(Not Too) Personalized Learning to Rank for Contextual Suggestion

Andrew Yates[1], Dave DeBoer[1], Hui Yang[1], Nazli Goharian[1], Steve Kunath[2], Ophir Frieder[1]
[1]*Department of Computer Science, Georgetown University*
[2]*Department of Linguistics, Georgetown University*
Washington, DC, 20057
andrew@cs.georgetown.edu, dpd29@georgetown.edu
{huiyang, nazli}@cs.georgetown.edu, sak68@georgetown.edu, ophir@cs.georgetown.edu

Abstract

In this work, we emphasize how to merge and re-rank contextual suggestions from the open Web based on a user's personal interests. We retrieve relevant results from the open Web by identifying context-independent queries, combining them with location information, and issuing the combined queries to multiple Web search engines. Our learning to rank model utilizes three types of profiles (a general profile, a city profile, and a personal profile) to re-rank and merge the results retrieved from the Web. We find that the learning model generates better results when the user profiles' weights are biased heavily towards major personal interests. The detections of major, minor and negative personal interests are done by statistical analysis across users, examples, and context-independent query types. For user interests detected by query types, we call the interests "macro-level interests", while for user interest detected by examples, we call them "micro-level interests". In our experiments, we find that "micro-level interests" effectively avoid favoring too much towards rare query types such as spa and game, and thus yields more balanced rankings. Finally, for the top ranked suggestions for each user and context, we generate result descriptions by learning to rank favorable Yelp comments and using a natural language generation algorithm to generate positive comments.

1. Introduction

In this work, we emphasize how to merge and re-rank contextual suggestions from the open Web based on a user's personal interests. We first identify context-independent queries from the given Toronto examples based on mapping to a two-level Yelp-like ontology. The context-independent queries combined with new location information are sent to multiple Web search engines, such as Google, Yelp, Yellow Pages, and Bing, to crawl and build a large pool of potential contextual suggestions. A learning to rank model is then used to merge and re-rank those potential suggestions from the pool for each query, context, and user profile combination based on a heterogeneous set of features. A resource merging algorithm is used to merge the results from all queries within each context and user pair. Finally, for the top ranked suggestions, besides extracting the top rated Yelp comments as their descriptions, we design a natural language generation algorithm to generate positive comments for the suggestions.

The learning to rank model utilizes three types of profiles: a general profile based on the training suggestions given in the Toronto examples, a city profile containing well-known attractions for each city, and a personal profile based on a user's personal interests. Since the pool of potential contextual suggestions is crawled from major search engines, a problem that we face is that many suggestions are local stores that seem 'not well-known' and therefore are 'not attractive' to visitors who may want to see the major attractions in a new place. Our use of general profiles and city profiles resolves this issue by improving the availability and ranking of famous attractions, thereby providing more balanced recommendation to a visitor.

Moreover, the learning model makes distinctions among major personal interests, minor personal interests, and negative personal interests for all personal profiles. We find that the learning model generates better results when the weights are biased heavily towards major personal interests. The detections of major, minor and negative personal interests are done by statistical analysis across users, examples, and context-independent query types.

For user interests detected by query types, we call the interests "macro-level interests", while for user interest detected by examples, we call them "micro-level interests". In our experiments, we find that "micro-level interests" effectively avoid favoring too much towards rare query types such as spa and game, and thus yields more balanced rankings.

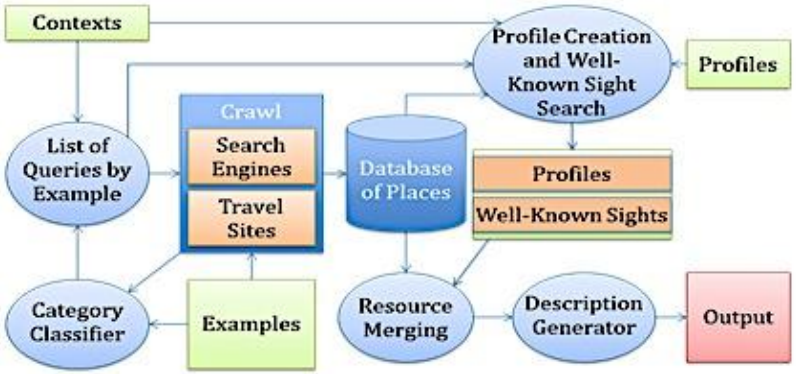

Figure 1: System Architecture

Our system consists of three primary components: a *query generator* that utilizes provided examples and crawled pages to formulate queries; a query ranking component that produces a ranked result set for each context and user pair, and a *description generator* that produces descriptions for our final results.

