Relevance Ranking for Real-Time Tweet Search[*]

Yan Xia
Facebook
@yanxia_wh

Yu Sun
Twitter
@aldrichyusun

Tian Wang
Twitter
@wangtian

Juan Caicedo Carvajal
Airbnb
@cavorite

Jinliang Fan
Coupang
@jinliangfan

Bhargav Mangipudi
Twitter
@bhargav91

Lisa Huang
Youtube
@lxyhuang

Yatharth Saraf
Facebook
@ysaraf

ABSTRACT

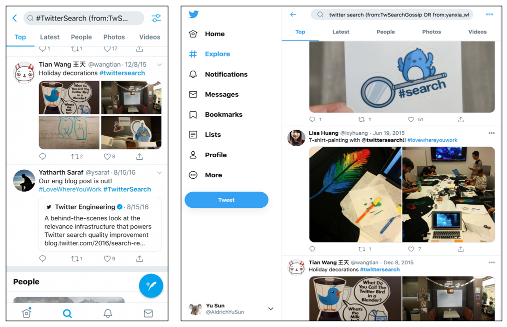
Relevance ranking is a key component of many search engines, including the Tweet search engine at Twitter. Users often use Tweet search to discover live discussions and different voices on trending topics or recent events. Tweet search is thus unique due to its focus on real-time content, where both the retrieved content and queries change drastically on an hourly basis. Another important property of Tweet search is that its relevance ranking takes the social endorsements from other users into account, e.g., "likes" and "retweets", which is different from mainly relying on clicks as implicit feedback. The relevance ranking of Tweet search is also subject to strict latency constraints, because every second, a large amount of Tweets are posted and indexed, while tens of thousands of queries are issued to search posted Tweets. Considering the above properties and constraints, we present a relevance ranking system for Tweet search addressing all these challenges at Twitter. We first discuss the formation of the relevance ranking pipeline, which consists of a series of ranking models. We then present the methodology for training the models and the various groups of features we use, including real-time and personalized features. We also investigate approaches of achieving unbiased model training and building up automatic online tuning of system parameters. Experiments using online A/B testing demonstrate the effectiveness of the proposed approaches and we have deployed the proposed relevance ranking system in production for more than three years.

CCS CONCEPTS

• **Information systems → Social networks**; **Learning to rank**; **Web and social media search**; *Recommender systems.*

ACM Reference Format:
Yan Xia, Yu Sun, Tian Wang, Juan Caicedo Carvajal, Jinliang Fan, Bhargav Mangipudi, Lisa Huang, and Yatharth Saraf. 2020. Relevance Ranking for Real-Time Tweet Search. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20), October 19–23, 2020, Virtual Event, Ireland.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3340531.3412743

[*]Work was done while all authors were at Twitter.



Tian Wang 王天 @wangtian · 12/8/15
Holiday decorations #twittersearch

Yatharth Saraf @ysaraf · 8/15/16
Our eng blog post is out!
#LoveWhereYouWork #TwitterSearch

Twitter Engineering · 8/15/16
A behind-the-scenes look at the relevance infrastructure that powers Twitter search quality improvement
blog.twitter.com/2016/search-re...

Lisa Huang @lxyhuang · Jun 19, 2015
T-shirt-painting with @twittersearch!! #lovewhereyouwork

Tian Wang 王天 @wangtian · Dec 8, 2015
Holiday decorations #twittersearch

Figure 1: Tweet search

1 INTRODUCTION

Search engine is an indispensable part of large-scale online platforms. Being able to search and discover content is key to the success of many social media platforms, including Twitter. Different from other search engines, such as web search [4], e-commerce search [11], personal file or email search [23], a key focus of Twitter search is enabling users to discover real-time information, i.e., to learn what is currently happening. There are many searchable entities at Twitter, including Tweets, users, and events. A vital type is Tweet search, which given a keyword-based search query returns a ranked list of Tweets containing the query keywords. Figure 1 illustrates examples of Tweet search from mobile and web clients. The search query can contain any keywords users intend to search for, which can be trending topics, hashtags, emojis, locations, or URLs. Similar to informational queries for web search [9], a large portion of the search queries intend to explore different voices discussing topics closely related to the query keyword(s), i.e., exploratory queries. Normally, there will be hundreds of thousands of Tweets matching (i.e., containing) the keywords of an exploratory query. Therefore, an effective ranking on the matched Tweets is the key to present both timely and relevant results. In this paper, we focus on such relevance ranking for exploratory Tweet search.

Relevance ranking for Tweet search is quite different from other types of searches. One major difference is the real-time nature of Tweet search. Every second, thousands of new Tweets are posted. These new Tweets are indexed and ready to be served as search results in a few seconds. This means that the content of retrieved results is constantly changing and the relevance ranking should be able to handle such rapid changes and serve fresh results. Similar to the change of content, the majority of queries users search for also shift from one topic to another within a few hours, especially when there are breaking news or live events. This creates significant challenges to perform effective relevance ranking for Tweet search.

Apart from the rapid change of searchable content and search queries, another major difference is the ranking objective. Since Twitter is a social media platform, endorsements through social actions such as "likes" or "retweets" are strong signals indicating result relevance. It is beneficial that we optimize the relevance ranking towards both social endorsement and click-through-rate. This further differentiates Tweet search from other types of searches such as web search or e-commerce search.

Another significant challenge for Tweet search relevance ranking is the strict constraint of low latency for serving online search traffic. There are tens of thousands of queries entering into the Tweet search engine every second, all of which should obtain search results within a low-digit hundreds of milliseconds. This requires us to carefully design the relevance ranking to comply with such tight latency constraints and achieve both efficient and effective relevance computation.

The main contribution is that we present a large-scale relevance ranking system used in production, which considers and addresses all the above properties and requirements of an industrial information retrieval system. In the rest of this paper, we first briefly introduce search at Twitter and the search engine components in Section 2, and then present the ranking pipeline for Tweet search in Section 3. We discuss removing biases when collecting feedback signals for model training in Section 4. We also investigate building online feedback loop and tuning system parameters automatically in Section 5. We conduct experiments on all the proposed techniques using online A/B tests and have productionized the relevance ranking system at Twitter for more than three years.

