Production Ranking Systems: A Review

Murium Iqbal
Overstock.com

Nishan Subedi
Overstock.com

Kamelia Aryafar
Overstock.com

ABSTRACT

The problem of ranking is a multi-billion dollar problem. In this paper we present an overview of several production quality ranking systems. We show that due to conflicting goals of employing the most effective machine learning models and responding to users in real time, ranking systems have evolved into a system of systems, where each subsystem can be viewed as a component layer. We view these layers as being data processing, representation learning, candidate selection and online inference. Each layer employs different algorithms and tools, with every end-to-end ranking system spanning multiple architectures. Our goal is to familiarize the general audience with a working knowledge of ranking at scale, the tools and algorithms employed and the challenges introduced by adopting a layered approach.

CCS CONCEPTS

• **Information systems** → **Learning to rank**; • **Computer systems organization** → *Real-time system architecture*; • **Computing methodologies** → Learning settings.

ACM Reference Format:
Murium Iqbal, Nishan Subedi, and Kamelia Aryafar. 2019. Production Ranking Systems: A Review. In *Proceedings of the SIGIR 2019 Workshop on eCommerce (SIGIR 2019 eCom)*, 10 pages.

1 INTRODUCTION

The domain of ranking has its roots in the field of Information Retrieval (IR). Early automated IR systems, used in the 1950s were first applied to library indexing and employed statistics to retrieve documents from catalogs of thousands [18, 44]. As the Internet has grown, an increasing number of industries rely on web and mobile platforms to reach end users. This has resulted in vastly larger catalogs of both public and private data. Ranking systems have emerged over time to extend the original IR systems to balance the goals of understanding user intent, scoring the relevance of an increasing number of items, and presenting users with results within fractions of a second. Organizations which use the Internet to interface with their users rely on ranking technologies to parse catalogs of millions or billions of items and surface the most relevant ones. These items range from music and movies available on streaming content services, to products for sale on e-commerce platforms, to web pages on the Internet cataloged by search engines, to advertisements for sponsored advertising and more. As such, ranking systems have become a core technology powering sales and user engagement. Users' interactions with the surfaced information is critical to the business of any such organization, and thus even a small improvement to these systems can yield significant growth for the business.

The need to increase user engagement has spurred an iterative experiment driven approach to improving ranking systems. The field has thus evolved into an intersection of research and application, with each system being built to simultaneously leverage complex machine learning (ML) methodologies and adhere to the constraints and tools required to support millions of users in real time. These methodologies select the most relevant items from catalogs of millions and present them in decreasing relevance to users. The need to accommodate experimentation with complex nonlinear models such as those based on deep learning, while still working within constraints, such as low latency, limited compute power and high parallelization, has driven ranking systems to evolve from a single system to a system of systems. These conflicting concerns are separated by isolating functionality within the systems. As such ranking systems are built with several layers of subsystems, including off line models which allow for flexibility of complex experimentation and online models which cater to live system constraints.

We view production ranking systems as having the following component layers:

data processing responsible for aggregating and featurizing raw data from various sources into training data for models
offline representation learning responsible for transforming raw data into embeddings or graph representations
candidate selection responsible for leveraging the learned representations to populate distributed databases with a selection of relevant candidates given a query
ranking model responsible for loading candidates from the distributed database or inverted index and ranking them in decreasing relevance given some context

Ranking systems are deployed to support recommendations, search, and sponsored advertising. In this work we examine production ranking systems as a general framework irrespective of their application. As such, in the context of this paper a query is generally the prompt to which the ranking system responds. This can be a text string used in search, a user or an item in recommendations, or a keyword in sponsored advertising. We use the term item to refer generally to any listing within a catalog, such as products for sale, advertisements, web pages, and more. We examine different approaches used in each of the layers of a ranking system across industries, but a convergence of methodology on any layer is still not apparent. Each individual application of ranking systems technologies requires it's own problem and data specific approaches to be developed. Instead of tabulating all approaches and caveats, which would be outside the scope of this work, we will examine the most popular general methodologies adopted for each layer of a ranking system.

In: J. Degenhardt, S. Kallumadi, U. Porwal, A. Trotman (eds.):
Proceedings of the SIGIR 2019 eCom workshop, July 2019, Paris, France, published at http://ceur-ws.org

The rest of the paper is organized as follows: Section 2 reviews how the conflicting goals of training models to rank items and serving models to support millions of users in real time has given rise to a system of systems. Section 3 reviews ways to aggregate and normalize raw data into training instances for future layers. Section 4 reviews various representations which are built to simplify the task of retrieval. Section 5 reviews how learned representations can be leveraged by online ranking systems to select initial candidates. Section 6 reviews the live models used to infer item to query relevance and how they are served. Section 7 reviews state of the architectures necessary to deploy ranking systems. Section 8 reviews several ways to validate new components within a ranking system, and possible faults that can arise.

2 SEPARATION OF CONCERNS

The separation of concerns across the ranking system layers has allowed organizations to create models which can approximate ideal ranking, learn query intent, perform query re-writing, and diversify retrieved results. All of these applications are addressed by training offline models to capture relevant relationships within their representations. The necessary, often massive computations can be performed in batch, allowing for feature spaces which encode desired relationships within the geometry of the representation space. These relationships can then be directly leveraged by candidate selection or the online model. As the computationally heavy processes are captured in offline representation learning and candidate selection, online models can be built as less computationally complex models, e.g. linear or shallow models, allowing for low latency response time.

This separation allows for highly complex systems to be built and leveraged, but introduces failure points in the form of decreased interpretability of the ranking system behavior, difficulty in tuning, and increased difficulty of validation. This makes it hard to interpret experimental test results and improve upon previous iterations. Architectural decisions within the various layers of a ranking system also introduce corresponding assumptions into the overall system. Often these assumptions although necessary, can obfuscate biases and weaknesses.

Ranking systems have developed two architectures which have facilitated the separation of concerns. These are distributed technologies and one box models. As item catalogs have grown, technologies which require the housing of indices in memory are no longer feasible. As such, distributed databases which can house an index across a cluster of machines and coordinate retrieval from this index have become prevalent [16, 28, 43]. These distributed database technologies are able to house petabytes of data and run computations over their entirety. In parallel to the development of distributed computing technologies, ML models have grown in complexity, especially with the advent of deep learning. These models require specialty hardware to support their high computational complexity, such as CUDA enabled graphics processing units (GPUs) [38]. Often the computations necessary for these models are infeasible to translate to distributed system frameworks, and the volume of model parameter updates required to coordinate across the clusters is prohibitive to deploying these models on distributed systems. As such, one-box architectures are attractive. These architectures pull processed training data from distributed data stores and train complex models in memory on a single machine [13, 23].