2. Query Formulation

As the overall task is to provide entertainment suggestions based on user preferences for the provided examples, we generated the queries using information from those examples. The example data consisted of a title, a description, and a url to the example website. We implemented various methods, including using phrases from the title, phrases from the description, and phrases from the website. We also utilized Web resources, such as categories assigned at Yelp for the provided examples. Among these, the Yelp categories seemed to perform the best as to generating a good query from an example. We observed two shortcomings in utilizing the Yelp categories:

First, the Yelp categories could be very specific, such as "Sushi Bar" for examples where we would also want to consider the query "Restaurant". This problem was solved using a mapping from Yelp categories to more general categories. The second problem was that not all of the examples had Yelp pages, and so we did not have a Yelp category for those examples. To automatically assign categories to examples we generated representative documents for each category and then used the probabilistic retrieval strategy BM25 [4] with the example title and description as the query to determine which representative document was most relevant to the query. The category whose document was found to be most relevant was then assigned to the example. The representative documents were generated by querying Google using the category, taking the words from the snippets that Google displays, and retrieving any Wikipedia articles that appear. From the snippets and Wikipedia articles we obtained word counts, and from the list of 100 most frequent words we manually extracted words for the document, weighted by their counts.

3. Profile Creation

Our ranking algorithm uses three types of profiles: general, personal, and city; the general and personal profiles consist of a list of categories with weights for each category, while city profiles are ranked lists of well-known attractions in a city. We tested two different types of profiles: one with the more specific Yelp categories, and one with the more general categories. The profiles were created as follows:

1. General Profile: This profile was created by counting each training example for a given category and using that count as the weight (normalized by the total number of examples). The purpose of the profile was to bias the result categories to obtain a distribution of categories equal to that of the example set.
2. Personal Profile: A personal profile was created for each of the user profiles given. These profiles were meant to reflect user interests and were used to bias our algorithm towards suggestions that the user was likely to favor. We had two different methods for creating these profiles: macro-level interests were detected by query type, and micro-level interests were detected by examples. The algorithm for each was the same, but with a subtle difference at step 2:
    a. Step 1: To determine a category score, we first built a matrix with each cell being the count of positive reviews for a given profile for a given category
    b. Step 2: For Macro, these counts were normalized by the total number of that category in the training examples. For Micro, these counts were normalized by the total number of examples.

c. Step 3: From this we determined the average user count and the average category count.
d. Step 4: We assigned a weight to each cell in the matrix. If the cell was greater than both the profile average and the category average, it received the highest weight (w1). If it was higher than the row average only, it was given second highest weight (w2). If it was not higher than either, but still non-zero, it was semi-preferred (w3). If it was zero, then it was not preferred (w4).
e. Step 5: We multiplied this weight by the category probability (*Pr(Example|Category)*), and then by the profile probability (*Pr(Positive Rating|Profile and Category)*). The weights we used were w1 = 2.0, w2 = 0,7, w3 = 0.3, and w4 = 0.0 with the micro method.

3. City Profile: The purpose of the city profile is to include well-known attractions that a city might be known for. For example, Central Park is a good suggestion for anyone visiting New York. We crawled each city's page on aviewoncities.com, wikitravel.org, and Wikipedia to find results that were mentioned on the page. We labeled each of these results as a well-known attraction. To create a ranked list of attractions for each city, each city's well-known attractions were ranked by the attraction's rank on aviewoncities.com and the number of Web sites the attraction was found on.

4. Crawling & Database Design

4.1 Crawling & Database Design

We used a relational database to store information about resources found on the Web. Our schema consisted primarily of tables for locations, users, and user rankings of locations. Location records stored information about potential locations for a contextual suggestion and allowed us to store a variety of associated metadata including precise geographical attributes and operating hours. Tables relating to user information stored information concerning a user's preference and links to actual location information. We evaluated Web sites that provided a geographical search capability and constructed a rough scheme of possible metadata fields for each entity including operating hours, location, and review. We designed crawlers to query Web search engines and crawl through the results, extracting as much metadata information as possible, and then inserting the associated data into our database system for later analysis and indexing.