2 TWEET SEARCH RANKING SYSTEM

Many users employ Twitter search to stay up-to-date about breaking news and their interests. It is challenging, however, to do this job well at Twitter's scale. In this section, we briefly introduce Twitter's search architecture and discuss the characteristics that enable relevance Tweet ranking with low latency and high throughput.

Twitter's search APIs serve tens of thousands requests per second within hundreds of milliseconds. Millions of Tweets are created every day. Newly created Tweets are searchable under a few seconds. We achieve such scalability and real-time availability via a carefully designed system architecture and ranking pipeline. Specifically, the backbone services for Tweet search ranking are *Blender* and *Earlybird*. Upon receiving a search request from clients, Blender processes the request and sends retrieval requests to Earlybird. (In fact, Blender calls various indices to retrieve different types of results, *e.g*. Tweets, accounts, related queries, during this process.) Earlybird queries various tiers of Tweet index clusters and returns the retrieved candidates to Blender. Upon receiving the candidates from Earlybird (and many other services), Blender performs hydration (*i.e.* retrieving the candidates' complete details and content), ranking, and finally "blending" of heterogeneous candidate results into single unified timeline, which is returned back to clients.

Blender is a stateless service, and as such, it is horizontally scalable and provides good throughput and fault tolerance. Being one of the most complicated services at Twitter, Blender makes more than thirty calls to different services with complex interdependencies. The robustness of Blender is achieved via the Nodes library [21], which coordinates the data dependency among processes with high performance and concurrency. It helps reduce Blender's latency by executing independent requests in parallel.

Earlybird is built on top of the open-source Lucene search engine [2]. To meet the requirements above, Earlybird is highly customized [5]. Earlybird servers are organized in a broker-coordinated, document-partitioned, replicated, and distributed architecture. A broker is responsible for forwarding requests to the partition servers and integrating partial results from each before passing the final output back to Blender. Partitioning reduces ranking and indexing latency since each partition operates independently. Each partition is replicated into multiple servers, which increases the throughput in a linearly scalable fashion. In addition to these conventional techniques, Earlybird has significant customization to support the demand of real-time search, including an unique way of organizing the inverted index and an efficient algorithm to perform real-time indexing (write) with highly-concurrent retrieval and ranking (read). Finally, caching is heavily used in both Blender and Earlybird to further reduce the retrieval of frequent queries and popular Tweets.

For each incoming search request, we evaluate tens of thousands of potential candidates to identify the best Tweets to serve to the user. To achieve such scalability, the ranking pipeline employs the scatter-gather pattern and organizes the retrieval into three stages: candidate selection, light ranking, and heavy ranking. As illustrated in Figure 2, a Blender server sends one search request to an Earlybird broker, who "scatters" the search to multiple partitioned Earlybird servers. The first two stages occur at the level of individual Earlybird server. At the candidate selection stage, we retrieve Tweets that match the search query (i.e., contain the query keywords) from the inverted index. The size of this pool is on the order of tens of thousands. A set of fast and light ranking models using limited features (available in Earlybird) are then applied to choose the top-20 potentially-relevant Tweets from this pool. The Earlybird broker gathers results from different partitions and returns a total of around four hundred Tweets back to the calling Blender server, in which a set of slower and heavy ranking models are used to finalize the top-20 most relevant results for a user. Candidates are funneled gradually to achieve a good balance between low latency and high ranking quality. Furthermore, product policy layers are applied to the selected results after both the light and heavy ranking stages to remove obviously bad Tweets and ensure high-standard search quality. These policy layers consist of rule-based heuristics and many are threshold-based filtering. Examples include filtering or down-ranking of pornographic and abusive Tweets.

In the search result page, we collect users' feedback on the search results. The feedback is used in training the ranking models in both the light and heavy stages. The feedback is also used for the automated tuning of hyperparameters in the ranking pipeline. We will discuss these in Sections 3 and 5, respectively.

3 RELEVANCE RANKING MODELS

3.1 Optimization Objectives

The optimization objective for the ranking pipeline is to improve the overall search result quality and relevance.

Relevance indicators. From a short-term perspective, the quality and relevance is indicated from users' implicit feedback, i.e., the

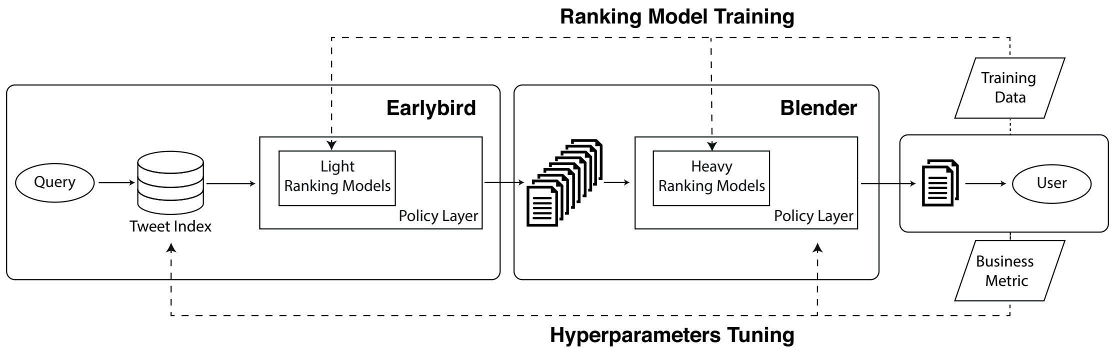


Figure 2: Relevance ranking pipeline for Tweet search

volume of online engagement the search results induce. The volume of online engagement can be quantitatively measured by the amount of clicks, social actions (e.g., likes, retweet and replies), long linger, etc. on the search results. Besides such implicit feedback, we also conduct periodic human evaluation of the search results for randomly sampled search queries to collect explicit feedback. From a long-term perspective, the quality and relevance is reflected by users' happiness and daily usage of the Twitter search functionality.

