Recommending Similar Items in Large-scale Online Marketplaces

Jayasimha Katukuri*
University of Louisiana, Lafayette
LA, USA
jaykatukuri@gmail.com

Tolga Könik, Rajyashree Mukherjee
eBay Inc.
San Jose, USA
{tkonik,rmukerjee}@ebay.com

Santanu Kolay*
Turn Inc
San Jose, USA.
santanuk2002@yahoo.com

***Abstract*—We are proposing a new similarity based recommendation system for large-scale dynamic marketplaces. Our solution consists of an offline process, which generates long-term cluster definitions grouping short-lived item listings, and an online system, which utilizes these clusters to first focus on important similarity dimensions and next conducts a trade-off between further similarity and other quality factors such as seller trustworthiness. Our system generates these clusters from several hundred millions of item listings using a large Hadoop map-reduce based system. The clusters are learned using user queries as the main information source and therefore biased towards how users conceptually group items. Our system is deployed on several eBay sites in large-scale and has increased user-engagement and business metrics compared to the previous system. We show that utilizing user queries helps capturing similarity better. We also present experiments demonstrating that adapting the ranking function, which controls the trade-off between similarity and quality, to a specific context improves recommendation performance.**

Keywords-Recommender systems, Hadoop; Clustering; Similarity-based recommendations

I. INTRODUCTION

E-commerce sites utilize various recommendation engines to help users find items they may like. *Similar-item recommendations* is a frequently used strategy which gauges user interest in an item (*seed item*) they are engaged with and recommends items that are similar. For example, a shopping site can recommend alternatives for an item a user is viewing as the result of a search, or suggest substitutes for an item that he/she might have lost in an auction.

Similarity-based recommendation is particularly challenging for large dynamic market places like *eBay*, where the number of active item listings are several hundred million and the listings are short-lived, with items often sold within 1-2 weeks. Moreover, most listings contain unstructured data, i.e., data not backed by a product catalogue. In an open market setting, the recommendation systems should also address quality factors like item condition, seller trustworthiness, etc., and negotiate a trade-off between these quality factors and similarity to the seed item. Furthermore, the right balance in this trade-off can depend on the context, i.e., it might differ for different placements at the same site. Finally, how much different item features should contribute to similarity determination is not a function of the inventory alone but depends on how users are interacting with the items listed on the site.

A common technique for recommending similar items is collaborative filtering [1]. In marketplaces with short-lived items, pre-computing recommendations using traditional item-to-item collaborative filtering is not feasible. An alternative approach is to treat the problem in an information retrieval framework where the system uses the seed item features to construct a query and matches it against an index of items in the inventory. This approach can be sufficient if similarity is the sole objective, but when other quality factors need to be weighed against similarity, the resulting queries can get very expensive.

In this paper, we describe a similarity-based recommendation architecture that can handle a dynamic inventory with large number of items, is scalable, captures relative importance of item features based on aggregate user behavior, and provides control over similarity and other quality factors in the mix of recommendations it generates.

Even though the need for trade-off between similarity and other factors can greatly increase computational requirements, our architecture maintains efficiency by separating its flow into a computationally intensive offline process and an efficient runtime component that utilizes the results of the offline process. Although the offline process is computationally expensive, it is highly parallelizable within a map-reduce paradigm and large scale production is feasible.

The offline algorithm uses user queries to capture how users group items that they consider similar in their search activities and creates a dictionary of cluster definitions by refining these queries using common static features of items in the recall set. The resulting dictionary does not depend on specific items and does not need to be refreshed frequently.

The runtime system efficiently combines the static cluster model with dynamic features of items that are active at a given moment. The system first uses clusters to retrieve a small set of items that are similar to the seed item in some important aspects, and then orders items in that set with a ranking function that negotiates similarity with quality. Separation of calculation with static and dynamic factors allows our system to be efficient and scalable enough to

* work done while working at eBay Inc.

cover hundreds of millions of items while serving tens of millions of active users.

In this paper, we take the position that similarity is subjective and should be determined by how users are interacting with the inventory rather than with methods that consider only the distribution of item features in the inventory. Furthermore, we argue that the degree of similarity users want in recommendations is also subjective and the recommendation algorithms can improve performance by optimizing the balance between quality and similarity. We support these claims by comparing our system against a legacy information retrieval (IR) system that was in production at *eBay.com* when we first developed our new system. Our new system captures similarity factors from user queries and negotiates quality against similarity, while the legacy system ignores quality factors and measures similarity based solely on item feature distribution in the inventory.

Our new system is currently deployed at large scale on eBay sites operating in several countries. Compared to its precursor IR system, it achieved significant increase in user engagement and conversion metrics, significantly reduced CPU load on the backend search index, and made a statistically significant impact on site-wise business metrics. We provide evidence that an analogous quality-similarity trade-off can be expensive for a naïve IR system. We show that our system provides customizable parameters to adjust this trade-off so that it works better at different placements with no change on the underlying model or runtime engine.

Next we go over scenarios to exemplify the expected outcome from our recommendation architecture. We follow with a detailed description of the architecture and end with experimental results, related work and concluding remarks.