4.2 Crawling Methodology

We settled on five Web search engines that were relevant to our task: Google, Google Places, Bing, Yelp, and Yellow Pages. We issued a query to each search engine for every category, location pair and crawled everything listed in the first 5 pages of results. We used the Nokogiri library (http://nokogiri.org/) to extract information from each search engine's result pages. Each tool extracted the result's title, URL, city, state, zip, address, telephone number, snippet, reviews, and hours of operation from the result's page and stored it in our database. Some result pages did not include all of this information; in those cases we extracted only the available information.

4.3 Filtering

To help avoid returning irrelevant resources as results, we filtered 3rd party pages (e.g. Wikipedia) by removing any result whose URL that was not rated in Yelp or Google Places. We also removed "under construction" and "coming soon" pages by removing pages that did not contain at least one link to another page on the same domain. To help avoid returning duplicate results, we combined results from different search engines that had the same title, URL, or phone number.

4.4 Context

We handled the location aspect of the context by including the location in the query issued to our search engines. We specified the desired location in the search engine's location field, where applicable (e.g., Yelp). For engines that do not have a separate location field, we appended the location to the query.

Each resource's hours of operations were used to handle the time and day aspects of the context. For resources in our list of famous sites, we assumed that theaters, operas, and concert venues were open in the evenings, and we assumed everything else was open in the mornings and afternoon. In cases where no hours of operation were available for a resource and the resource was not a famous site, we assumed the resource was closed at the specified time and did not return it.

We handled the season aspect of the context by ignoring a category, boosting a category, or doing nothing. This approach is based on the ideas that some types of activities are undesirable in the winter (i.e., outside activities) and that some types of activities are more desirable during the warmer seasons (i.e., restaurants and bars that may have outside seating). We did not

account for locations with different weather patterns, such as locations where outside activities would be comfortable all year. The table below shows the action take for each season, category pair. We did not alter the ranking for season, category pairs not shown.

Seasons	*Action*	*Categories*
winter	ignore	amusement parks, botanical gardens, cultural districts, farmers market, landmarks, landmarks and historical buildings, mini golf, tours, walking tours, zoos, shopping, shopping centres, and parks
spring, summer, and fall	boost	cafes, restaurants, bars, dim sum, dive bars, irish, mediterranean, mexican, pubs, tea rooms, and vegetarian

5. Resource Merging

5.1 Learning to Rank

After unwanted resources were filtered and duplicate resources were combined, we used learning to rank to create a result set for each context. To do so, we used SVMRank [3] to rank a list of resources for each query/category. We biased this ranking towards Google's ranking by giving Google's ranking a weight of 0.4 and our ranking a weight of 0.6, so that popular results and famous sites were weighted more heavily. We trained SVMRank using the results from the TREC example context (Toronto). We ranked the example results by their popularity, which was computed as the ratio of final 1's to final -1's in the profiles. We also experimented with computing popularity using only initial ratings and using a combination of both ratings.

For each example result we used the following features, which were based on those used in [1]: the average rating across all search engines; the total number of reviews across all search engines; the mean reciprocal rank across all search engines; the average percentage of query terms appearing in the title and snippet across all search engines; the maximum percentage percentage of query terms appearing in the title and snippet; a boolean indicating whether the sentiment expressed in the example's reviews is mostly positive or negative, determined by using SentiWordNet [2] to look up terms; the sum of the IDF of query terms found in reviews; the sum of the IDF of query terms found in snippets; the total size in words of the reviews; the average size of the title in words across all search engines; the average size of the snippet in words across all search engines; the number of slashes in the URL; the length of the URL in characters; the sum of the BM25 similarity between the query and example's titles; the sum of the BM25 similarity between the query and example's snippets; and a boolean for each search engine indicating whether the example was found by the search engine.