However, it is challenging to combine all these measurements into one composite objective from which we train the ranking models. Besides the difference and overlap among these measurements, we also need to address the problems that (1) the models at different stages in the ranking pipeline have different purposes, and (2) the available supervised training signals/labels for different evaluation metrics are severely skewed with the largest volume from online engagement and smallest from human evaluation and users' daily usage. Therefore, for simplicity and robustness, we train the ranking pipeline towards optimizing the online engagements in terms of clicks and social actions.

Separate clicks and social actions. We treat clicks and social actions as different objectives, because the reasons a Tweet induces a click or social action are quite different. For example, results with links are more likely to be clicked but not necessarily lead to social actions like retweet; while results that can sparkle discussions or come from friends are more likely to attract social actions like replies, but not clicks. Therefore, we treat clicks and social actions separately, i.e., use them as different types of engagement labels.

3.2 Ranking Features

For an effective ranking, we aim to have features that capture different facets of relevance, including popularity, personalized preferences, and timeliness. Therefore, we obtain features from the query, searcher, Tweet, author, and the interactions between them.

Static features. At the Tweet creation time, we extract various attributes of the Tweet, such as the Tweet creation time, whether the Tweet contains URLs, the influence of its author, and the language of the Tweet. These creation-time features are static (or immutable) because Tweets are, by design, immutable objects. Since there are no race conditions or concurrent update issues, we can use the static features with high efficiency and maintainability.

Real-time features. To fulfill the real-time nature of Tweet search, we also rely heavily on features that are updated in real time, capturing the global popularity and virality. The real-time features are mostly aggregated counts related to impressions and engagements, such as the number of social actions a Tweet receives, the number of views on a video in a Tweet, or the number of impressions on a Tweet if it is embedded into a web page. We use a real-time stream processing system built on top of Apache Storm [3] to tackle the large amount of impressions and engagements continuously generated from different Tweet viewing and searching platforms. We update this set of features in real time and use them in both the Earlybird and Blender ranking stages.

Query features. The search query itself carries much relevance information. To get real-time knowledge about a query, we compute running statistics of the queries entered into the Tweet search system. The running statistics are obtained from the aggregates of the query keywords, the served results, the searchers issuing the query, and the searchers' actions on the served results. Examples of such aggregates include the number of times a query is searched, the language distribution of the result Tweets for this query, the interests of the searchers issuing this query, etc. We also obtain static features derived from a query, such as the query time and the query source (e.g., explicitly typed text or redirected from clicking on a hashtag in Tweets). We also collect features to account for the interactions between a query and a Tweet. For example, for a single term query, a Tweet can match the query term one or multiple times in different fields such as the Tweet text, the account name of the Tweet author, the webpage title of the URL the Tweet contains, or the hashtags (or cashtags) in the Tweet. For a multi-term query, the order and distance of the matched keywords also indicate the result's relevance.

Personalization features. To achieve high personalization, we also use features from the follow graph (i.e., the graph formed by the follow relationship where vertices are users and directed edges point from a follower to a followee). We use the features measuring the proximity and similarity between the searcher and Tweet author, which are derived from direct follow, triangle relationship, paths with more than two hops, or detected communities based on common interests. We also use a set of real-time counting features to capture the short term user preference to different types of content, e.g., a counting feature can be the number of Tweets a searcher engaged with that are from her followers in the past hour.

3.3 Earlybird Models

Candidate selection. In Earlybird, we first use multiple inverted indices to find a large pool of Tweets matching the query keywords.

To achieve low latency, we employ *early termination* on each Earlybird server, i.e., stop the retrieval once we have collected a max number of Tweets (e.g., 750 in production) from one partition of the inverted index. During this process, we apply basic ranking heuristics using classic techniques such as term frequency, inverse document frequency, and document length normalization. Due to the Earlybird's customization, we retrieve Tweets in reverse chronological order. Therefore, fresh Tweets have a higher chance to be surfaced. For each Tweet, Earlybird also returns a set of attributes, which are used as features for the subsequent ranking steps.

Light ranking. The number of Tweets returned from the previous step can be very large (e.g., $16,500 = 750 \times 22$ shards in production), even though we apply early termination. Sending all these Tweets to Blender to rank will significantly slow down the processing speed. Thus, we first perform a round of light ranking on this Tweet pool. Using Tweet features available in Earlybird and a subset of query and searcher properties that are not expensive to obtain as features, the light ranking is accomplished by two logistic regression models. The two models' objectives are to predict the probability that a Tweet will induce social actions and clicks, respectively. As discussed in Section 3.1, the reason we use two separate models is to account for the different causes for clicks and social actions. Another reason is to avoid the considerable imbalance between the two signals for training one model, where the amount of clicks is much larger than social actions. At training time, the loss function for both models is cross-entropy. At inference time, the Tweets are ranked by a weighted sum of the predicted probabilities from the two models. Recall that this light ranking step occurs on each Earlybird server shard and the top-20 Tweets from each server are gathered by the broker and returned to Blender.

The goal of Earlybird stage ranking is to select a promising candidate set that consists of hundreds of Tweets having both fresh content and keyword match. We use logistic regression models due to their high efficiency, which is a trade-off of accuracy and latency.

3.4 Blender Models

Heavy ranking with more features. In Blender, we rerank the candidates selected by Earlybird and select a even smaller set of Tweets for the next stage. Since the number of candidates is much smaller, we can afford to use the higher capacity model with considerably more expressiveness. The model we use in this step is a multilayer perceptron (MLP) with two hidden layers of size 100×100 and aims to utilize more personalization signals and capture nonlinear and complex relations among the query, user, and Tweets. We accumulate more features from different sources as discussed in Section 3.2. We train this Blender MLP model to only optimize for social actions, because the cause and pattern for social actions is more complex and harder to represent. The loss function for this model is cross-entropy. At inference time, Tweets having the highest predicted probability of inducing social actions are sent to the next content relevance reranking step.

