online Advertising and Internet Economy (ADKDD '12).* ACM, New York, NY, USA, Article 2, 6 pages.

[35] Geoffrey I. Webb, Roy Hyde, Hong Cao, Hai Long Nguyen, and Francois Petitjean. 2016. Characterizing Concept Drift. *Data Min. Knowl. Discov.* 30, 4 (July 2016), 964–994.

[36] Dragomir Yankov, Pavel Berkhin, and Lihong Li. 2015. Evaluation of Explore-Exploit Policies in Multi-result Ranking Systems. *CoRR* abs/1504.07662 (2015). arXiv:1504.07662

[37] Bianca Zadrozny and Charles Elkan. 2001. Obtaining Calibrated Probability Estimates from Decision Trees and Naive Bayesian Classifiers. In *Proceedings of the Eighteenth International Conference on Machine Learning (ICML '01).* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 609–616.

[38] Sen Zhao, Mahdi Milani Fard, Harikrishna Narasimhan, and Maya Gupta. 2019. Metric-Optimized Example Weights. In *Proceedings of the 36th International Conference on Machine Learning (Proceedings of Machine Learning Research)*, Kamalika Chaudhuri and Ruslan Salakhutdinov (Eds.), Vol. 97. PMLR, Long Beach, California, USA, 7533–7542.