5.2 Personalizing the Learning to Rank Results

Once we had a ranked list of results for each category, we built the merged result set 10 results at a time. Our goal was to include a variety of results from categories that the user liked. For example, if a user gave high scores to only spas, restaurants, and bars, each set of 10 results returned for the user should contain only a mixture of spas, restaurants, and shopping centers. To do this we calculated a score for each example result for each user as described in *Profile Creation*. We combined these scores into per-user category scores by summing the scores for every resource in each category. Categories with a non-positive score were ignored. In each set of 10 results, the number of results from a category was proportional to the category's score with regard to the other categories.

$$Number\ of\ results\ from\ category\ X \propto \frac{score}{\sum_y score_y}$$

5.3 Boosting Well-Known Attractions

We used city profiles to boost well-known attractions to the top of each set of 10 results that we returned. To do so, we added well-known attractions from each category which had a positive user profile score. The well-known attractions which were added to the result set were ranked by their ranking in the city profile. To avoid overwhelming the result set with well-known attractions for locations that have many of them (e.g., New York), we placed constraints on how many well-known attractions could appear in each set of 10 results. This approach allowed us to heavily weight well-known attractions in the

first 20 results while still keeping other types of resources. This approach allowed us to add well-known attractions that were from categories that the user liked while keeping the user's result set from being entirely determined by the user profile.

5.4 Result Set Size

For some contexts it was difficult to produce 50 results given the constraints imposed by the user's profile, the season, and the desired time of day. In cases where it was difficult to compile 50 results that satisfied all our constraints, we slowly relaxed constraints until we had enough results. We first relaxed the season and time of day constraints. That is, we did not filter the results based on the season or based on the result's operating hours. If we still did not have 50 results after relaxing those constraints, we removed the user's profile constraint and replaced it with a general profile constraint. The general profile was computed from all profiles as described in *Profile Creation*. We also experimented with smoothing the user profile scores using the general profile scores and a city profile score.

6. Automatic Natural Language Description Generation

Our goal was to use descriptions that talked about the resource favorably, explained what the resource was, and gave the reader some idea of what could be expected at the resource. We chose each result's description from the favorable reviews on Yelp and Google Places. We used SVMRank [3] to rank the favorable reviews and selected the highest-ranked review as the result's description. We manually ranked descriptions from a test context not included in contexts.txt (Pittsburgh, PA) to create a training set.

The features we used were the review's: word count, character count, percentage of characters that are capital letters, percentage of characters that are numbers, percentage of characters that are not alphanumeric, percentage of words that come from the category's language model, BM25 similarity between the description and the category, and a boolean indicating the sentiment of the review as determined by SentiWordNet [2].

In order to ensure that each resource's description included a reference to the resource's category, we added a sentence to the beginning of the description in the format. This sentence was generated from the resource's name, a list of favorable adjectives, and the resource's category. The structure of the sentence varied with the resource's category, so that the sentence was well-formed for each category. For example, for the "music venue" category sentences were constructed with this structure: "(result name) is (a favorable adjective) (the result's category)," such as "Lincoln Center is a good music venue." Each category was mapped either to this rule or to another rule, which was: "(result name) is (a favorable adjective) place for (the result's category)."

7. Experimental Results

We submitted two runs for judgment in this track: GUINIT and GUFINAL. Both used judgments to find the micro-level interests as outlined above. GUINIT used the initial judgments of a person in the sample profile based on the description of a suggestion. GUFINAL used the final judgments which were given after the person visited the website of the suggestion. We chose these runs because they contained more famous attractions than runs that used macro-level interests and runs that used different user profile weights. Suggestions were judged based on four criteria: geographic appropriateness, temporal appropriateness, interestingness of the website, and interestingness of the description. Geographic and temporal judgments were made by NIST assessors, while description and website interestingness judgments were made by the original pool of students who judged the example suggestions. Judgments were made on a 0 to 2 scale, with 0 being not appropriate/interesting, and 1 being partially appropriate/interesting, and 2 being completely appropriate/interesting. NIST then calculated metrics of MRR and P@5 for different combinations of judgments, with a suggestion only being relevant if it received a 2 for all judgments in the combination. NIST also provided best, worst, and median results for all runs for P@5 of Website, Place and Time, and Website, Place and Time combinations. Based on these results we see that our approach achieved some of the best results for the geotemporal judgments. As seen in Figure 3, our mean scores when taken over the profiles were all above the median, and many times were the best. Figure 2 shows that our system's Website recommendation performance is not as clear. We did do well, with most of our results being above the median, but we were not as consistently at the top of the ratings. However, upon inspection it seems to show that GUFINAL did better for website judgments, which would be expected because its personal profile scores were calculated based on people's rating of suggestion sites.