II. Motivating Scenarios

In this section, we present two use cases in a buying flow at the *eBay.com* site. When an item is listed as an auction item, several interested buyers bid on the item, but the item is sold only to a single buyer. In this scenario, buyers who placed a bid on the item but could not buy will often be interested in a comparable item. Figure 1 shows an example of this particular scenario. The input item (first item from the left in Figure 1) is a running shoe priced at "$79".

Our proposed method first maps the input item to static cluster definitions. In this case, the item is mapped to the following two cluster definitions: *nike air max white gray running* and *nike air max gray black running*. Next, the system retrieves items that match these clusters and recommends top n items after ordering the retrieved items with a ranking function. In this context, users are interested to buy items that are highly similar to the input item, therefore the system uses a ranking function that prioritizes similarity to input item compared to other quality factors such as seller trust worthiness.

Another common scenario occurs when a user visits the page of an item by selecting the item on a search page. In this context, the recommendations should still be similar to

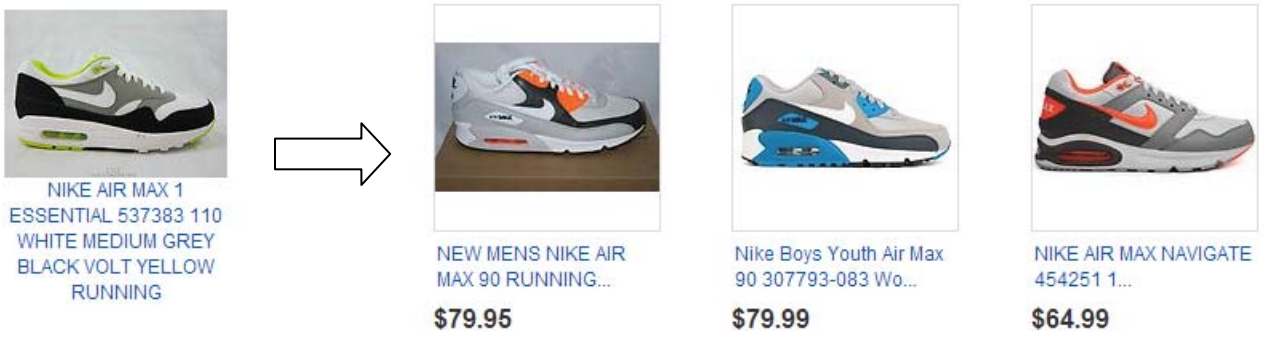

Figure 1. Similar Item Recommendations for very specific user context.

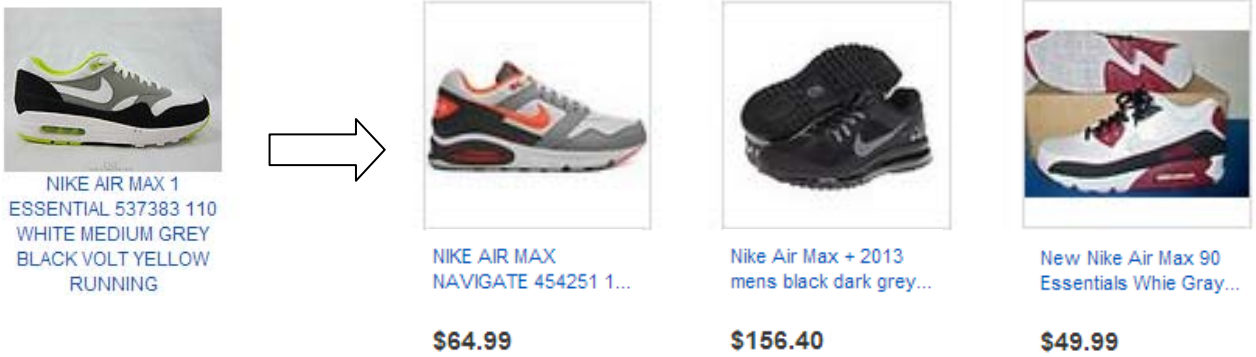

Figure 2. Similar Item Recommendations for broader user context.

the viewed item, but can diverge from the original item in some dimension to provide alternative choices (Figure 2). The candidate items are retrieved using the same clusters in the first use case, which ensures some basic similarity to the seed item, but the ranking function uses higher weight for quality factors such as seller trustworthiness. This allows the system to surface better quality options compared to the first use case by relaxing similarity requirements.

III. ARCHITECTURE

Our recommendation system architecture (Figure 3) consists of a number of components that can be partitioned in three major groups: The *data store* contains data about the active and temporarily changing state of the website as well as models learned using that information; the *performance system* generates recommendations at the site given a session state and information in the data store; and the *offline model generation* creates models by conducting computationally intensive analyses. Broadly speaking, the offline model caches generalized information about a large number of item listings and how people are interacting with them. This information helps to increase efficiency and recommendation quality of the online performance system. However, caching all processing is not possible since some item features that are important for recommendation (e.g. highest bid amount, number of bids on the item, etc.) change in real-time. The online component responds to a large volume of requests, by retrieving data from the cached model and further optimizes its results in real time with dynamic information available about the items.