An end to end ranking system employs both of these architectures, with distributed architectures providing a sink for raw data, embeddings, candidates and model predictions, and a source of training data and features for one-box models. This system of systems requires orchestration across frameworks, which can be handled via schedulers that are responsible for coordinating workflows for training and deployment of online models [3]. Each set of tools employed by the layers of the ranking model must be selected with care, as increasing the number of employed tools increases the complexity of the overall ranking system. This can cause some layers to be poorly configured, as practitioners are required to master many technologies. In some cases, separation of concerns can lead to isolation of practitioners who specialize on specific layers within the ranking system. This can cause further poor configurations, as practitioners may treat other layers of the system as black boxes, leading to layer specific optimizations which may give rise to suboptimal behavior across the entire system [42].

We feel the separation of concerns is a useful tool, but only when employed with care. Research on productionized ranking systems tends to focus on just a single layer [37, 48, 49, 51]. Practitioners must be careful to examine the individual component layers as well as behavior across the entire system. Only the ability to interpret the system at both scales can yield an understanding of behavior and enable proper iteration to improve results. A diagram of the architectural components necessary to support a full ranking system is provided by in Figure 1

3 DATA PROCESSING

3.1 Datasets

Increasing user interactions is the primary goal of ranking systems [10, 30]. As such, user interaction logs captured by the platform often serves to be the richest source of training data for the system. Some user interactions are ubiquitous, such as user clicks or user item ratings. Others are platform specific, such as purchases on e-commerce platforms, duration of viewing time on streaming content platforms, or likes on social media platforms. Data about the items themselves, such as title, category, associated text and cost, is referred to as side information. Side information is gathered by platforms either by user feedback, such as tagging on social media platforms, or by content providers, such as attribute labels provided by vendors on e-commerce platforms, or by the platform itself such as item categories.

Most data used by ranking systems are sparse high dimensionality vocabularies. This is especially true for user interaction data, where each item in a catalog can be seen as a word in the vocabulary with few user interactions [53]. Side information contains a mix of sparse and dense data, such as sparse multi-hot encodings of text, or dense data of item price and size or item images. Prior to representation learning and model training, raw data must be normalized, featurized and formatted. To reduce noise and reduce computation time, large cardinality spaces can be trimmed via thresholding to drop highly sparse dimensions from the vocabulary. Out of vocabulary components can either be dropped or mapped to a default null

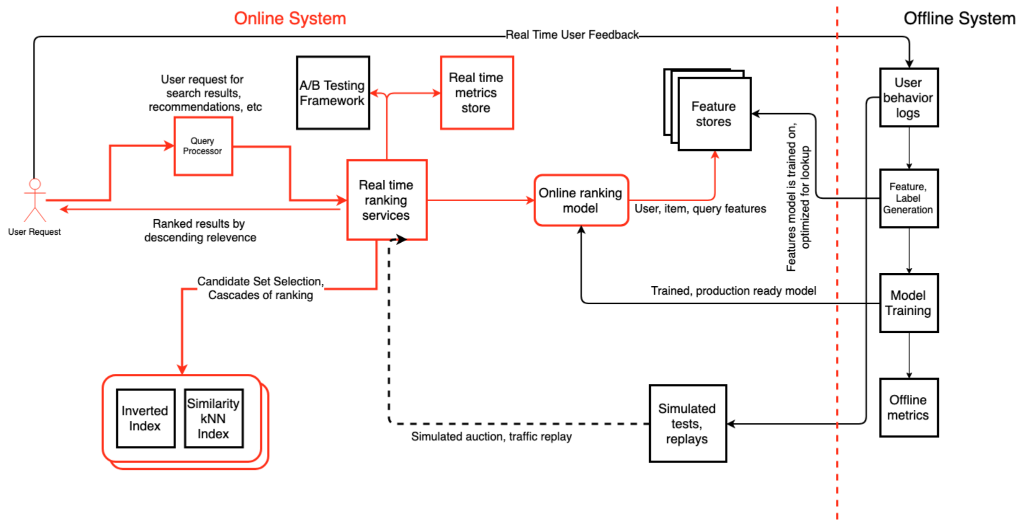
Online System
Offline System
Real Time User Feedback
User request for search results, recommendations, etc
Query Processor
A/B Testing Framework
Real time metrics store
Feature stores
User behavior logs
User Request
Real time ranking services
Online ranking model
User, item, query features
Features model is trained on, optimized for lookup
Feature, Label Generation
Ranked results by descending relevence
Candidate Set Selection, Cascades of ranking
Trained, production ready model
Model Training
Inverted Index
Similarity kNN Index
Simulated tests, replays
Simulated auction, traffic replay
Offline metrics

Figure 1: General system architecture that covers the life cycle for a ranking request. The query can be items, user provided query strings or users in generalized ranking approach, and thus query processor can either act as a query re-writer, user personalization, or an item diversification layer.

representation [49]. Dense features which contain high variability or follow an exponential distribution can be normalized and/or smoothed prior to ingestion by models.

3.2 Data Aggregation And Normalization

User interaction data can be aggregated by various methods, each building its own assumptions into the ranking system. Excepting reviews and ratings, user interaction data is often referred to as implicit feedback data. This is due to the fact that although platforms can present items to users and track specific user interactions, the relationship between those interactions is only implied. Negative signals, indicated by lack of user interaction, can also only be implied as it is impossible to know exactly what items were viewed by the user. Thus, a complete set of ranking labels for all items is impossible to capture [24]. To reduce noise within this dataset, outliers, such as those associated with bot-like behavior or accidental clicks, are removed. This is generally done by thresholding and weighting user interactions by dwell time [21, 48]. Selection of the date range over which to populate data can also affect models, as the volume and sparsity of data changes over the training window. Seasonal trends must also be accounted for when aggregating user interaction data. These aggregations, although necessary, and often specific to dataset and application, propagate assumptions through all layers of the ranking system and should be made with care [42].

look into this further →

Attributing user clicks to searches or recommendation carousels affects both model training and evaluation [15, 27]. This is not limited to sponsored advertising and affects all ranking problems. For example, in gathering training data for search, do products associated with the search only include those clicked immediately after the search? Or should they include items clicked with subsequent searches assuming that these searches are refinements on the original search? Should two items clicked by the same user across days be considered related or only those clicked within the same hour be considered related? Adjusting these data aggregation layers and their underlying assumptions dictates which correlations are and are not captured within the dataset. Class labels on training instances per user click based relevancy can also change depending on the assumptions made [49]. These decisions can be seen as a form of data tuning which build the assumptions into the datasets used for both training, tuning and evaluating ranking systems.