Content relevance reranking. The term-based method in the candidate selection step helps to maintain the content relevance between the query and Tweet to some extent, yet such content relevance is often overshadowed by the influence of the previous two ranking steps, which prefer Tweets that are popular and engaging but can be off-topic. Therefore, in the last step, we reexamine the content relevance by checking the closeness of the query with the Tweets in a same semantic space. In particular, we model the query and Tweets as fixed dimension embedding vectors. We obtain the embedding vector of a query or a Tweet by a simple average on the word-level embeddings [13] trained on the Tweet corpus. We then concatenate and feed the query and Tweet embedding vectors to another MLP with one hidden layer of size 100, which is trained using signals indicating whether the Tweet content is relevant to the query (more details in Section 3.5). Due to the aforementioned strict latency constraint, we can only afford to examine a small amount of the top results (e.g., top-30 in production) and rerank these results by their content relevance.

3.5 Training Data Generation

We obtain the training data for all the aforementioned models through *search sessions*. A search session contains search queries from a specific user and their corresponding results. We denote by $S = \{r^1_{q_1}, r^2_{q_1}, \ldots, r^s_{q_1}, \ldots, r^1_{q_n}, r^2_{q_n}, \ldots, r^t_{q_n}\}$ a search session, where $r^j_{q_i}$ denotes the jth result for the ith query. A search session also keeps record of which results receive a click or social action.

Reduce noise in training labels. The positive labels (clicks or social actions) are implicit relevance feedback, which can be quite noisy due to, e.g., accidental clicks, automated scripts, or other inconsistencies during the logging process. Therefore, we clean the positive labels by thresholding a Tweet's engagement rate for a given query, i.e., aggregating the same query-Tweet pairs in all collected search sessions and only use the pair as a positive example if the Tweet leads to engagement in many search sessions. Specifically, let q and r_q be a query and a corresponding Tweet result, respectively. Let S^r_q be the set of search sessions containing both the query q and the result r_q. Let $\mathcal{E}(r_q)$ be the indicator function where $\mathcal{E}(r_q) = 1$ if r_q is engaged and 0 otherwise. A positive-labeled instance r_q is only kept if the ratio of engagement across all search sessions is above a predefined threshold δ, i.e,

$$\frac{|\{r_q \in S^r_q \mid \mathcal{E}(r_q) = 1\}}{|S^r_q|} > \delta \ .$$

When this condition is met, in order to further enhance the positive labels, the same results $\{r_q \in S^r_q \mid \mathcal{E}(r_q) = 0\}$ receiving no engagement are disregarded. Similarly, when the thresholding condition is not met, all the results even leading to engagements (uncleaned positive examples), i.e., $\{r_q \in S^r_q \mid \mathcal{E}(r_q) = 1\}$, are dropped from the training data.

The engagement rate threshold serves as a knob for tuning the signal-to-noise ratio of the training data. A higher threshold can remove more accidental noise but incur the loss of personalization signals, because a Tweet can be engaged by only a few friends, close followers, or users with niche interests. As a result, the positive labels will focus more on popular content. In the ranking pipeline, we use a higher threshold for Earlybird models in accordance with the latency constraint, since retrieving more personalized content requires more personalization features together with a model with higher complexity, which will cause significant latency increase. For the Blender MLP model, we use a lower threshold to achieve a good balance between personalized and globally popular content.

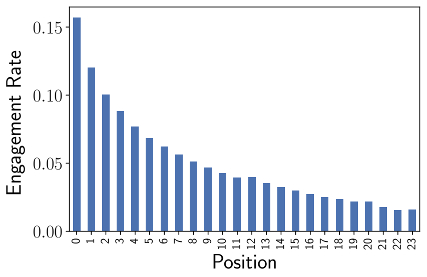


Figure 3: Engagement rate at different positions

For the content relevance model, we use the engagement rate to indicate whether the content of a Tweet is relevant to a query. The rationale is that if a query-Tweet pair has a high empirical engagement rate, the Tweet should also be semantically relevant to that query, whereas pairs with a low rate are likely off-topic.

4 BIAS CORRECTION

Bias from multistage ranking. When collecting the training data, we have two main sources of biases: selection and position biases.

Selection bias exists in both individual model retraining and among models in the ranking pipeline. This bias can be more severe for the models further away from the label collection step (i.e., collecting clicks or social actions). Due to the selection bias, the training data we collect is a biased sample of the whole Tweet population [12] since the data come from the selection of a specific version of the ranking pipeline that has its own preferences. When retraining a model with the newly collected data, the model will be affected by the selection bias and thus has impaired generalization ability. When one model in the ranking pipeline is retrained, the downstream models that consume its output may also be affected by distribution shift [17], since the Tweet candidates produced by the retrained model can be significantly different than before.

The collected training data is also affected by the presentation order, i.e., position bias. Search results are normally presented to the end users in a vertical list. Users have a natural tendency to engage with results at higher positions. Figure 3 plots the engagement rate against position, where each position is assigned a set of randomly selected Tweets. We can see that Tweets placed at the top are more likely to be engaged. Therefore, the positive labels reflect both the relevance of Tweets and the position they were shown to users.

Propensity-based bias correction. Following existing work [14], we correct the selection and position biases with importance sampling [1, 10]. Let X be the random variable denoting the feature vector and Y whether a Tweet is engaged. The expectation of the true loss is computed by

$$\mathbb{E}_{p(X,Y)}\mathcal{L}(f(X),Y) = \sum_{(x,y)\,\in\,X\times Y} \mathcal{L}(f(x),y)p(x,y)\ .$$

where $p(x,y)$ is the density of the joint distribution of X and Y, $\mathbb{E}_{p(X,Y)}(\cdot)$ denotes expectation for $(X,Y) \sim p(x,y)$, $f \in X \rightarrow Y$ is the ranking model, and $\mathcal{L}$ the loss function. Let S be the random variable indicating whether a Tweet is selected and R the position (or rank) of a Tweet shown to a user. Whether a Tweet is engaged y is determined by both the Tweet feature x and its position r. Before that, whether a Tweet is selected s and if selected its rank position r both depend on the relevance score from the ranking models, which in turn also depends on the Tweet feature x. Therefore, the training data can be considered to be following the conditional density $p(x,y \mid S=1,R=r)$, which means the loss of each training example is off by a factor of $\frac{p(x,y|S=1,R=r)}{p(x,y)}$. By scaling each training instance by the inverse of this ratio, denoted by $\omega(x,y) = \frac{p(x,y)}{p(x,y|S=1,R=r)}$, we obtain the unbiased expectation of the loss, i.e.,