A. Data Representation

The data store is the glue between the computationally-intensive offline processing and the efficient real-time performance systems. It serves to both offline and online components but the efficiency and availability of access may differ. For example, the offline component has access to the complete history of changes in the item inventory but it does not have efficient search ability on properties of active items. On the other hand, the performance system can query an indexed version of the current inventory, but does not have access to changes over time. Next we describe the data sources that our system uses as input and output.

1) Input Information Sources

The data store contains specific *observed facts* and *generalized models*. Specific facts about the actions on the website are continuously logged. As in any typical e-commerce site, the stored facts can be categorized into *inventory data*, which contains a set of items and their attributes; *clickstream data*, which stores actions and dynamic state of the site. Unlike many ecommerce sites, the inventory at eBay is user generated and it is changing rapidly. A large proportion of items stay active on the site only for less than a few weeks.

The data store also contains generalized models and other information sources. This includes hand-curated knowledge bases like the category hierarchy, which is an ontology organizing the inventory in a hierarchy as well as learned models such as spell correction rules, term frequency models etc. The focus of this paper is learning and using cluster models and therefore we assume other models as given regardless whether they are manually-curated or learned.

2) Output Cluster Model

Our approach assumes a large inventory with short-lived items. In this setting, building models that maintain statistics on individual items is difficult to be of practical use. A model depending on large number of specific items could be difficult to support in an online infrastructure; the model would need to be updated very frequently and more importantly, we would have sparse data per item, since a large portion of items would be involved in little or no user activity.

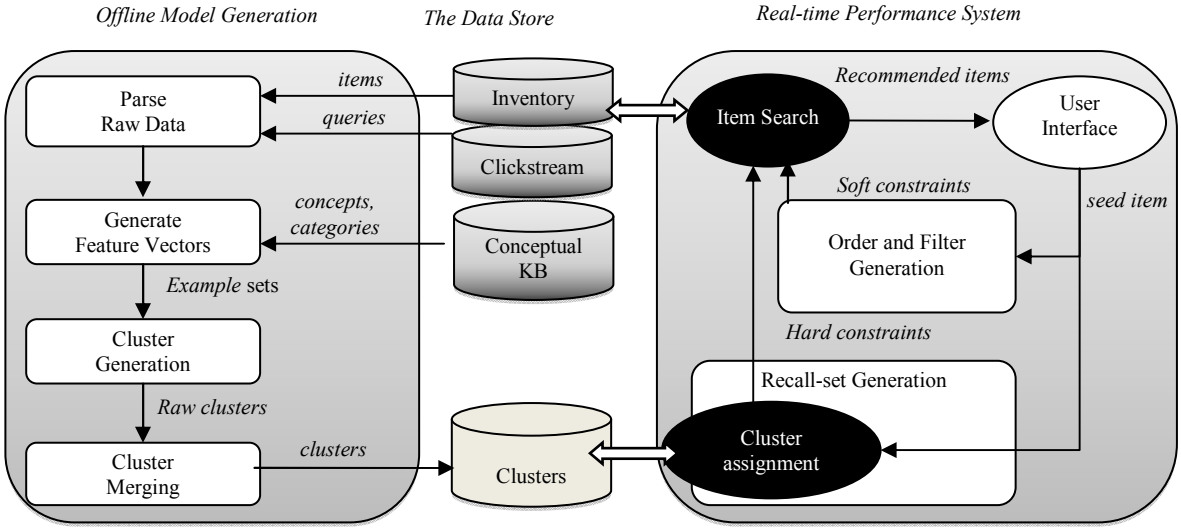

Figure 3. Architecture Overview.

A natural solution is to use generalized knowledge structures that summarize aspects of items but are defined independently of individual items. One of the key elements of our system is a cluster model, which consists of *cluster expressions* that group a set of similar items into a class. One of the most important architectural commitments of our approach is to represent clusters with explicit definitions. For example, Table I shows clusters, where each cluster label is linked to a bag of phrases and Table II shows the items that correspond to these clusters.

TABLE I. CLUSTER EXPRESSIONS AND ITEMS IN THEIR RECALL SET

Clusters
c_1: {nike, air-max, white, gray, running}
c_2: {nike, black, running}

Machine learning systems can use various representation schemes to map a set of items to a cluster label. However, not all of these representations create explicit definitions for clusters. Our system requires explicit definitions to retrieve a set of items given a cluster. More precisely, a predictive cluster model, mapping items to cluster labels, is not sufficient for our purpose and we need a generative model that can efficiently retrieve items given a cluster. In particular, assuming the items in the inventory are indexed using a search engine, given a cluster label, we can use the corresponding cluster expression as a search query and retrieve a set of items that are satisfying the expression. This is a critical feature of the online performance system that we will describe in the next subsection.