3.3 Creating A Balanced Dataset

The Learning To Rank (LTR) framework poses ranking as a supervised learning problem [30]. As such, both positive and negative samples need to be inferred from user interactions. Just as with positive samples, there are numerous ways to attribute negative samples, for example, if a user exited a video stream before finishing the entire video. There are far more possibilities for negative samples, represented in the lack of user interactions, than positive samples. Various methods for balancing the dataset are employed. One popular method is for each positive sample defined in the dataset only sampling one corresponding negative sample. Different mechanisms of negative item sampling are employed to select negative samples such that specific relationships are reflected in balanced pairs of negative samples and positive samples. This could include choosing negative items from categories that are far away from the item, excluding highly co-viewed or co-engaged products from the negatives list [25]. Selecting negative interactions from only a window around a positive interaction can allow an assumption of user impression.

Ranking systems have an intrinsic positional bias associated with them [40]. Users click on higher presented results irrespective of query relevance, leading to false positive labels. Ignoring this bias and training on naively collected data can lead to models that simply fit the behavior of the existing global ranking function. The *FairPairs* method modifies search results in a non-invasive manner that allows us to collect pairs of results that are unaffected by this presentation bias by randomizing the order of results between a small window of items during presentation. [31].

3.4 Discussion

Although user interaction data is rich, incorporation of side information is necessary in any ranking system. This data is employed to combat popular item bias, and to address the cold-start problem [29]. This side information allows correlations learned from user interactions to be extended to items with few impressions based on relationships which exist in item descriptions, item category information, etc. Although proper processing of user interaction data is required for training high performance ranking models, we find effective ways to leverage side information affords the highest coverage and most diversity. Models which rely too heavily on user interaction data overfit to head queries and popular items.

Processing this data, which is often multiple gigabytes, must be done over a distributed computing platform, as it would be infeasible to fit the dataset in memory on a single machine. These processes are often done in batch, with user interaction data being processed into new training data at regular intervals. As the need for real time personalization and sponsored advertising increases, stream processing methodologies are emerging which allow data to be processed and featurized in near real time. This trend allows ranking systems to become increasingly responsive to user interactions and increase user engagement by modifying representations of users and query intent as users progress through a platform.

4 REPRESENTATION LEARNING

Vector representations, learned over processed data, facilitate communication across the different layers of ranking systems. Representation learning leverages complex state of the art models, often relying on deep learning architectures. These models are highly platform specific, with architectures and solutions which yield large lift in one domain not necessarily providing effective representations on another platform. As such, much recent research on production ranking systems revolves around learned representations and much of a practitioners time is spent on this layer. These models are tuned to transform raw data into succinct expressive representations which can be used either for candidate set retrieval or reranking. These representations can be seen as mappings which project input data into low dimensional embedding spaces where distance is inversely related to relevance. Representation learning is performed offline, which allows this layer to leverage complex nonlinear ML architectures without the constraints of real-time systems. This allows the computational burden inherent in representation learning to be placed ahead of the live ranking layer. These representations are often learned from multiple modalities, incorporating both user interaction data and side information. We examine several methods of learning representations from these modalities both jointly and separately.

4.1 Shallow Embeddings

The simplest architectures to form representations employ shallow architectures such as word2vec over a combination of vectorized interaction data and side information. Here a "word" within the vocabulary can be categorical user actions, a discretized continuous feature, or English words [22]. User action data can be taken strictly as words, or as sequences of actions, in which case embeddings can be built from skip-grams of user action sequences [22, 49]. Multi-hot user action vectors can also be weighted by dwell time [22]. This allows us to learn query and item representations unsupervised from user engagement data. To handle cold start, out of vocabulary items can be taken as linear combinations of embeddings of associated in vocabulary items, or corresponding content data [22, 25, 49].

4.2 Multi Modal Representations

User interaction, text based side information and image based side information are each distinct datasets from various sources that follow distinct distributions. They are, however, related in that they express information about the same items. To simultaneously leverage all datasets and learn a single representation ranking systems employ multimodal learning [36]. The simplest approach to this end is concatenation of raw vectors to generate a single input to a model which learns a representation [37]. Although simple, this method fails to take advantage of the separate structures within each modality. Another approach is to train separate embedding layers for each modality and use these as input to another model which learns a combined embedding. This methodology allows separate architectures to be employed for each modality [48], but adds complexity in that the individual architectures have no obvious method of validation. Furthermore if one modality of data is not present for an item, this could cause unexpected results in the final representation.

Extensions to this method train separate embeddings for each modality but with a cost function used across the separate models to force them to map to the same space [48]. Items can then be taken as a weighted sum of the embeddings of their associated data points. This method allows for items with incomplete side information. Mapping modalities to the same embedding space can also be performed in a the methodology of *search2vec* [22], in which vectors built from side information are first initialized to the corresponding user action vectors. Side information is then sampled to form n-grams which are subsequently embedded into the same space as the user action vectors, and only those within a minimum cosine similarity to the user action vector are kept. Embeddings can be trained jointly over modalities, by employing siamese networks [26], or by allowing certain subgraphs within a network share weights [49]. In these architectures side information is used as features to learn supervised embeddings with the user interactions providing the relevance to encode within the space. This methodology introduces training and architectural complexities but provides powerful representations.

4.3 Multi Task Learning

Multi-task learning aims to train robust representations by jointly training representations for multiple applications [37, 49]. Each task can be viewed as a regularization to the other tasks. This can yield powerful generalizable representations, but requires tasks to be related. Multi-task learning can yield unstable results when poorly configured.

4.4 Graph Representations

User action data can also be represented as graphs instead of vector embeddings. Here nodes are individual items and edges are user interactions. Graphs are initialized with raw user interaction data and side information. The final structure of the graph is learned by training models to prune edges within the graph via logistic regression, gradient boosted decision trees or multi-layer perceptrons [13, 51]. Hierarchical graph structures can also be built, where each level of the graph represents a type of node. For example, in sponsored search setting, the first tier can represent a query signal, the second tier can represent keywords, and the third tier can represent ads. These representations can be used directly with graph based candidate selection methods without the use of embeddings as described in Section 5. Once in graphical form, random walks can be employed to transform graph representations into sample data points. These samples can then be used as input to an embedding model similar to the raw sequences of user clicks. Embeddings built off of these samples purportedly capture higher-order item similarities than direct sampling of user interaction sequences [13, 48].