$$\begin{aligned}
&\mathbb{E}_{p(X,Y|S=1,R=r)}\mathcal{L}(f(X),Y)\omega(X,Y)\\
&= \sum_{(x,y)\in X\times Y} \mathcal{L}(f(x),y)\omega(x,y)p(x,y \mid S=1,R=r)\\
&= \sum_{(x,y)\in X\times Y} \mathcal{L}(f(x),y)\frac{p(x,y)}{p(x,y \mid S=1,R=r)}p(x,y \mid S=1,R=r)\\
&= \sum_{(x,y)\in X\times Y} \mathcal{L}(f(x),y)p(x,y)\\
&= \mathbb{E}_{p(X,Y)}\mathcal{L}(f(X),Y)\ .
\end{aligned}$$

The ratio $\omega(x,y)$ is often referred to as the inverse propensity scoring (IPS) [10] weight. In order to estimate the IPS ratio for each instance, we propose to decompose the ratio into

$$\begin{aligned}
\frac{p(x,y)}{p(x,y \mid S=1,R=r)} &= \frac{p(S=1,R=r)}{p(S=1,R=r \mid x,y)\cdot p(x,y)}\cdot p(x,y)\\
&= \frac{p(S=1)\cdot p(R=r \mid S=1)}{p(S=1 \mid x,y)\cdot p(R=r \mid S=1,x,y)}\\
&= \frac{p(S=1)}{p(S=1 \mid x)}\cdot\frac{p(R=r \mid S=1)}{p(R=r \mid x,y,S=1)}
\end{aligned}$$

where the mild assumption $p(S=1 \mid x,y) = p(S=1 \mid x)$ is used in the final step. The first term $\frac{p(S=1)}{p(S=1|x)}$ after the decomposition can be considered correction for the selection bias, while the second term $\frac{p(R=r|S=1)}{p(R=r|x,y,S=1)}$ for the position bias. For both terms, the numerators $p(S=1)$ and $p(R=r \mid S=1)$ are marginal distributions that can be estimated from the training data using maximum likelihood. For the denominators $p(S=1 \mid x)$ and $p(R=r \mid x,y,S=1)$, we propose to estimate them by (training and) using two additional correction models. The first model predicts the probability of an instance being selected based on its features, and the second one predicts the probability of an instance being put at position r from its features and engagement label. The training data for these two models can be collected from the ranking pipeline in the same way as described above (cf. Section 6 for more details). A common problem of the IPS approach is the high variance of the IPS weights. For example, when an instance is very unlikely to be selected, then its IPS weight can become unbounded. Approaches to solving this problem includes the normalized IPS [19], CRM framework [18], etc. Through empirical experiments, we find that simply capping the IPS weight can also give reasonable results. We leave exploring other effective approaches as future work.

5 ONLINE PARAMETER TUNING

Besides the learnable parameters in the models, there are many other system parameters in the ranking pipeline (e.g., thresholds or configuration options for heuristics in the policy layers). These parameters can easily become out-of-sync with the constantly evolving ranking pipeline. To maintain the most effective performance (and hence deliver good user experience), we need to tune these parameters after any non-trivial change in the ranking pipeline. To avoid manually tuning these parameters repetitively, which is laborious and inefficient, we propose to use an online parameter tuning module to automatically find a proper set of parameter values.

Online parameter tuning module. The parameter tuning module is designed to optimize a given metric measured through the online user behavior, e.g., number of clicks or social actions, by continuously updating a predefined set of system parameters. Specifically, we keep the system parameters in a shared hyperparameter store. The ranking pipeline reads from this store and serves search results using the read parameters. A daemon job collects relevant online user behavior, computes the metric of interest, and sends such information to the parameter suggestion component. The parameter suggestion component proposes a new set of parameter values based on the history and new feedback, which is sent to the shared store for the ranking pipeline to read. More formally, we are solving the maximization problem

$$\boldsymbol{\theta}^* = \operatorname*{argmax}_{\boldsymbol{\theta} \in \Theta} g(\boldsymbol{\theta})$$

where $\Theta \in \mathbb{R}^d$ is the parameter space and $g(\cdot)$ computes the metric of interest using online user behavior. Although the functional form of $g(\cdot)$ is unknown, the solution of this maximization problem can be approximated through a trial and error approach since we can evaluate the value of $g(\cdot)$ at specific points in the parameter space. During the trial process, we also want to minimize our cost or *regret*, which is defined as the accumulated difference between the metric value at the optimal point and the points during the trial, i.e., $r_T = \sum_{t=1}^{T} g(\boldsymbol{\theta}^*) - g(\boldsymbol{\theta}_t)$.

Tuning with Bayesian optimization. To make the optimization quickly converge to the optimal and ensure good user experience while exploring, we propose to use Bayesian optimization [15, 16] for suggesting the next set of parameters. Specifically, assume that the unknown function $g : \Theta \rightarrow \mathbb{R}^d$ is sufficiently smooth and drawn from a Gaussian process and has the property that any finite set of T evaluations $\{g(\boldsymbol{\theta}_t) : \boldsymbol{\theta}_t \in \Theta\}_{n=1}^{T}$ induces a multivariate Gaussian distribution. Given the parameters $\boldsymbol{\theta}_t$ to be suggested at step t and the set $\mathcal{D}_{1:t-1}$ of previous evaluated parameters and their feedback, the value of the metric of interest follows the following posterior Gaussian distribution