TABLE II. CLUSTER EXPRESSIONS AND ITEMS IN THEIR RECALL SET

Item	*Title*	*Cluster(s)*
i_1	Nike white gray 7 mesh Air Max Torch 4 running athletic sneaker shoe NEW $80	c_1
i_2	WOMENS 10.5 NIKE AIR MAX NAVIG. GRAY PINK WHITE BLACK RUNNING SHOES	c_1
i_3	Nike Air Max red black & gray 2010 mens running shoes sz.10- NICE	c_2
i_4	New MENS NIKE AIR MAX SKYLINE RUNNING SHOES black, gray white & neon volt 11	c_1, c_2

Clusters enable our system to track aggregate statistics about the activity of items that belong to a cluster, even though the activity of most these items can be sparse, as it is typical in a dynamic marketplace like eBay. Our system maintains a similarity "importance" weight for each cluster, by matching keywords (e.g. nike) against known entities (e.g. brand=Nike) and tracking the usage of those entities in user activities such as filtering search results. This allows our system to capture subjective importance of each cluster in grouping conceptually similar items. Usage of those statistics is optional in our architecture but the ability to associate statistics with clusters plays a crucial role for a *related item recommendation* system [12], which is also based on the cluster dictionary presented here.

B. Real-time Performance System

The performance system takes a seed item as input, and returns a set of items that are similar to the seed item as output. In adding to the seed item similarity, the process also aims to maximize item quality factors that may depend on dynamic features of the item, like the *remaining-time-to-auction-closing* feature, as well as static or slow changing features like the *seller-trustworthiness* feature. Moreover, the performance system should respond to a high volume of user queries efficiently and work over an inventory with large number of items. Consequently, it aims to strike a balance in a three-way trade-off consisting of similarity to seed item, quality of recommended items, and response efficiency.

Given a seed item, our system generates two separate constraint expressions to query the backend system for items. The first part of this query, *the recall expression*, is applied as hard-constraint. It is satisfied by a small set of items that are similar to the seed item in some important dimension as determined by the cluster dictionary. The second part produces a soft-constraint (ranking function) called the *preference expression*, which is used to determine the order of retrieved items by maximizing a trade-off between similarity to the seed item and item-quality factors. The system next merges these expressions in a search query, sends the query to the data store service that is indexing the inventory, and retrieves n items that best satisfy the query. Next, we describe how our system constructs these expressions.

1) Cluster Based Hard Constraints for Recall

The first part of the online performance system is determination of the recall constraints. These are the hard-constraints that limit the scope of recommendation to a set of items that are similar to the seed item in some important dimensions. The system uses the cluster model to determine what dimensions are important for similarity to a particular item. For a given seed item, the system first retrieves *n* best matching clusters. This is a fuzzy matching process maximizing the intersection of seed item and cluster tokens. It is typical that the seed item contains tokens that do not occur in the retrieved clusters, since clusters are shorter and more abstract than item titles, but we also allow that retrieved clusters to include tokens that are not contained in the seed item. We describe the service that assigns items to clusters in detail in Section D.

Next, the algorithm constructs a query using the matching tokens (single word or phrase) between the seed item and best matching *n* clusters. Let's say we have the matching tokens: $\{t_{11}, t_{12}, \ldots\}$ and $\{t_{21}, t_{22}, \ldots\}$, where t_{ij} is the j^{th} matching token of the i^{th} best matching cluster, we construct the recall constraint expression:

$$(t_{11} \text{ and } t_{12} \text{ and } \ldots) \text{ or } (t_{21} \text{ and } t_{22} \text{ and } \ldots)$$

The disjunction in this expression aims to retrieve items that are similar to the seed item in different dimensions, each represented by a cluster (or rather the portion of a cluster intersecting with the seed item tokens). This expression is used to create a recall set of items that are similar to the seed item in some but not necessarily all important dimensions. The algorithm leaves further refinement on similarity to the next ranking phase, where the selection occurs based on trade-off between similarity and quality.

An alternative approach would be to construct a single conjunction of all matching terms in top clusters. This would yield a query that retrieves items similar to the seed in all-important dimensions our cluster dictionary predicts. However, this query could retrieve few or no items, which leaves little room for quality-based optimization described in the next section. The union query does not have this problem, because the clusters are constructed only when there is sufficient inventory matching the cluster and the union query is guaranteed to contain all items that belong to any cluster used in construction of the query.

2) Soft Constraints for Ordering

The second part of the performance system constructs the *preference soft-constraints* for ordering the items that satisfy the recall constraints described above. The ordering has significant impact on the behavior of the recommendation engine because typically only a small portion of the items in the recall set is presented to the user.

Our system uses a ranking function that takes into account a large number of factors related to item quality. Most of these details are tailored to a particular use case and do not have much theoretical importance. However, there are two points about this procedure we want to highlight. First, this is an expensive computation when applied on a recall set of large number of items. In our system, its use is feasible in practice only because the recall constraints obtained using the offline cluster model restricts the search to a small set of items. Second, the ranking function is a weighted average of two factors of the general form:

$$Score(seed, reco) = w_1 * Sim(seed, reco) + w_2 * Quality(reco)$$

Here, *Sim* is a similarity function between seed item *seed* and recommended item *reco*, and *Quality* function estimates the quality of the candidate recommendations considering multiple dimensions like the item value or seller ratings. Unlike the recall expression in previous subsection, which focuses only on the most important seed item terms, *Sim* function measures the distance between the seed item and the recommended item considering all tokens in the seed item title as well as the dynamic item features.

In summary, our runtime system balances between relevance, quality and efficiency: While the cluster constraints increase efficiency and ensure some lower-bound on relevance, the ranking process negotiates the trade-off between further relevance and the quality of recommended items.