4.5 Discussion

As state of the art work in deep learning continues to produce more effective representation learning techniques, we find the best approach is to employ embeddings which can surface relationships within the underlying data. Validation of this layer is difficult, and not often discussed in the literature. Instead this layer is often validated in conjunction with the subsequent layers. This can cause improvements within this layer to be hidden by poor tuning in these subsequent layers. As such effective use of representation learning requires development of clear validation of this layer in isolation to the ranking system.

typical case of amazon product catalog.

5 CANDIDATE SELECTION

In ranking systems, candidate selection functions over the learned representations output from the offline models and populates databases which can be read from rapidly by online models. It should be noted that representations used for candidate selection can be distinct and separate from representations used as features to the online model. Representations used by candidate selection support projecting queries into a shared representation space with items that encapsulates similarity, query intent, and support personalization. The goal of candidate selection algorithms is to use the representations to populate a distributed database with a relatively small set of candidates for each query. [22].

After offline systems build embedding spaces in which spatial relationships encode relevance, candidates are selected by nearest neighbor searches. A query is represented in the embedding space, and all items within the catalog closest to it, given some distance metric, are selected as candidates. As directly computing exact $k-NN$ from catalogs of millions is prohibitively computationally expensive, various methods for approximate nearest neighbor (ANN) searches are employed. All of these methods populate an optimized lookup index. Approximate nearest neighbor methods can be broadly broken into three categories, hash based, tree based and graph based.

5.1 Approximate Nearest Neighbors

Hash based. Hashing based approaches, such as locality sensitive hashing or FALCONN [41], are simple models which can be scaled using distributed frameworks as each item can be hashed independent of others within the catalog. These methods compress high dimensional data, via hashes, and assume similar items' hashes will result in collisions [1]. This form of ANN has drawbacks in that in high dimensions false positives and false negatives can appear, as the data is highly sparse and the randomness inherent in the selected hashing function can incorrectly cluster items due to the curse of dimensionality [47].

Tree based. Tree based methods are frequently built as in-memory models with many open source implementations available [4, 34, 35]. Trees are built to be balanced by applying splits along different dimensions of input data. Nearest neighbors search is then performed by traversing the tree starting at the query node and finding nodes within minimum traversals. This method works well on low dimensional data, but at higher dimensions, performance degrades as tree based approaches are complex, often relying on several trees to obtain high performance and traversal of several trees is time consuming [33].

Graph based. Unlike hashing and tree based approaches, graph based approaches function over graph representations of raw data, instead of embeddings. Graphs can be initialized with all incidences of user interaction indicating an edge, which populates a highly dense graph. The graph is then pruned, with models trained to learn relevance [13, 51]. Neighbors are discovered by employing navigable small world (NSW) or walk based proximity algorithms [32].

5.2 WAND

Weighted And (WAND) method is a candidate selection algorithm adopted by some production ranking systems [7]. This method matches queries directly to items using raw features. Items are scored to relevance with queries by a weighted average of all features which they have in common with the query. The top scored items are taken as candidates. This method requires a weight matrix to be learned via constrained feature selection, but provides low latency response with minimal model complexity.

5.3 Serving Candidates

The databases used to serve the results from candidate selection are distributed frameworks to allow for rapid responses. These are updated in batch, with new candidates being populated on regular schedules. Search and advertising systems leverage distributed inverted index technologies such as Solr and Elasticsearch [19, 20]. Other similar architectures leverage distributed key value stores which prioritize high availability, such as Cassandra or Redis [9, 28]. These systems are built to be highly fault tolerant and scalable to support growing numbers of users and items.

5.4 Discussion

Concerns with this layer of ranking and form of separation is propagation of error from new embedding experiments leaking into later layers due to proper lack of tuning of this layer. We find that

