Interactive Search with Contextual Navigation Recommendations

Tolga Könik
eBay Inc.
San Jose, CA 95125
tkonik@ebay.com

Yoni Medoff
eBay Inc.
San Jose, CA 95125
ymedoff@ebay.com

Rajyashree Mukherjee
eBay Inc.
San Jose, CA 95125
rmukherjee @ebay.com

Abstract

We present an interactive search assistance agent that integrates search and recommendation-based experiences into a novel unified presentation. The agent guides the user toward more desirable inventory, while also satisfying the constraints that the user constructs with the help of the agent. We built a prototype which supports this mechanism by connecting to the search functionality on *eBay*. We have also created the popularity knowledge-base of the agent by aggregating data from billions of user transactions. The accompanying video[1] demonstrates the resulting prototype in this context.

1 Introduction

Many popular applications that provide services like shopping, search, media, and social networking, rely on information retrieval (IR) technologies that can be broadly categorized into two groups: Query-based systems that primarily aim to satisfy user input (i.e. keyword search, filtering, and browsing) [1] and Recommender systems [2] that take an active role in recommending content without the need for direct user input. The lines between these two IR systems can blur; a keyword search system might customize its results based on recent user behavior and a recommendation system could narrow its results to be consistent with recent search queries. However, these components typically function independently, and user interfaces emphasize certain functionality based on the expected utility for the user.

Here, we present *Focus*, and interactive assistant for a shopping website, which blurs the lines between user query-based search and system-initiated recommendations. Our proposal involves an intelligent system that unifies keyword search, category browsing, attribute filtering, query completion, related query recommendation, and popularity-based recommendations.

Focus behaves similar to a search engine in that it returns results satisfying user queries and selected filters, while simultaneously acting like a recommendation engine in introducing popular inventory. It can assist users by suggesting refinements for narrowing a query and proposing changes to particular parts of a query in order to move the search towards alternative directions. The latter functionality allows the user to discover popular parts of inventory that contain products with high seller supply and buyer demand. This also helps the user avoid (or backtrack from) low result scenarios.

Our design observes several principles:

- *Interactive*: Shopping/search should be a conversational interaction between the user and shopping assistant.
- *Mixed-initiative* [3]: System and user take the initiative in turns and the UI provides a mechanism for the two agents to share information and align their goals.
- *Incremental*: The UI enables the user to make incremental changes (refinement, pivoting, and generalization) to the query, and receive immediate feedback on how those changes affect the results. This helps the user build a mental model of the inventory while navigating through each part of the process.
- *Shared narrow focus*: The UI enables the user to focus on a narrow context at a time (e.g. changing part of the query), while the system keeps track of that focus and provide contextual recommendations at every step.

Researchers often discuss personalization of search and recommendations in two distinct contexts: *customization* with respect to long term user models, and *adaptation* to short-term user actions. The mechanisms that we propose here belong to the latter category. However, we believe that this system would benefit from long-term user models as well, and so we pose that potential integration as an open question for future work.

We implemented a prototype of *Focus* and demonstrate its feasibility in the e-commerce domain. The system powers its recommendations with data aggregated from billions of transactions on *eBay*. Next, we go through a motivating example, followed by an overview of the runtime architecture and offline processes that build the system's knowledge-base.

2 User-Interface Interaction

The paper is accompanied by a video demonstrating the user-interface interaction[1]. In preparing a demo system we were forced to make certain UI design choices, but the claims and architectural commitments we make in this paper are more abstract in nature and they should not be evaluated limited to this particular interface.

[1] http://youtu.be/RgoLvGf96Bg

The user starts the interaction by typing an initial query to a search box, or by selecting a group of filters from the landing page of the experience. This landing page is currently hand-curated but a natural extension would be to construct it by determining popular clusters of items, or by personalizing it based on past experience of the user. Subsequently, the system will respond by showing relevant results for the selected set of filters and recommends potential changes to the query. After every user action, the system retrieves new results and updates its query recommendations. These recommendations are incremental, targeting only the specific user selected parts of the query.

At the heart of the communication between the user and the IR system is a user interface that represents the query, along with changes recommended by the system. This acts as shared knowledge between the user and the system, and helps to align their goals. This assisted search approach is similar to widely used autocomplete functionality in recommending changes to a query, but it differs in two key ways. First, the query is partitioned into named entities. Additionally, recommended changes are incremental and can target any selected segment of the query. This provides the user with multiple directions for changing the query in incremental fashion, resulting in a more interactive experience, which allows the user to navigate through the inventory efficiently.

The user interface (Figure 1) consists of three elements: *search context* (top two rows), which characterizes the current query for information retrieval, *recommendations for changing the query* (dark bar), which list the recommendations to lead the user towards popular parts of the inventory, and a *result set* consisting of items that best match the search context.