C. Clusters Generation

In this subsection, we describe the offline cluster model generation process. This process first selects a set of frequently used user queries and retrieves a set of items for each of those queries. In the second step, it converts the record of each item into a feature vector to be used in clustering. Next, for each query, it runs a clustering algorithm on the item feature vectors that belong to that query. Finally, it merges clusters learned from different queries by removing duplicates and generating parent-child relations.

This approach has two important features. First, a separate clustering process can run for each user query and therefore the algorithm is highly parallel. Although it is computationally intensive, the overall runtime on a Hadoop system makes large-scale application possible. Second, using user queries is critical to capture how users are grouping items. We treat a frequent user query Q as a signal that the items that satisfy Q is group of items that the users would find similar in some dimension. Each clustering process starts with the items covered by Q and tries to find regularities in the features of those items, further grouping subsets.

1) User Query Selection

Our system selects queries using demand and supply data. It mines the frequency of every query (demand) and the average number of items (supply) that we have on the site for a given query at a time. The algorithm selects queries whose demand and supply meet a minimum threshold. Queries are also normalized to remove duplicates.

2) Item Feature Vector Generation

Item listing titles at eBay typically contain the most important features of the item. We use tokens extracted from the item title as features. The item listings also allow the user to enter a set of attribute value pairs for a given item, like *brand=nike*. Our system first creates a dictionary to detect the attribute-value pairs that frequently occur in each category, and uses this dictionary to mark the attributes in the title tokens. If the multiple title tokens correspond to the same attribute value, the tokens are converted into phrases. For example, given an item title "*NWT New Polo Ralph Lauren CUSTOM FIT MESH Solid Shirt XL*", attribute extractor identifies the attribute-value pairs, *Brand=Polo Ralph Lauren* and S*ize=XL* and the system constructs phrase features like "*polo ralph lauren*" rather than treating 'polo' 'ralph' and 'lauren' as separate features.

Once the features are extracted, the input item is represented as a feature vector with weights, using the *Mutual Information* shared by the feature and the category in which the item is listed:

$$MI(F, C) = \sum_{\substack{c \in \{0,1\} \\ f \in \{0,1\}}} p(F=f, C=c).\, log\left[\frac{p(F=f, C=c)}{p(F=f).\, p(C=c)}\right]$$

Here F is a feature and C is a category in the category hierarchy. The values 0 and 1 correspond to the absence and presence of the random variables respectively. The above

formula provides higher scores for features that occur frequently in a given category and occur few times in other categories.

The feature vectors form the basis of the clustering algorithm and provide a way to compute the intra and inter-cluster distance or similarity as described below.

3) Parallel Clustering of Queries

Our system considers the selected user queries as base clusters from which more granular clusters are created. The feature vectors of the items that belong to the base clusters are used to generate more granular clusters. Our implementation runs the clustering algorithm on a Hadoop map-reduce stack, where cluster generation for each of the queries is done in parallel and independent of each other.

We use the bisecting K-means clustering algorithm [13], which starts with each base cluster and at every step bisects existing clusters to create sub-clusters that maximize *intra-cluster similarity I*, while minimizing the *inter-cluster distortion E*.

$$maximize\left(\frac{I}{E}\right) \text{ such that:}$$

$$I = \sum_{i=1}^{k} \sqrt{\sum_{v,u \in C_i} cosine(v,u)}$$

$$E = \sum_{i=1}^{k} n_i \frac{\sum_{v \in C_i, u \in C} cosine(v,u)}{\sqrt{\sum_{v,u \in C_i} cosine(v,u)}}$$

In these equations, C_i is the set of items assigned to the i^{th} cluster, C is the set of all items that are to be clustered and n_i is the number of items in the i^{th} cluster. One of the difficulties in using *k-means* clustering is choosing the value of the parameter *k*. In our method, we have instead selected the size of the cluster (number of items within the cluster) as a parameter and *k* is computed based on the cluster size parameter.

4) Merging Clusters

The granular clusters are generated in parallel from different queries and therefore, we may get duplicate or close clusters from different queries. The item inventory within a cluster is used to compute relationships between two given clusters. We observe the following relationships between any two clusters generated from different queries:

Non-overlapping clusters: The two clusters represent different inventory groups within the inventory. The clusters do not share any items or have an insignificant overlap. In this case, we do not need to take any action.

Overlapping clusters: The two clusters have some overlap, but they are not duplicates of each other. In this scenario, we keep both clusters, if they capture significant inventory that is not overlapping with any other clusters.

Duplicates or near duplicates: The two clusters cover the same item inventory or nearly the same item inventory, we remove the smaller cluster.

Parent-child: One cluster is contained in the second cluster. In this scenario, we keep both clusters and mark the parent-child relationship.

D. Cluster Assignment Service

We have a service that assigns one or more clusters to an input item. To this effect, we index the cluster definitions in a Lucene[2] index that retrieves clusters given a seed item's features.