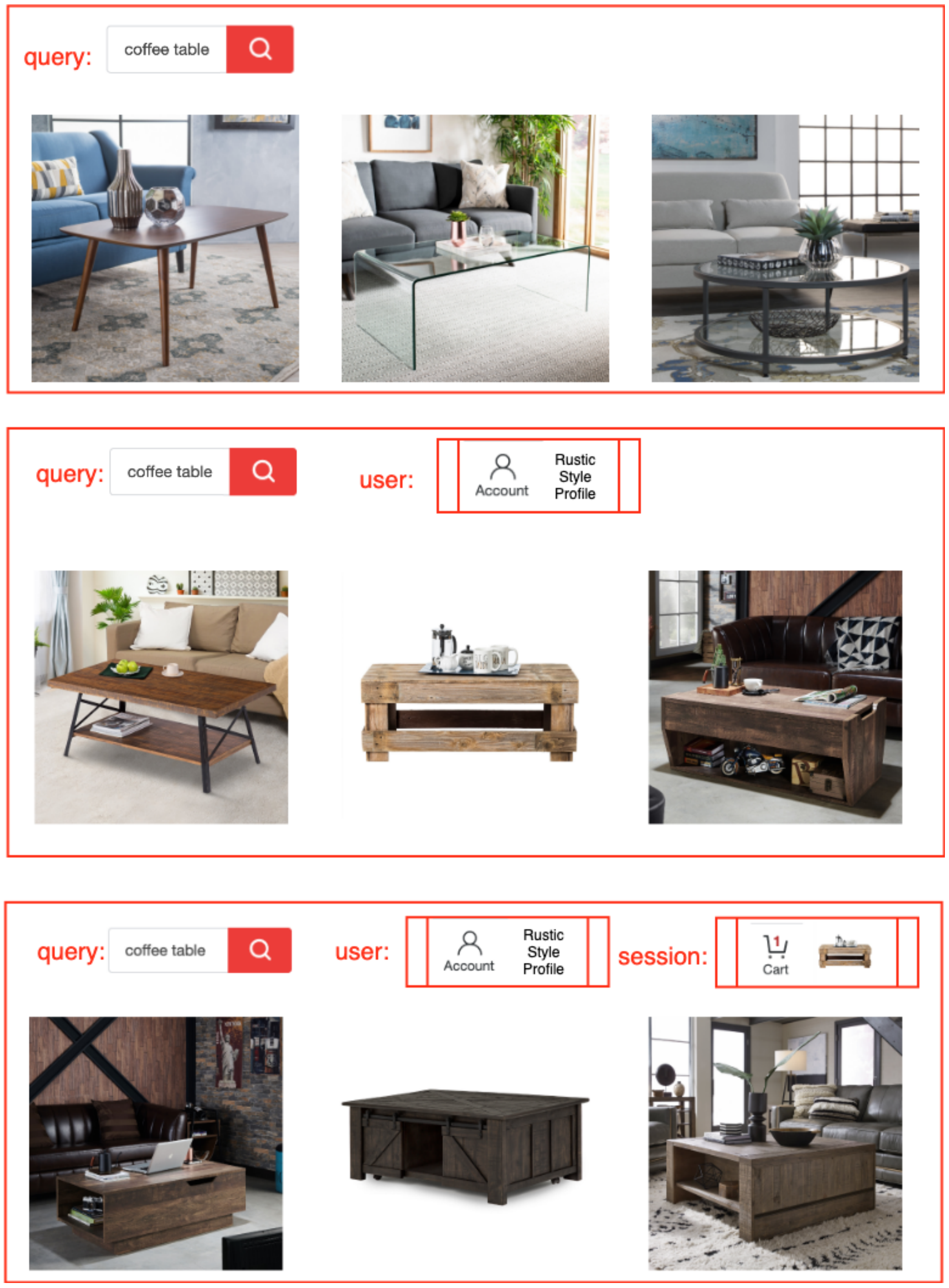

Figure 2: This figure highlights how ranking systems enable real time interaction with users. Users interact with the platform and the system processes their interactions to update their query results. The first row illustrates a global result. The second row shows results for the same query when the user signs in, and provides context based on their past history. The third row presents results for the user for the same query once a product has been added to the cart. This updates the context to the ranking model by providing an additional item, thus narrowing down to the user's information need. At each stage the system is dynamically ranking candidates after interpreting new information about the user to better infer their intent and each items relavance to the intent.

hashing based methodologies are likely the most relevant to ranking, as much work is spent on representation learning. Graph based approaches are weak in that they require massive datasets, and only the largest organizations can afford to employ them. Graph based approaches have the further complication of lacking an obvious approach to addressing the cold start problem, as graph based representations cannot directly represent items which do not already exist within the graph.

6 RANKING MODELS

Online ranking models return candidates in descending order of relevance to a query. Shallow models are used for real time services, making use of offline learned representations as input features [37]. This allows online models to optimize for latency, limited storage and limited compute available to real time services. Figure 2 depicts how the online ranking inference models allow platforms to better interact with and engage users. There are three distinct ways to formulate the ranking problem to train online models; these are pointwise, pairwise and listwise approaches [30].

6.1 Pointwise Approach

$\rightsquigarrow$ RankNet

In pointwise approaches a model is trained to individually score the relevance of each candidate to the query. The problem is formulated as a binary classification, with a positive class indicating a user interaction such as a click or purchase, and the negative class indicating a lack of interaction. Training instances are taken from logs of user interactions. The model provides a score for each candidate reflecting its probability of eliciting an interaction given the query and any context of the user and items captured in the provided features. Candidates can then be ranked in descending order of likelihood for an interaction. This approach affords the use of shallow models, such as logistic regression. This methodology suffers from a lack of context about other candidates, as each candidate is scored individually. This causes an assumption that the output space of the candidates is a multi-variate Bernoulli, where each candidate's score is independent of each other. This is a poor assumption, as users view several items at once, and choose from among them. As such an item's probability of being clicked is affected by its neighbors. This approach is detailed in Algorithm 1.

6.2 Pairwise Approach

$\rightsquigarrow$ LambaRank

In pairwise approaches a binary classifier is trained to score a $\boxed{\text{pair}}$ of candidates simultaneously. The positive class indicates that the first candidate is more likely to be interacted with than the second, and the negative class indicates the opposite. Training data is initialized with all positive classes and balanced by randomly swapping the order of items within a pair with a fifty percent chance. This is the most popular approach as it allows for the model to consider relationships between candidates, rather than scoring them independently. This approach requires an additional sorting to be performed based on the scoring which causes further computation overhead. Pairwise approaches allow for regression models to be used, such as logistic regression and gradient boosted decision trees. Although gradient boosted trees provide better results, the logistic regression still provides the lowest latency response. Popular loss functions used to train these models are binary cross entropy and λ loss function utilized by LambdaRank [8]. This approach is detailed in Algorithm 2.

6.3 Listwise Approach

Listwise approaches require models to be trained over an entire list of items simultaneously. Formulation of a loss function for such a model is difficult, as the true ranking of an entire list is not possible to populate. One method extends pointwise approaches by assuming a multinomial instead of a multivariate bernoulli. This methodology forces candidates to compete with one another for limited probability mass. Other methods try to directly maximize NDCG as their objective [46, 50]. This approach is detailed in Algorithm 3.

6.4 Discussion

The most common approaches for real time models are shallow binary classifications, especially those relying on pairwise approaches, which can rank an entire set of candidates against one another while still posing the problem as a binary classification[37]. These approaches are best suited for machine learning, as binary classifiers are a well studied set of models. We view both pointwise and pairwise approaches as being the most optimal solutions. The challenge with these models is addressed by careful feature engineering in data processing and representation learning layers. Listwise approaches seem ill-posed in comparison, as true ranking data is impossible to obtain with implicit feedback and the models suffer

Algorithm 1 Pointwise Approach (Users, Queries, Clicks)

```
1: Nonclicks ⟵ sample_nonclick(c) ∀ c ∈ Clicks
2: Labels, Interactions ⟵ stack([0, Nonclicks], [1, Clicks])
3: Features = featurize(u, q, i) ∀ tuple(u,q,i) ∈ (Users, Queries, Interactions)
4: P ⟵ sigmoid($W^T$(Features) + $\vec{b}$)
5: C ⟵ Labels × log(P)
```

Algorithm 2 Pairwise Approach (Users, Queries, Clicks)

```
1: Nonclicks ⟵ sample_nonclick(c) ∀ c ∈ Clicks
2: S ⟵ s ~ Bernoulli ∀ c ∈ Clicks
3: Labels, Interactions ⟵ stack(swap_or_not(s, c, n)) ∀ tuple(s, c, n) ∈ (S, Clicks, Nonclicks)
4: Features = featurize(u, q, i) ∀ tuple(u,q,i) ∈ (Users, Queries, Interactions)
5: P ⟵ sigmoid($W^T$(Features) + $\vec{b}$)
6: C ⟵ Labels × log(P)
```

Algorithm 3 Listwise Approach (Users, Queries, Clicks)

```
1: Nonclicks ⟵ sample_nonclick(c) ∀ c ∈ Clicks
2: Labels, Interactions ⟵ stack([0, Nonclicks], [1, Clicks])
3: Features = featurize(u, q, i) ∀ tuple(u,q,i) ∈ (Users, Queries, Interactions)
4: P ⟵ softmax(sigmoid($W^T$(Features) + $\vec{b}$))
5: C ⟵ Labels × log(P)
```

from complexities of trying to learn relevancy of multiple classes at once.