$$\begin{aligned}\mathbb{P}\left(g(\boldsymbol{\theta}_t) \mid \mathcal{D}_{1:t-1}, \boldsymbol{\theta}_t\right) &= N\left(\mu_{t-1}(\boldsymbol{\theta}_t), \sigma^2_{t-1}(\boldsymbol{\theta}_t) + \sigma^2_{\text{noise}}\right)\\ \text{with } \mu_{t-1}(\boldsymbol{\theta}_t) &= \mathbf{k}^\top \left[\mathbf{K} + \sigma^2_{\text{noise}}\mathbf{I}\right]^{-1} g(\boldsymbol{\theta}_{1:t-1})\\ \text{and } \sigma^2_{t-1}(\boldsymbol{\theta}_t) &= k(\boldsymbol{\theta}_t, \boldsymbol{\theta}_t) - \boldsymbol{\theta}^\top \left[\mathbf{K} + \sigma^2_{\text{noise}}\mathbf{I}\right]^{-1} \mathbf{k}\end{aligned}$$

where $\mathbf{k} \in \mathbb{R}^{t-1}$ is the kernel vector between $\boldsymbol{\theta}_t$ and the history $\boldsymbol{\theta}_{1:t-1}$, $\mathbf{K} \in \mathbb{R}^{(t-1)\times(t-1)}$ is the kernel matrix whose entries $\mathbf{K}_{ij} = k(\boldsymbol{\theta}_i, \boldsymbol{\theta}_j)$ where $k(\cdot,\cdot) : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R}$ is the kernel function, σ_{noise} is a scalar accounting for noises in observation, $g(\boldsymbol{\theta}_{1:t-1})$ represents the previous observations from $\boldsymbol{\theta}_{1:t-1}$, and $\mathbf{I} \in \mathbb{R}^{(t-1)\times(t-1)}$ is the identity matrix. The posterior mean $\mu_t(\boldsymbol{\theta})$ and variance $\sigma_t^2(\boldsymbol{\theta})$ quantifies the expected metric value and uncertainty after considering the $T-1$ evaluations. We decide the next set of parameters by finding the point that maximizes the expected improvement from the best, i.e.,

$$\boldsymbol{\theta}_t = \operatorname*{argmax}_{\boldsymbol{\theta} \in \Theta} \mathbb{E}\left(\max\left\{0, g(\boldsymbol{\theta}_t) - g_{\max} \mid \mathcal{D}_{1:t-1}\right\}\right)$$

where $g_{\max}$ is the best observed metric value among $\boldsymbol{\theta}_{1:t-1}$. This expected improvement is a commonly used utility function and has the following analytical expression [16]

$$\text{EI}(\boldsymbol{\theta}_t) = \begin{cases}(\mu(\boldsymbol{\theta}_t) - g_{\max})\,\Phi(Z) + \sigma(\boldsymbol{\theta}_t)\phi(Z) & \text{if } \sigma(\boldsymbol{\theta}_t) > 0\\ 0 & \text{if } \sigma(\boldsymbol{\theta}_t) = 0\end{cases}$$
$$\text{where } Z = \frac{\mu(\boldsymbol{\theta}_t) - g_{\max}}{\sigma(\boldsymbol{\theta}_t)}$$

and $\Phi(\cdot)$ and $\phi(\cdot)$ are the cumulative distribution and probability density functions of the standard normal distribution, respectively. It can be proved that the expected improvement has good convergence rate and is asymptotic regret-free, i.e., $\lim_{T\rightarrow\infty} \frac{r_T}{T} = 0$ [22]. Computing the expected improvement is challenging when the observed metric values are noisy, because $g_{\max}$ is no longer accurate. To handle this case, we apply a simple heuristic by using the largest Gaussian process mean ($\mu_{\max}$) among $\boldsymbol{\theta}_{1:t-1}$ instead of $g_{\max}$ when computing the expected improvement.

6 EXPERIMENTS

We evaluate the proposed techniques with both offline and online experiments (i.e., A/B testing). For simplicity, the main task we use for evaluation is to predict whether a given Tweet will be engaged, i.e., induce any clicks or social actions. The training data we use typically contain tens of millions of examples with balanced positive and negative classes. We use ROC-AUC as the main offline metric and the business metrics summarized in Table 1 for online A/B testing. For online A/B testing, normally 2% to 5% search traffic is used for each treatment. All results using business metrics are reported as the percentage of improvements comparing to the production configuration used during corresponding experiments. We do not have access to the user id or any other signals that can associate a user's identify with search queries. All related data were permanently deleted after the experiments to protect user's privacy.

6.1 Experiments for Relevance Ranking

In the first set of experiments, we evaluate the overall contributions of relevance ranking models. The contributions are revealed by "holdback" A/B testing, in which features are disabled for a small percentage of traffic so that changes in business metrics causally represent the contribution of disabled features. To examine the importance of relevance ranking, we conduct one holdback experiment by turning off the relevance ranking at the Blender stage alone. Earlybird stage relevance ranking along with all front-end UI/UX and rule-based heuristics are still intact. The results are summarized in Table 2. We can see that the Blender relevance ranking models are responsible for 29.3% of total social actions in search, 28.2% of searches with social actions and 1.2% of the overall Twitter user active minutes (UAM). We also conduct another holdback experiment in which relevance ranking in all stages is disabled. As reported in the second row of Table 2, the contribution of overall relevance ranking is 47.7% of the total number of social actions in search, 47.3% of searches with social actions, and 4.0% of the Twitter UAM. These results demonstrate the significant importance of machine-learning based relevance ranking. Given that we only started building the relevance ranking framework since late 2016 [20], it is encouraging to see that 62% of total social actions, 60% of searches with social

Table 1: Business metrics in A/B testing

Business Metric	Meaning
Total SA	Num. of social actions (sa) in search result page
Total Clicks	Num. of clicks in search result page
SA Searches	Num. of searches with at least one sa
Engaged Searches	Num. of searches with at least one sa or click
UAM	User active minutes

Table 2: Contribution of relevance ranking

	Total SA	SA Searches	UAM
Blender	29.3%	28.2%	1.2%
Overall	47.7%	47.3%	4.0%

actions, and 30% of UAM contributions are achieved by the ranking framework in less than two years (at the time of examination).