We utilize the built-in Lucene scoring function[3] for ranking matching clusters. Our general goal for finding the best matching clusters is to "reward matching features and penalize non-matching features of a cluster". For an input item *i* represented as a set of tokens *features f*, the score for a cluster *c* is described by:

$$score(i,c) = C(i,c) \sum_{f \in i} idf(f)^2 \cdot B(f) \cdot N(f,c)$$

where $C(i,c)$ is a factor that rewards matching features by measuring the number of features matching from the input item. The *idf(f)* factor captures the importance or rarity of the feature *f* in the corpus of clusters. $B(f)$ is a boosting factor capturing relative importance of the features as pre-computed from user behavioral data (queries, usage of filters on the search page). For example, when users search for golf clubs, they frequently use the filter attribute 'dexterity (left handed or right handed)' based on which the algorithm weights the 'dexterity' feature higher than other features that have less user engagement in golf clubs. Each cluster *c* is also penalized for its non-matching features through the index time boosting factor $N(f, c)$, which normalizes the cluster by the number of features used to define it. The scoring function above assigns items to clusters, by maximizing the number of matching features and the importance of the matched features.

IV. Experimental Results

We compared our similar-item recommendation system SIR against the naïve information retrieval system (NIR) that was handling similarity-based recommendations at eBay before SIR was deployed in production. NIR uses the seed item title as a search query to recommend the best matching items in the inventory. Like most retrieval systems, this system gives stronger weight to keywords that are less frequent in the index, however, unlike SIR, it does not use information about the set of keywords users use to group similar items. NIR also does not consider the trade-off between quality and similarity; instead, it optimizes recommendations only for similarity to the seed item.

We hypothesized that these two changes, namely, *increasing quality by sacrificing similarity* and *utilizing similarity factors learned from user queries* should improve recommendation quality. We tested this hypothesis with a

[2] http://lucene.apache.org/core/4_10_1/index.html

[3] http://lucene.apache.org/core/4_4_0/core/org/apache/lucene/search/similarities/TFIDFSimilarity.html

large-scale A/B test at eBay.com site. Our first result shows that SIR outperforms NIR by a statistically significant margin and puts significantly less load on the search-backend system that is indexing the inventory.

Next, we investigate the individual contribution of both of these effects via a lesion study. In the second subsection, we look at the effect of removing the clusters or reducing their number while keeping a fixed ranking function. In the final subsection, we investigate effect of changing the ranking function when the cluster model is fixed. We conclude that both clusters generated from user queries and the similarity-quality trade-off ranking are instrumental in success of SIR.

A. Comparison with Pure Retrieval

We conducted an A/B test on the Closed View Item Page (CVIP) at *eBay.com* site to compare SIR against NIR. Users typically come to the CVIP page upon losing an auction. The goal of this page is to keep users engaged with the site and provide options to buy a closely similar item in place of the item they could not win.

Even though NIR is a less sophisticated system, there are several reasons why it could potentially outperform SIR. Unlike SIR, which focuses on a small set of items that match the best clusters for the seed item, NIR considers all items in the inventory as candidates to find the most similar items. This means that if SIR clusters were not good in reflecting important similarity factors, or if their weights do not reflect user preference for important features of items, the algorithm could get stuck with a poorly chosen recall set and may do poorly against NIR. SIR could also fail, if we use a bad quality measure or if the ranking function does a poor job capturing average user preference for the balance between quality and similarity.

In our A/B test, SIR algorithm showed statistically significant improvement over NIR in both engagement and purchase metrics, although we are not allowed to publish purchase metrics (Table III). We saw significantly higher user engagement, 38.18% increase in users clicking on recommendations and 89.6% increase in people saving the recommendations in their "watch-list" for future viewing. The new system also increased the number of items converted into a purchase. Moreover, the new algorithm made a statistically significant positive impact on eBay's overall revenue, although, we are not allowed to disclose the numerical impact on financial metrics. These results support our proposition that measuring similarity with clusters generated from user demand and striking a balance between item similarity and quality can improve recommendation quality.

TABLE III. A/B TEST RESULTS COMPARING SIR AND NIR.

	ΔCTR	***ΔWTR***	***ΔPL***
(SIR – NIR)/ NIR	+38.18%	+89.6%	-81%

CTR: clicktrough rate, WTR: watch through rate, PL: processing load.

We also measured the processing load the system puts on the inventory backend system. The results show a dramatic reduction in processing load; SIR is **five times** more efficient in item retrieval compared to NIR, even though, SIR uses a more complicated ranking function, because SIR is first retrieves a small set of items that match the clusters.

B. Contribution of Clusters

Next, we investigated the contribution of clusters to SIR via a lesion study. We compared SIR against two systems with zero (SIR-Z) or reduced number of clusters (SIR-R), where all of these systems use the same ranking function.

First, we tested SIR-Z. Our backend system does not handle ranking queries without any hard-constraints, so we chose a generic retrieval constraint, which should contain most items similar to the seed to some degree. In particular, SIR-Z recall set contains all items that match the general category of the seed item and at least one nonstop-word keyword from the seed item title. The resulting system performed extremely inefficiently and put **57 times** more CPU load on the backend system compared to SIR. This result supports the utility of clusters in reducing computational complexity. Unfortunately, the bad performance of SIR-Z made an A/B test impossible and we do not know how SIR-Z would fare against SIR in terms of recommendation quality.