7 SERVING COMPLEX MODELS

Different architectures are used to deploy various types of ranking systems. Each architecture has specific purposes and supports various layers of the overall ranking system.

7.1 Distributed Architecture

Distributed databases employ a cluster of machines and coordinate data storage and computations across the cluster. This architecture can be leveraged for data processing layers of ranking systems to aggregate, normalize, and featurize training data [14, 39]. Recent work has also employed this architecture for representation learning via proprietary distributed computing technologies, such as those used for distributed computation of embeddings [14, 22] or via open source libraries such as those available in Spark [52]. Several forms of candidate selection, those based on hashing, can also be performed on distributed databases. These architectures are used to serve learned representations and results from candidate selection to online models. Distributed databases further provide the ability to simultaneously write billions of records, affording the ability to capture user interaction data from millions or billions of users simultaneously. Distributed databases also afford high scalability, as new machines can be added to a cluster ad hoc to support increases in volume of data. These systems have their own limitations though. One such limitation is described by the CAP theorem which maintains that a distributed data store can not simultaneously provide consistency, availability and partition tolerance [17]. Thus each distributed framework trades off one of these goals for the others, some prioritizing rapid capturing of user interaction with others prioritizing high availability to support real time response. This may require a single ranking system to employ several different distributed datastores, one to capture and process data, another to make data highly available to online models. Distributed data stores also require computations to be written within specific programming paradigms, such as MapReduce, which do not easily represent deep learning computations or graph based computations [23]. Furthermore, model training and prediction on distributed data stores require transference of model weights across entire clusters, which is infeasible for complex deep learning models with many parameters.

7.2 One-box Architecture

One-box architectures allow for the use of complex modeling techniques to be employed in ranking systems. Here a single machine loads training data from a distributed database and trains a model in memory [13, 23]. These architectures are used for representation learning via deep learning models or graph based approaches. Candidate selection can also performed on one-box architectures via tree or graph based methods, which are described further in Section 5. Online models are often trained via one-box solutions, and can be served in parallel to scale to support requests from high volumes of users, such as millions or billions [5]. This architecture affords ML practitioners the freedom to use different tools, without the constraints imposed by distributed architectures. Recent development of containerized solutions allows one-box architectures to simultaneously support a number of tools and solutions by isolating system dependencies [6]. Practitioners are thus free to employ complex models without the need to work within system specific paradigms. One-box architectures are limited to in-memory computations and cannot fully leverage the entire dataset and thus are still reliant on distributed datastores to pre-compute data.

7.3 Serving Deep Learning Models

As research on deep learning has progressed, a push to use these complex models in real time ranking applications has been made. This is a divergence from prior architectures which employ shallow models for real time inference. Several methods have been employed to enable deep learning models to be served and respond in real time. Each has with its own assumptions and trade-offs.

Deep learning training can be scaled by distributing training over multiple GPUs by leveraging recent open source tools, such as Horizon [14]. These tools provide support for training on CPU, GPU and multi-GPU in one box architectures and can provide the ability to conduct training on many GPUs distributed over numerous machines. This framework requires GPU servers, which can be cost prohibitive to purchase and maintain.

Other platforms allow for distributed embeddings by adopting a parameter server (PS) paradigm which employs a cluster of servers. Model updates are communicated across the cluster through a central parameter server. These systems are designed with the following constraints to allow for fast training: Column-wise partitioning of feature vectors among PS shards; No transmission of word vectors across the network; PS shard-side negative sampling and computation of partial vector dot products [11]. Here a shard is a subset of the entire dataset which is acted upon independently by one server in the cluster. These systems are complex to maintain and support as updates are made asynchronously to the model over each shard.

Embedding layers within deep learning models are fully connected, requiring many parameters. These computations are incredibly expensive. To allow for real time embedding of raw signals, model compression via quantization of model parameters is employed [39]. For example, in some layers floating point precision of weights is reduced to 8 and 16 bits. This comes with a reduction in precision, but allows for a smaller memory foot-print. In this approach representation learning must be done such that models and their hardware are co-designed. The complexity of training such a model is much higher. Different types of quantization are performed given the acceptable drop in precision at each layer. Each layer is individually optimized, as well as the entire graph as a whole. Reduction of model parameters can be performed by model architecture decisions as well. For example, in the case of sequential models, GRUs are chosen to learn representations instead of LSTMs as they require fewer parameters [49].

To allow model training to be completed in a timely manner, models with many parameters trained over large datasets can be trained incrementally, with a one-time training occurring infrequently and weight incremental updates being calculated daily [37].

7.4 Discussion

Current design paradigms rely heavily on the coordination of tasks across both distributed and one-box architectures. Data processing and candidate selection are performed distributed, while representation learning and online models are handled via one-box architectures. Tasks are coordinated via schedulers [3] and one box architectures are containerized and deployed in parallel to support requests from many users [5].

We find that recent trends aim to allow service of complex models directly by making complex models more efficient [11, 39, 49, 51]. This methodology has the potential to reduce the layers and separation of concerns within ranking systems. This would be a powerful improvement as it would reduce the complexity of the overall model, allowing for a unified approach. Improvements to streaming data processing technologies could further support these developments as complex computations can be run over distributed data caches in near real time to transform data into input features for deep learning models which are served live. Although promising, this allows for deep learning models to be deployed in real time, but the work lacks a generalized approach, lacks open source support and thus lacks wide spread adoption.