One of the main advantages of the proposed relevance ranking framework is that it enables an efficient and principled way of evaluating new features. Due to space limitation, we only demonstrate the assessment and contribution of two feature types: counting features and the text embedding features for content relevance. Feature's contribution to model performance is assessed via feature ablation, in which two models are trained and evaluated on a same dataset, with or without a set of features. The change of evaluation metric (ROC-AUC) then represents the effect of those features. We also deploy these models and compare them with online in A/B testing to ensure that the offline improvements lead to online business metric gains. The results are summarized in Table 3. In offline evaluation, both feature types are able to improve AUC by a sizable amount across all three social action types. These results suggest that the counting features are effective for capturing users' preferences to specific types of information on the search result page, and the text embeddings are useful on capturing semantics of the query and Tweet text and transferring them from head queries to tail queries. These offline gains also lead to business metric gains on social actions in the online A/B testing. These reassuring results indicate that the proposed training data processing methods are able to align offline improvements with online gains. This is not trivial to achieve because of the dataset shift, biases and noises, and simplified modeling assumptions.

6.2 Experiments for Bias Correction

We conduct the bias correction experiments at the Blender ranking stage. For the selection bias model, we train a multilayer perceptron that predicts the probability of selection using the same set of features used by the Blender ranking model. Positive examples are Tweets selected by Blender. For the position bias model, we collect the positions of selected Tweets in the ranked list and train a multi-class classifier that predicts the probability of the shown position using the ranking features and engagement signal as input. We cap the IPS weights by 100 which is the 95^{th} percentile. For the Blender ranking model, we collect a new set of training data. We train three Blender models that incorporate either selection-bias correction (**SBC**), or position-bias correction (**PBC**), or both, using the IPS weights from the selection and position bias models. We also train two baseline Blender models. We do not apply any bias correction for the first baseline model. We apply a naive position bias adjustment in the second baseline model, for which the example weight $w(r)$ is computed by $w(r) = r$ for positive examples and $w(r) = N - r$ for negative ones where N is the total number of Tweets in one search result page (normally $N = 20$). This heuristic is from the examination hypothesis [8] and deals with position bias by increasing the importance of positive (*i.e.* engaged) examples that appear lower in the ranked list, or negative (*i.e.* not engaged) examples that appear higher. Models trained using this heuristic perform significantly better in A/B testing comparing to those trained without any position adjustment. The experiment setup is summarized in Table 4a. We report the results using changes in number of social actions in Table 4b. We can see that the model only correcting position bias performs significantly better than both baselines, showing 3.1% and 0.79% more social actions, respectively. This indicates the usefulness of removing the position bias from the training data. The models correcting only the selection bias or both biases lead to no significant (ns) changes. One possible reason is that the large variance of IPS weights of the selection bias model. Capping the IPS weights by 100 may in turn introduce too much bias into the IPS weights.

Table 3: Contribution of features

	Offline ROC-AUC			Online	
	Like	Retweet	Reply	Total SA	SA Searches
Counting	3.1%	2.1%	3.4%	0.71%	0.50%
Text	1.7%	1.7%	2.1%	0.71%	0.41%

Table 4: Selection and position bias correction

(a) Experiment setup

	SBC	PBC
Baseline 1	✗	✗
Baseline 2	✗	naive
Selection	✓	✗
Position	✗	✓
Both	✓	✓

(b) Change of social actions

	Baseline 1	Baseline 2
Selection	-0.21% (ns)	0.58% (ns)
Position	3.1%	0.79%
Both	-0.01% (ns)	0.78%

6.3 Experiments for Parameter Tuning

We implement the parameter suggestion component with an in-house Bayesian optimization toolkit. There are two types of parameters. One is hyperparameters for model training and the other is parameters used by heuristics. We first experiment with hyperparameter tuning for the Blender model. We jointly tune seven hyperparameters that control the learning rate and strength of regularizations. Since these hyperparameters do not directly affect ranking and hence online metrics, we use AUC as the objective and a fixed validation dataset for tuning. For each round of tuning, we select a set of hyperparameters using the component and train the model from scratch. Figure 4a shows the result. We can see that the initial hyperparameters obtain an AUC of 0.918. After 30 rounds of tuning, the suggested hyperparameters consistently achieve a better performance (except one exploring dip) and the objective value also converges. The final AUC value is 0.933 which represents a 1.6% improvement. This demonstrates that the parameter tuner is able to work as expected when the environment is relatively stable.

We then conduct experiment with the parameters used by heuristics. We jointly tune four parameters related to the pagination of search result page (scroll or refresh). These parameters determine

Table 5: Suggested parameters performance

Total SA	Total Clicks	Refresh	Scroll
0.30 %	0.75%	0.70% (ns)	-1.23%

the retrieval time range from Earlybird. We use as the tuning objective the ratio of induced social actions between suggested parameter values and the default values. The result is reported in Figure 4b. We can see that the objective value does not converge after 30 rounds of tuning, instead it shows periodic fluctuations. This is mainly because the number of social actions have seasonal patterns. We normally need to collect data from a wide time span (e.g., two weeks) to obtain stable statistics for one set of trial parameters. Such seasonal changes introduce significant noise to the parameter tuner. The result shows that the objective value is too noisy to allow effective exploration in the parameter space, even if we use the ratio as objective. The gain from a new set of parameter values may be too small comparing to the intrinsic variance of the objective value, making it difficult for the tuner to consistently move the parameter values in the direction of increasing the objective value.

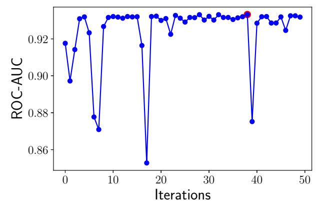
We further investigate the online performance of the suggested heuristic parameters. We experiment with four sets of suggested parameters having the highest objective values since the tuning does not converge. We summarize the results of the best set of parameters in Table 5. We can see that there are more social actions and clicks. However, freshness of the content seems to suffer a slight loss, as there are more refreshes and less scrolls. Although there are still issues such as non-stationary optimal parameter values and low signal-to-noise ratio to resolve, the above evaluation of online parameter turning shows promising results, which demonstrates the effectiveness of the proposed online parameter tuning approach.