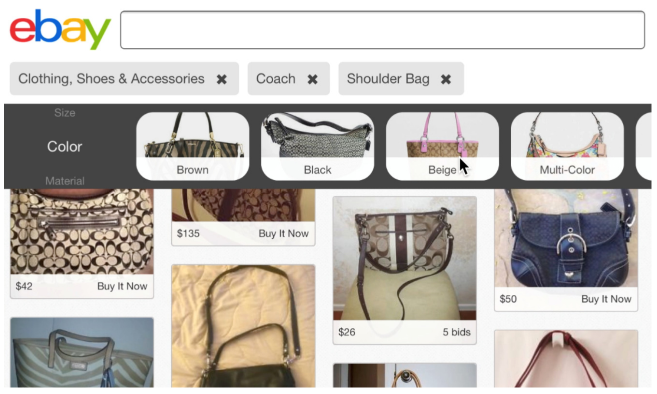

Figure 1. *Focus* interface - refinement

The search context includes a *keyword query* and a set of *selected filters* consisting of attribute name-value pairs. The search context is constructed by user's initiative but the recommender system helps by suggesting new refinements and possible changes to the existing query. Whenever the user changes the search context (either by direct editing or by selecting a system recommendation), the system updates the search results that match that new context.

For example when the user enters the query "coach shoulder bag," the system recognizes the attributes *brand=Coach*, style=*Shoulder Bag*, and *category=Clothing Shoes and Accessories* by applying a named entity extraction mechanism, which resolves ambiguities by considering co-occurrence frequencies in historical transactions. Next, the system updates the interface with these filters and shows relevant results from the inventory. Using this information, the system also recommends new attribute filters such as *color* and *shoe size* since those are filters that work well with the selected filters (in terms of increasing the likelihood of retrieving desirable inventory). At this point, the user can click on one of these attributes (e.g. *color*) and get an ordered list of recommended values that work well with the previous set of filters (Figure 1). In this example, *brown* is at the top of that list because it is a popular color for coach shoulder bags.

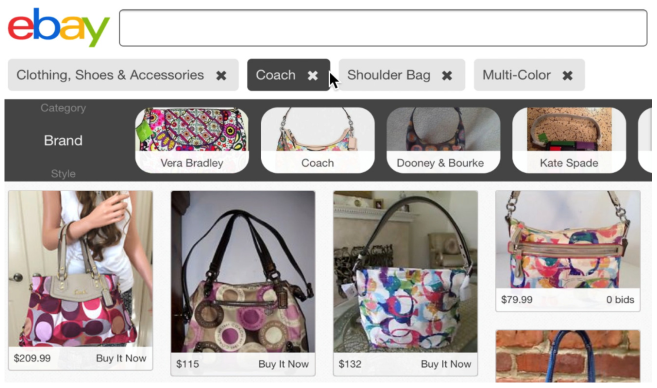

Figure 2. *Focus* interface - pivoting

The user can also select one of the existing filters and get alternative recommendations that work best with the current search context (Figure 2). For a given selected filter, the values are recommendations that lead the user towards popular parts of the inventory. Those recommendations are determined dynamically based on past popular user activity statistics, which we will describe in the next section. For example, clicking on *brand* filter might retrieve the value *Vera Bradley* as another popular brand, which also correlates with *shoulder bags*. Similarly, clicking on style would trigger recommendations for popular item styles for *Coach,* including *totes* and *messengers*.

Finally, the user can continue typing new keyword queries. During this process, the system may suggest autocompleting the keyword (e.g. the partial query "bl" could trigger the filter selection *color=black*). This autocomplete mechanism also prefers entries that work best with the current search context, among the ones that match the partial query.

3 Architecture

Focus architecture is partitioned into data storage, runtime system, and offline data generation (Figure 3). Next, we review these components and explain how they support the functionality described in the previous section.

3.1 Runtime Performance System

The runtime system includes a number of services that respond to user actions. The interface state consists of search context, query change recommendations, and search results. The user actions change the search context and the runtime performance system responds by changing the result set and recommendations.

The runtime performance system consists of three main components. First, the item retrieval system returns a set of items from the inventory that best match the search context. Second, the attribute extraction service extracts a number of attribute-value pairs from a keyword query. Finally, the third service recommends changes on the query given the search context. In this work, we are focusing only on the third component that generates the recommendations and therefore we will not describe the remaining components in more detail.