Next, we evaluated SIR-R, a variant of SIR model, which uses the same ranking function but contains 15 % fewer user queries compared to SIR. We hypothesized that recommendation quality would be sensitive to changes of the number of clusters and could reduce with fewer clusters. To test this hypothesis we compared SIR-R and SIR at eBay.com site using an A/B test.

We observed that reduction in number of user queries has indeed reduced performance (Table IV) both in click-through-rate and watch-through-rate. We also computed the average cosine-similarity between input item title and recommended item titles for both systems. The average similarity score decreased by 5% suggesting that the performance degradation may be correlated with decrease in relevance.

TABLE IV. A/B TEST RESULTS COMPARING SIR TO SIR-R

	ΔCTR	***ΔWTR***	***ΔSimilarity***
(SIRR – SIR)/SIR	- 4.2 %	- 4.8 %	- 5%

C. Contribution of Ranking Function

Earlier in the paper, we claimed that our system can negotiate the trade-off between the similarity and quality, and this trade-off can have different optimal points for different contexts. To substantiate this claim, we compare two versions of SIR. Both versions use the same cluster model and differ only in the parameter that adjusts the trade-off between quality and similarity. The first variant SIR-Q has more bias towards quality while the second variant SIR-S has more bias towards similarity.

We compared SIR-Q and SIR-S with a user A/B test on two different placements at eBay site with qualitative different properties: *Closed View Item Page (CVIP)* shows an item with completed auction. Users often come here after

losing an auction on that item and they would be interested in items that are strongly similar. On the other hand, people usually visit *Active View Item Page (AVIP)* while browsing the inventory and might be more likely to consider slightly different alternatives that have higher quality. Based on domain expertise and intuition, we are assuming that similarity in recommendations is more important at CVIP compared to AVIP.

We hypothesize that the performance of the SIR algorithm is sensitive to changes in the ranking function and should differ significantly for different settings of the trade-off parameter. Furthermore, increasing weight on similarity should improve performance in placements that users expect similarity (CVIP) and reduce performance in placements where users are more flexible in terms of similarity and value quality more (AVIP). Table V compares relative click-through-rates of SIR-Q and SIR-S on AVIP and CVIP pages. SIR-S outperformed SIR-Q (+**3.13%** CTR margin) while SIR-Q outperformed SIQ-S on AVIP page (+**4.5%** CTR margin). We observed the same trend in purchase metrics but we are not allowed to report those numbers. Indeed, we find a correspondence between the system that is tuned for more similarity and the page that is expected to require more similarity. These results support our hypothesis that the performance of SIR can be optimized by changing a single parameter to tune the trade-off between similarity and quality. Furthermore, the optimum values for that balance may lie at different points for different placements.

TABLE V. SIR-Q VS SIR-S ON CVIP AND AVIP PAGES

	Closed Item Page	***Active Item Page***
Best system	SIR-S	SIR-Q
ΔCTR	3.13%	4.5%

V. RELATED WORK

Large scale recommendation engines like Amazon's product recommendations [1], YouTube video recommender system [2], and Google news personalization service [3] are popular and used routinely by a large volume of users. Recommendation engines can be broadly categorized into 'content based' and 'collaborative filtering based' systems. Content based algorithms use item features to compute similarity with respect to other items and recommendations are based on similarity. Collaborative filtering methods compute item-item matrix using the user behavioral data such as co-purchase [1] or co-views [2]. Collaborative filtering methods come in two major flavors: user-based and item-based. Item-based methods are shown to be more scalable [4]. Deshpande and Karypis [5] provide a detailed comparison of user-based and item-based recommendation methods.

Most of the existing recommender systems address recommendations in a stable collection of items or products. Amazon's recommender system [1] works in the space of products that are stable and do not expire in a short period time. Netflix recommends movies [6] from a slowly growing collection. Therefore, both of these systems can pre-compute item-item relationships using collaborative filtering methods. The Google news personalization [3] is one of the few works that addresses the issue of recommendations when there is item-churn. In addition to item churn, in dynamic e-commerce settings like eBay, recommender systems need to address 'seller quality and item quality' issues. Our proposed system provides control over the trade-off between relevance and quality. Rodriguez, Posse and Zhang describe one of the few recommendation systems that negotiate a trade-off between multiple factors [7]. In this paper, the authors propose a method for matching candidates for job postings. Their proposed method aims to maximize competing objectives such as similarity of the candidate with respect to job posting and the job seeking intent of the candidate. One major difference of our settings is that the item listings have a short duration. Aggarwal, Zhao, and Yu [8] propose a model for clustering text documents, which combines textual features with other information like as user actions on the documents. Their approach is analogous to ours but it is applied to a different problem. Aggarwal and Zhai [9] provided a comprehensive survey of text clustering algorithms. Wang et al. [10] proposed the use of related categories to mine related products for post-purchase recommendations in e-commerce. Their proposed method does not address the scenario with short-lived items. Chen & Canny [11] describe a method for recommending related items when items are short-lived. Their method first maps the items to a stable representation and then uses transactional data to compute relationships between the groups. However, their method differs from ours in that they do not use user queries to generate clusters.