7 RELATED WORK

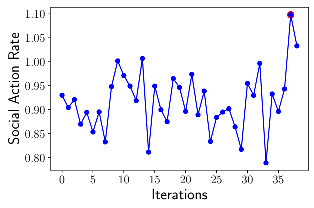
Many studies have discussed their framework for applying relevance ranking in large-scale search engine, which requires the system to be fast, robust, maintainable, and extensible. For example, Yin et al. [24] report the relevance ranking framework in Yahoo search. Covington et al. [7] and Cheng et al. [6] introduce the ranking pipelines that make content recommendations for YouTube and Google Play, respectively. These ranking approaches, including the introduced Tweet relevance ranking, all adopt a similar candidate retrieval and then reranking framework. However, Tweet ranking has its unique properties and challenges from its real-time nature. It is also very expensive for us to replicate these deployed systems and compare with them due to heavy engineering costs.

8 CONCLUSIONS

We have introduced the Tweet search engine at Twitter and the machine-learning based relevance ranking system therein. We have also discussed the ranking models and the features and labels they use in the multi-stage relevance ranking pipeline, which is designed for low-latency, high-throughput, high-availability, and reliable real-time processing. We have also investigated principled approaches for correcting selection and position biases during training data collection. To free engineers from laborious parameter tuning, we have also studied online automatic parameter tuning for the Tweet search system. Experiments using large-scale online A/B testing demonstrate the effectiveness of the relevance ranking pipeline, the approach for bias correction, and the automatic parameter tuning. The introduced Tweet relevance ranking system has been deployed and powering Tweet search at Twitter for more than three years.



(a) Model hyperparameters



(b) System parameters

Figure 4: Performance of parameter tuning module

REFERENCES

[1] Qingyao Ai, Keping Bi, Cheng Luo, Jiafeng Guo, and W Bruce Croft. 2018. Unbiased Learning to Rank with Unbiased Propensity Estimation. *arXiv preprint arXiv:1804.05938* (2018).
[2] Apache. 2019. *Lucene.* Retrieved August 13, 2019 from https://lucene.apache.org
[3] Apache. 2019. *Storm.* Retrieved August 13, 2019 from https://storm.apache.org
[4] Andrei Broder. 2002. A taxonomy of web search. In *ACM Sigir forum*, Vol. 36.
[5] Michael Busch, Krishna Gade, Brian Larson, Patrick Lok, Samuel Luckenbill, and Jimmy Lin. 2012. Earlybird: Real-time search at twitter. In *ICDE.* 1360–1369.
[6] Heng-Tze Cheng, Levent Koc, Jeremiah Harmsen, Tal Shaked, Tushar Chandra, Hrishi Aradhye, Glen Anderson, Greg Corrado, Wei Chai, Mustafa Ispir, et al. 2016. Wide & deep learning for recommender systems. In *Proceedings of the 1st Workshop on Deep Learning for Recommender Systems.* ACM, 7–10.
[7] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *RecSys.* 191–198.
[8] Nick Craswell, Onno Zoeter, Michael Taylor, and Bill Ramsey. 2008. An experimental comparison of click position-bias models. In *WSDM.* 87–94.
[9] Bernard J Jansen, Danielle L Booth, and Amanda Spink. 2008. Determining the informational, navigational, and transactional intent of Web queries. *Information Processing & Management* 44, 3 (2008), 1251–1266.
[10] Thorsten Joachims, Adith Swaminathan, and Tobias Schnabel. 2017. Unbiased learning-to-rank with biased feedback. In *WSDM.* 781–789.
[11] Shichen Liu, Fei Xiao, Wenwu Ou, and Luo Si. 2017. Cascade Ranking for Operational E-commerce Search. In *KDD.* 1557–1565.
[12] Benjamin M Marlin, Richard S Zemel, Sam Roweis, and Malcolm Slaney. 2007. Collaborative filtering and the missing at random assumption. In *UAI.* 267–275.
[13] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *NIPS.* 3111–3119.
[14] Tom Minka and Stephen Robertson. 2008. Selection bias in the LETOR datasets. In *SIGIR Workshop on Learning to Rank for Information Retrieval.* 48–51.
[15] Bobak Shahriari, Kevin Swersky, Ziyu Wang, Ryan P Adams, and Nando De Freitas. 2015. Taking the human out of the loop: A review of Bayesian optimization. *Proc. IEEE* 104, 1 (2015), 148–175.
[16] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. 2012. Practical bayesian optimization of machine learning algorithms. In *NIPS.* 2951–2959.
[17] Masashi Sugiyama and Motoaki Kawanabe. 2012. *Machine learning in non-stationary environments: Introduction to covariate shift adaptation.* MIT press.
[18] Adith Swaminathan and Thorsten Joachims. 2015. Counterfactual risk minimization: Learning from logged bandit feedback. In *ICML.* 814–823.
[19] Adith Swaminathan and Thorsten Joachims. 2015. The self-normalized estimator for counterfactual learning. In *NIPS.* 3231–3239.
[20] Twitter. 2016. *Blog.* Retrieved August 13, 2019 from https://blog.twitter.com/engineering/en_us/topics/insights/2016/moving-top-tweet-search-results-from-reverse-chronological-order-to-relevance-order.html
[21] Twitter. 2016. *Nodes.* Retrieved August 13, 2019 from https://github.com/twitter/nodes
[22] Emmanuel Vazquez and Julien Bect. 2010. Convergence properties of the expected improvement algorithm with fixed mean and covariance functions. *Journal of Statistical Planning and inference* 140, 11 (2010), 3088–3095.
[23] Xuanhui Wang, Nadav Golbandi, Michael Bendersky, Donald Metzler, and Marc Najork. 2018. Position bias estimation for unbiased learning to rank in personal search. In *WSDM.* 610–618.
[24] Dawei Yin, Yuening Hu, Jiliang Tang, Tim Daly, Mianwei Zhou, Hua Ouyang, Jianhui Chen, Changsung Kang, Hongbo Deng, Chikashi Nobata, et al. 2016. Ranking relevance in yahoo search. In *KDD.* 323–332.