The query recommendations component consists of a number of services, which all utilize the same backend data storage and inference engine. The *structured autocomplete* service inputs the search context along with a partial keyword query, and returns filter values that match the partial keyword query, preferring popular filters that co-occur with selected filters in the search context in past transactions. The *attribute name refinement* service returns most popular attribute names to be used for further filtering given selected filters. Finally, the *attribute value selection* service return best values for a chosen filter given the selected filters in the search context. All of these services use the same backend engine that stores information about popularity of attributes and their interaction.

3.2 Backend Reasoning Engine

All query recommendation services rely on the backend reasoning engine. Each service uses this engine to retrieve candidate values and order them based on a single popularity metric. We will first describe this popularity metric and then we will discuss the retrieval process specific to each service.

Candidate Sorting

Our goal is to estimate the conditional probability $P(y \mid x)$ representing the likelihood of the user finding desired items in the result set, when filtering by a name-value pair y, given the selected name-value filters $x = x_1, x_2 \ldots x_n$. For example, given the selected filter vector x=<*brand=nike*, *type=shoe*>, candidates for y may include constraints like *color=white* or *size=10*. In that case, the recommendation engine would need to determine whether *color=white* or *size=10* is a better refinement. The algorithm merely sorts candidate instantiations for y by maximizing $P(y \mid x)$ for a given x. We achieve this by Naïve Bayesian inference, where we have:

$$P(y \mid x) \sim P(x_1 \mid y)\, P(x_2 \mid y) \ldots P(x_n \mid y)\, P(y)$$

assuming conditional independence:

$$P(x_i\, x_j \mid y) = P(x_i \mid y)\, P(x_j \mid y)$$

Under that assumption, the order of $P(y \mid x)$ for a fixed x depends only on $P(x_i \mid y)$ and $P(y)$ values.

Retrieval

The three recommendation services use the backend engine differently, depending on how they select candidate values of y. For efficient retrieval, all values are stored in a table which has one row for each y instance and has columns observing the following schema:

$$<name(y), value(y), P(y), P(x_1 \mid y), P(x_2 \mid y), \ldots, P(x_n \mid y)>$$

This table has $O(n)$ columns and rows, where n is the number of attribute-value pairs and in our domain. Moreover, we assume that the matrix is sparsely populated, which is an assumption that holds in our shopping domain.

The typical goal is to return y names and/or values (the first two columns), given the probability values. For a set of selected context filters $x = <x_1, x_2, .. x_n>$, all services retrieve only y rows such that:

$$P(x_1 \mid y) \neq 0, P(x_2 \mid y) \neq 0, \ldots, P(x_n \mid y) \neq 0$$

Given a sparse matrix, this significantly reduces the number of candidates and their retrieval becomes very efficient with usual indexing.

On top of the shared retrieval constraints above, the suggestion services use additional constraints. In structured autocomplete with the partial keyword query q, the system enforces the additional constraint *contains*(*name*(y), q), e.g.

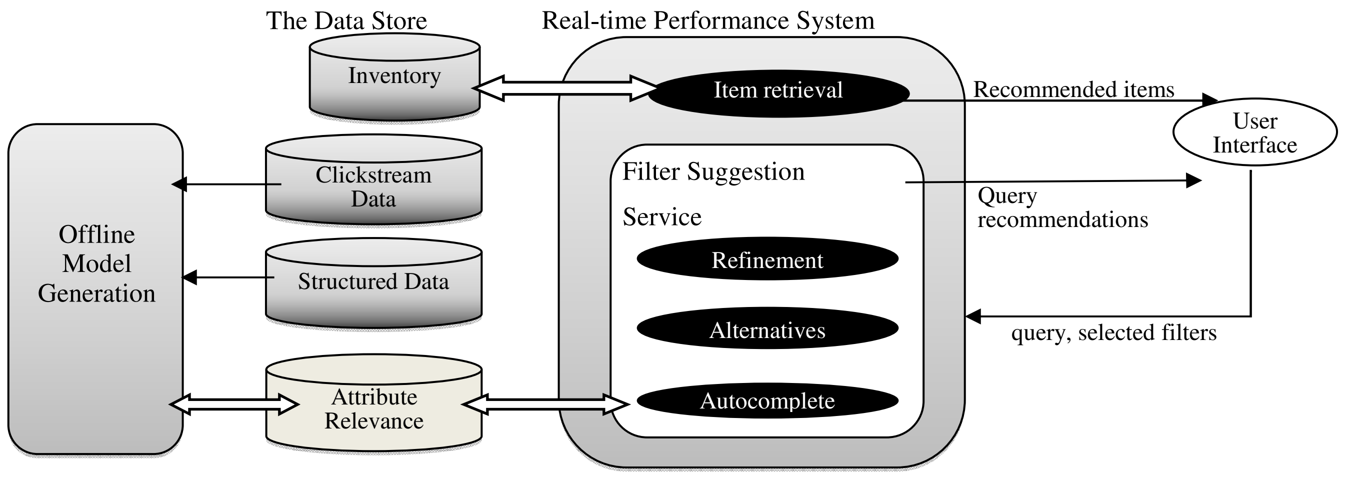

Figure 2. Architecture Overview.