8 VALIDATING A SYSTEM OF SYSTEMS

We have thus far shown that a single ranking system is composed of several layers, each with its own complexity. The need for experimentation to increase user engagement requires both offline validation and online test results. Validating a ranking system, which spans several frameworks is not straightforward. Each individual layer should be validated in isolation in addition to the whole, but often in ranking systems, the reported results are only those of the candidate selection or the final ranking model. Such complex systems thus lend themselves to a change one thing change everything (CACE) data dependency and system entanglement [42]. This makes metrics unreliable, obfuscating errors and making it difficult to prove improvements to the overall system. Despite these reservations, both offline and online testing is performed primarily after candidate selection and after ranking. Depending on application the system and its component layers are generally tested for improvement on click-through-rates, purchases, dwell times and advertisement engagement to name a few. Selection of the desired metric to optimize for must be done with user behavior in mind, but often suffers from biases and is also affected by functionality outside of the ranking system itself, such as user interfaces.

8.1 Increasing experiment bandwidth

Speed of experimentation is hindered by the bandwidth for online testing, as there is a finite amount of traffic that lends itself to each particular test. One approach to improve throughput of testing involves early detection of poor or invalid experiments. This afford greater throughput of experiments as poorly performing tests are detected and terminated early. To allow for this standard metrics are populated frequently and made consistent across all related experiments. A multi-layer experiment architecture can also be employed where experiments are grouped into statistically independent layers. Each user is then simultaneously used as a data point for multiple tests, one from each layer, allowing multiple tests to be run simultaneously [45]. Experiment duration can also be decreased by employing variance reduction techniques, which separate users within the test group into two strata: those with prior purchase behavior and those without. For those with prior purchase behavior, this past data is used as a control covariate for additional variance reduction. This has been shown to reduce the duration of experimentation by half while maintaining equivalent confidence [12].

8.2 Offline evaluation

Even with such methods, bandwidth for online tests is limited. As such new model experiments must prove a significant improvement on offline validation to be selected for a live A/B test. Common offline metrics used are area under the curve (AUC) for the receiver operating characteristic (ROC) and normalized discounted cumulative gain (NDCG) as these correlate well with expected click through rate [22, 37, 49, 51]. Simulated experiments can also be employed to estimate performance of models prior to A/B tests. These simulations must be calibrated to avoid incorporating bias leading to poor estimates of expected click through rate [2].

8.3 Discussion

Focus on ranking metrics for the overall system is necessary, but we propose that each layer requires its own independent metrics as well to avoid obfuscating errors and biases. Data processing layers should document assumptions with metrics dashboards, and gauge distributions of data as well as any underlying shifts within these distributions over time. Validation on learned representations is not documented in most production ranking system architectures, instead they are only measured in their improvement of applications for modeling and candidate selection. Requiring each component to have independent functionality tests as well as tests of the entire system can more clearly surface errors [14, 42].

9 CONCLUSION

We examine production ranking systems and find that a layered approach is adopted in every case. This is necessary to offset the computational cost of leveraging the most effective machine learning models, which are unable to produce real time inference for users. This layered approach causes ranking systems to be composed of a system of systems, each layer employing different algorithms over different architectures. This approach allows for rapid experimentation both within and across layers and allows practitioners to employ state of the art modeling techniques while still adhering to real time service constraints such as low latency and limited available memory. However, this same layered approach causes ranking systems to be incredibly complex, with each layer introducing its own assumptions and requiring its own tuning. This can obfuscate errors and makes it difficult to measure iterative successes. As ranking systems develop new methods to facilitate the direct serving of more complex systems, reliance on this layered approach could be reduced.

REFERENCES

[1] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2019. ANN-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* (2019).

[2] Gang Bai, Zhihui Xie, and Liang Wang. 2018. Practical Constrained Optimization of Auction Mechanisms in E-Commerce Sponsored Search Advertising. *arXiv preprint arXiv:1807.11790* (2018).

[3] Maxime Beauchemin et al. 2016. Airflow.

[4] Erik Bernhardsson et al. 2018. Annoy (Approximate Nearest Neighbors Oh Yeah).

[5] David Bernstein. 2014. Containers and cloud: From lxc to docker to kubernetes. *IEEE Cloud Computing* 1, 3 (2014), 81–84.

[6] Carl Boettiger. 2015. An introduction to Docker for reproducible research. *ACM SIGOPS Operating Systems Review* 49, 1 (2015), 71–79.

[7] Fedor Borisyuk, Krishnaram Kenthapadi, David Stein, and Bo Zhao. 2016. CaSMoS: A framework for learning candidate selection models over structured queries and documents. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 441–450.

[8] Christopher JC Burges. [n. d.]. From ranknet to lambdarank to lambdamart: An overview. ([n. d.]).

[9] Josiah L Carlson. 2013. *Redis in action*. Manning Publications Co.

[10] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM conference on recommender systems*. ACM, 191–198.

[11] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. 2012. Large scale distributed deep networks. In *Advances in neural information processing systems*. 1223–1231.

[12] Alex Deng, Ya Xu, Ron Kohavi, and Toby Walker. 2013. Improving the sensitivity of online controlled experiments by utilizing pre-experiment data. In *Proceedings of the sixth ACM international conference on Web search and data mining*. ACM, 123–132.

[13] Chantat Eksombatchai, Pranav Jindal, Jerry Zitao Liu, Yuchen Liu, Rahul Sharma, Charles Sugnet, Mark Ulrich, and Jure Leskovec. 2018. Pixie: A system for recommending 3+ billion items to 200+ million users in real-time. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 1775–1784.

[14] Jason Gauci, Edoardo Conti, Yitao Liang, Kittipat Virochsiri, Yuchen He, Zachary Kaden, Vivek Narayanan, and Xiaohui Ye. 2018. Horizon: Facebook's Open Source Applied Reinforcement Learning Platform. *arXiv preprint arXiv:1811.00260* (2018).

[15] Sahin Cem Geyik, Abhishek Saxena, and Ali Dasdan. 2014. Multi-touch attribution based budget allocation in online advertising. In *Proceedings of the Eighth International Workshop on Data Mining for Online Advertising*. ACM, 1–9.

[16] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. (2003).

[17] Seth Gilbert and Nancy Lynch. 2002. Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services. *Acm Sigact News* 33, 2 (2002), 51–59.

[18] Cyril W Gleverdon and Cyril W Cleverdon. 1962. Report on the testing and analysis of an investigation into the comparative efficiency of indexing systems. (1962).

[19] Clinton Gormley and Zachary Tong. 2015. *Elasticsearch: The definitive guide: A distributed real-time search and analytics engine.* " O'Reilly Media, Inc.".

[20] Trey Grainger and Timothy Potter. 2014. *Solr in action*. Manning Publications Co.

[21] Mihajlo Grbovic. 2017. Search ranking and personalization at Airbnb. In *Proceedings of the Eleventh ACM Conference on Recommender Systems*. ACM, 339–340.