VI. CONCLUSION

We presented a similarity-based recommendation architecture that can handle large volumes of dynamic inventories, is scalable, captures relative importance of item features based on aggregate user queries and other user behavior metrics, and provides control over similarity and other quality factors in the mix of recommendations it generates.

To maintain efficiency, the architecture is divided into a computationally intensive, yet highly parallel offline process and an efficient runtime component. The offline process is implemented on a Hadoop map-reduce framework. This process generates clusters by specializing frequent user queries with common features of items that satisfy those queries. This approach, to a certain extent, captures similarity dimensions that the users consider important. We further capture relative importance of features (and consequently the clusters that contain them), by tracking frequency of usage of those features when users filter or refine their search queries.

The runtime component efficiently retrieves the best matching clusters given a seed item and uses those clusters to construct a small recall set, which consists of items that are similar to the seed item in some important dimension. Next, it ranks the items in this set using a complex function that

negotiates similarity with quality. The algorithm provides a parameter to optimize this trade-off for a particular placement of the recommendation system. The real-time system can store the cluster model learned offline in memory and can easily respond to millions of queries daily.

Our system is deployed at *eBay.com* in large-scale and we showed statistically significant improvement in user engagement metrics as well as site-wide business metrics compared to the legacy system that was previously in production at *eBay.com*. Moreover, the new system has significantly lower CPU load on the search backend although it is making a substantially more complicated ranking computation.

We claimed that the main differences, 1) capturing user preferences for similarity using clusters learned from user queries and 2) striking a balance between quality and similarity, are jointly responsible for the success of our system. We have further investigated the partial contribution of both of these qualitative differences using a lesion study and conclude that the success of our system is sensitive to changes in both. We showed that removing clusters completely would be dramatically costly due to the complicated ranking function and reducing the number of queries in the input (and therefore reducing the number of clusters) degrade the recommendation quality of our system in terms of user engagement and purchase metrics. We have also showed that the system is sensitive to changes to the parameter that controls the quality-similarity trade-off in the ranking function and the recommendation quality improves by tuning this parameter for different placements.

Future work includes using structured representations in clusters in the form of attribute-value pairs to more accurately represent differences, using natural language parsing to better retrieve important group of words and personalize recommendations for individual users based on their past activities.

Our approach proved statistically significant benefit to the business of a multi-billion company. A large production level system such as ours have many details that might not be theoretically relevant, therefore, we highlighted ideas that seem to be reusable in other circumstances and supported those ideas with experimental evaluation.

ACKNOWLEDGMENT

We thank Abhinaya Sinha, Riyaaz Shaik, Kranthi Chalasani, Aravind Ragipindi, Murali Padavala, Venkat Sundaranatha and Merchandizing Team at eBay for contributions to this research with implementation and ideas.

REFERENCES

[1] Linden, G., B. Smith, and J. York, *Amazon.com Recommendations: Item-to-Item Collaborative Filtering*. IEEE Internet Computing, 2003. **7**(1): p. 76-80.

[2] Davidson, J., et al., The YouTube video recommendation system, in Proceedings of the fourth ACM conference on Recommender systems. 2010, ACM: Barcelona, Spain. p. 293-296.

[3] Das, A.S., et al., Google news personalization: scalable online collaborative filtering, in Proceedings of the 16th international conference on World Wide Web. 2007, ACM: Banff, Alberta, Canada. p. 271-280.

[4] Adomavicius, G. and A. Tuzhilin, Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. IEEE Trans. on Knowl. and Data Eng., 2005. **17**(6): p. 734-749.

[5] Deshpande, M. and G. Karypis, *Item-based top-N recommendation algorithms.* ACM Trans. Inf. Syst., 2004. **22**(1): p. 143-177.

[6] Koren, Y., Factorization meets the neighborhood: a multifaceted collaborative filtering model, in Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining. 2008, ACM: Las Vegas, Nevada, USA. p. 426-434.

[7] Rodriguez, M., C. Posse, and E. Zhang, Multiple objective optimization in recommender systems, in Proceedings of the sixth ACM conference on Recommender systems. 2012, ACM: Dublin, Ireland. p. 11-18.

[8] Aggarwal, C.C., Y. Zhao, and P.S. Yu, On Text Clustering with Side Information, in Proceedings of the 2012 IEEE 28th International Conference on Data Engineering. 2012, IEEE Computer Society. p. 894-904.

[9] Aggarwal, C. and C. Zhai, *A survey of text clustring algorithms*, in *Mining Text Data*. 2012, Springer. p. 77-128.

[10] Wang, J., B. Sarwar, and N. Sundaresan, Utilizing related products for post-purchase recommendation in e-commerce, in Proceedings of the fifth ACM conference on Recommender systems. 2011, ACM: Chicago, Illinois, USA. p. 329-332.

[11] Chen, Y. and J.F. Canny, Recommending ephemeral items at web scale, in Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval. 2011, ACM: Beijing, China. p. 1013-1022.

[12] Katukuri, J., Mukherjee, R., and Konik, T, Large-scale recommendations in a dynamic marketplace. LSRS-2013, ACM Conference on Recommender Systems. 2013. Hong Kong

[13] Steinbach, M., G. Karypis and V. Kumar, A comparison of Document Clustering Techniques, In KDD Workshop on Text Mining, 2000, Boston, USA.