the query "re" retrieves only aspects like *color=red*, where the query is contained in the aspect value. On the other hand, in attribute value refinement, the name *N* of the attribute is given as input, we use the additional constraint *name*(*y*) = *N*, e.g. when the filter *color* is selected, all retrieved *y* rows satisfy *name*(*y*)="*color*". The attribute name recommendations are handled with a slight extension where *y* represents only the attribute name (without value). This fits in our database with minor changes.

3.3 Offline Architecture

The mechanism we defined in the previous section defines a conditional probability metric P(x_i| *y*), which we defined as the probability of selection of *y* (filter name or name-value pair) to be useful for the user, given a selected attribute filter x_i, and P(*y*) the unconditional probability of selection of the filter *y* to be useful. However, we have not described how we can obtain those values in a practical use case.

If we have a system running in production, those values can be obtained based on usage of filters and the engagement with their results. However, it is not clear how we should initialize such a system. In the shopping domain, there are several alternatives that can be used as proxies to these probabilities, each resulting in different runtime behavior. One can use data from engagement with an existing (but static) filter system, or use inventory-based metrics (i.e. how many items in inventory satisfy *brand=nike* and *category=basketball shoes* at a given time). Instead, we decided to optimize our system for popularity. Hence, we calculate the P values by listing item view actions and counting co-occurrence of attribute-value pairs (e.g. number of viewed items with attributes *brand=nike* and *color=white*). Consequently, the resulting system is biased towards preferring items with higher rates of user interaction.

4 Evaluation

Our results are quite preliminary, and we have not yet conducted a systematic user experiment. We implemented a prototype user interface (demonstrated in the accompanying video) powered by real user activity data. While this kind of interface could be useful for different computing environments, its strength is highlighted in touch screen devices, where selecting from suggestions is easier than typing, and there is limited space for suggestions.

The backend data is automatically generated using billions of items viewed at *eBay* and it covers a large portion of items in the inventory that contain attribute data. Our item retrieval service and concept extractor utilizes existing services at eBay. We built the query recommendation services from scratch and they utilize a Solr [4] database for storing and accessing attribute relevance data efficiently.

Related Work

A related approach that is recently gaining importance is conversational search like "Google Now," "Apple Siri," and "Microsoft Cortana," [5] where the IR system also actively interacts with the user, but in these systems there is no explicit and agreed representation of the shared context, and resolving ambiguities and determining which part of past dialog is relevant for the current query remains to be a difficult challenge. Facebook graph search [6] was employing structured autocomplete until recently, where named entities were recommended to complete user queries, but no method is provided for changing parts of a query to reduce low recall. Pinterest's search interface [7] represents query words as tokens, which can be individually deleted and recommends refinement words incrementally but the tokenization does not seem to happen in terms of name-value pairs, and there is no mechanism to change part of a query to alternative values. At the time of submission, Bing researchers described a similar search experience, which also includes pivoting [8]. We find it quite encouraging and validating that other teams have independently built systems motivated by similar ideas.

CONCLUSION

We presented an information retrieval experience for a shopping domain that merges search and recommendations in a novel way. The system facilitates an incremental mixed-initiative interaction between the user and a recommender agent, which leads the user towards popular parts of the inventory, while also satisfying user queries. We implemented a functioning prototype that utilizes data mined from billions of transactions on *eBay* and the backend algorithm is efficient and scalable.

We are at early stages of this work and lack crucial evaluation demonstrating the utility of this system. However, we believe that our contribution is a novel step towards more intelligent interactive information retrieval systems and can encourage future research in this area.

References

1. Manning, C. D. and Raghavan, P. *Introduction to Information Retrieval*, Cambridge Press, USA, 2008.
2. Jannach, D. and Zanker, M. Recommender Systems: An Introduction, 2010.
3. Walker, M., and Whittaker, S. Mixed initiative in dialogue: An investigation into discourse segmentation. *Proceedings of the 28th Meeting of the ACL*, 1990, 70-78.
4. Apache Solr. http://lucene.apache.org/solr/
5. Lison, P. and Meena, R. Spoken dialogue systems: the new frontier in human-computer interaction *XRDS: Crossroads, The ACM Magazine for Students - Natural Language 21* 1 (2014), 46-51.
6. Facebook. http://facebook.com
7. Pinterest. http://pinterest.com
8. Bing – Type less, search images faster. http://blogs.bing.com/search/2015/04/16/type-less-search-images-faster/