[22] Mihajlo Grbovic, Nemanja Djuric, Vladan Radosavljevic, Fabrizio Silvestri, Ricardo Baeza-Yates, Andrew Feng, Erik Ordentlich, Lee Yang, and Gavin Owens. 2016. Scalable semantic matching of queries to ads in sponsored search advertising. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*. ACM, 375–384.

[23] Pankaj Gupta, Ashish Goel, Jimmy Lin, Aneesh Sharma, Dong Wang, and Reza Zadeh. 2013. Wtf: The who to follow service at twitter. In *Proceedings of the 22nd international conference on World Wide Web*. ACM, 505–514.

[24] Thorsten Joachims. 2002. Optimizing search engines using clickthrough data. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 133–142.

[25] Bhargav Kanagal and Sandeep Tata. 2018. Recommendations for all : solving thousands of recommendation problems a day. In *Proceedings of the 34th IEEE International Conference on Data Engineering (ICDE)*.

[26] Wang-Cheng Kang, Chen Fang, Zhaowen Wang, and Julian McAuley. 2017. Visually-aware fashion recommendation and design with generative image models. In *2017 IEEE International Conference on Data Mining (ICDM)*. IEEE, 207–216.

[27] PK Kannan, Werner Reinartz, and Peter C Verhoef. 2016. The path to purchase and attribution modeling: Introduction to special section.

[28] Avinash Lakshman and Prashant Malik. 2010. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review* 44, 2 (2010), 35–40.

[29] Blerina Lika, Kostas Kolomvatsos, and Stathes Hadjiefthymiades. 2014. Facing the cold start problem in recommender systems. *Expert Systems with Applications* 41, 4 (2014), 2065–2073.

[30] Tie-Yan Liu et al. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval* 3, 3 (2009), 225–331.

[31] Corey Lynch, Kamelia Aryafar, and Josh Attenberg. 2016. Images don't lie: Transferring deep visual semantic features to large-scale multimodal learning to rank. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 541–548.

[32] Yury A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* (2018).

[33] Marius Muja and David G Lowe. 2014. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine intelligence* 36, 11 (2014), 2227–2240.

[34] Marius Muja and David G Lowe. 2015. Fast library for approximate nearest neighbors.

[35] Bilegsaikhan Naidan and Leonid Boytsov. 2015. Non-metric space library manual. *arXiv preprint arXiv:1508.05470* (2015).

[36] Jiquan Ngiam, Aditya Khosla, Mingyu Kim, Juhan Nam, Honglak Lee, and Andrew Y Ng. 2011. Multimodal deep learning. In *Proceedings of the 28th international conference on machine learning (ICML-11)*. 689–696.

[37] Yabo Ni, Dan Ou, Shichen Liu, Xiang Li, Wenwu Ou, Anxiang Zeng, and Luo Si. 2018. Perceive Your Users in Depth: Learning Universal User Representations from Multiple E-commerce Tasks. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 596–605.

[38] CUDA Nvidia. 2010. Programming guide.

[39] Jongsoo Park, Maxim Naumov, Protonu Basu, Summer Deng, Aravind Kalaiah, Daya Khudia, James Law, Parth Malani, Andrey Malevich, Satish Nadathur, et al. 2018. Deep Learning Inference in Facebook Data Centers: Characterization, Performance Optimizations and Hardware Implications. *arXiv preprint arXiv:1811.09886* (2018).

[40] Filip Radlinski and Thorsten Joachims. 2006. Minimally invasive randomization for collecting unbiased preferences from clickthrough logs. In *Proceedings of the National Conference on Artificial Intelligence*, Vol. 21. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 1406.

[41] Ilya Razenshteyn and Ludwig Schmidt. 2018. FALCONN-FAst Lookups of Cosine and Other Nearest Neighbors.

[42] David Sculley, Gary Holt, Daniel Golovin, Eugene Davydov, Todd Phillips, Dietmar Ebner, Vinay Chaudhary, Michael Young, Jean-Francois Crespo, and Dan Dennison. 2015. Hidden technical debt in machine learning systems. In *Advances in neural information processing systems*. 2503–2511.

[43] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, et al. 2010. The hadoop distributed file system.. In *MSST*, Vol. 10. 1–10.

[44] Amit Singhal et al. 2001. Modern information retrieval: A brief overview. (2001).

[45] Diane Tang, Ashish Agarwal, Deirdre O'Brien, and Mike Meyer. 2010. Overlapping experiment infrastructure: More, better, faster experimentation. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 17–26.

[46] Michael Taylor, John Guiver, Stephen Robertson, and Tom Minka. 2008. Softrank: optimizing non-smooth rank metrics. In *Proceedings of the 2008 International Conference on Web Search and Data Mining*. ACM, 77–86.

[47] Michel Verleysen and Damien François. 2005. The curse of dimensionality in data mining and time series prediction. In *International Work-Conference on Artificial Neural Networks*. Springer, 758–770.

[48] Jizhe Wang, Pipei Huang, Huan Zhao, Zhibo Zhang, Binqiang Zhao, and Dik Lun Lee. 2018. Billion-scale commodity embedding for e-commerce recommendation in alibaba. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 839–848.

[49] Wenjin Wu, Guojun Liu, Hui Ye, Chenshuang Zhang, Tianshu Wu, Daorui Xiao, Wei Lin, Kaipeng Liu, and Xiaoyu Zhu. 2018. EENMF: An End-to-End Neural Matching Framework for E-Commerce Sponsored Search. *arXiv preprint arXiv:1812.01190* (2018).

[50] Jun Xu and Hang Li. 2007. Adarank: a boosting algorithm for information retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 391–398.

[51] Su Yan, Wei Lin, Tianshu Wu, Daorui Xiao, Xu Zheng, Bo Wu, and Kaipeng Liu. 2018. Beyond keywords and relevance: a personalized ad retrieval framework in e-commerce sponsored search. In *Proceedings of the 2018 World Wide Web Conference*. International World Wide Web Conferences Steering Committee, 1919–1928.

[52] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. 2010. Spark: Cluster computing with working sets. *HotCloud* 10, 10-10 (2010), 95.

[53] Meizi Zhou, Zhuoye Ding, Jiliang Tang, and Dawei Yin. 2018. Micro behaviors: A new perspective in e-commerce recommender systems. In *Proceedings of the eleventh ACM international conference on web search and data mining*. ACM, 727–735.