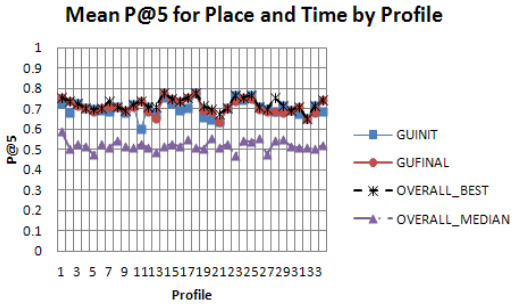

Figure 2

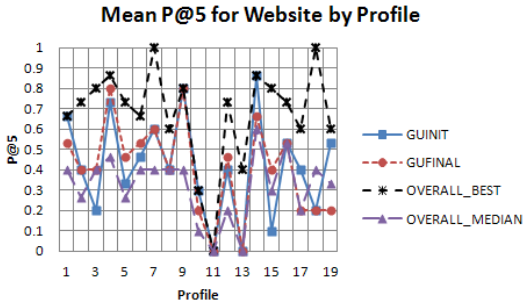

Figure 3

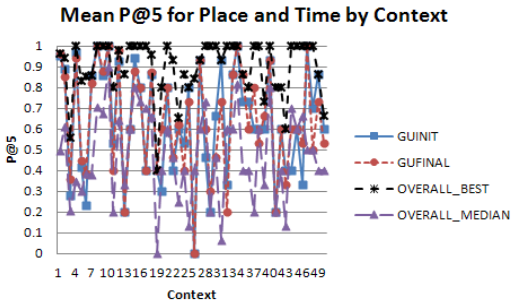

Figure 4

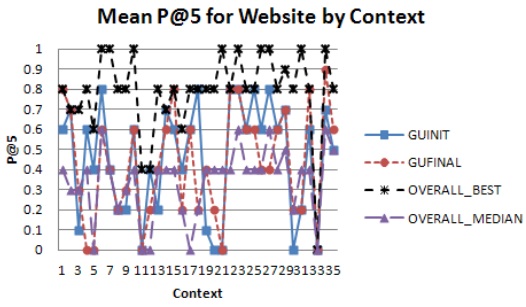

Figure 5

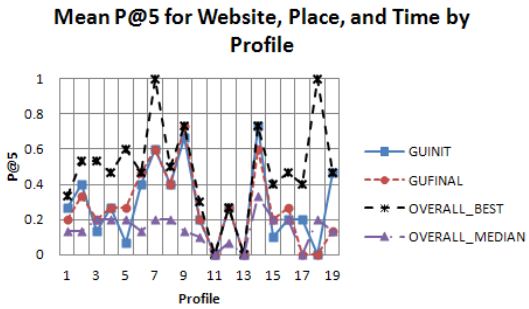

Figure 5

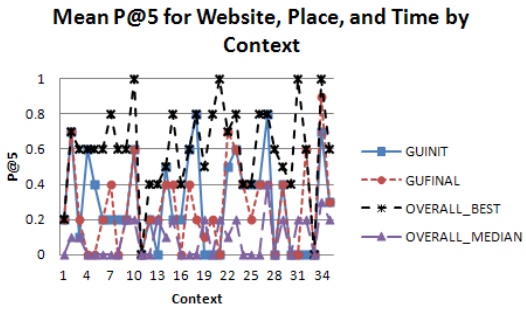

Figure 6

8. Conclusion

We aggregated results from many Web search engines to produce results. This approach allowed us to take advantage of structured contextual information on the Web and thus worked well for the context evaluation. Utilizing multiple search engines also gives enough varied results that we were usually able to find appropriate results for the contexts. We cannot yet tell how well it performed for the profile evaluation, but our hope is that the mixture of personal, general, and city profiles allowed us to personalize the results for each user while avoiding "overfitting" the personal profiles by utilizing general profiles and city-specific profiles. By incorporating general profiles we were able to model general users interests in cases where not enough information was available to produce a full result set for the user profile. Finally, we were able to provide balanced rankings by using micro-level interests, which avoid favoring rare categories (e.g., spas) as macro-level interests do.

References

1. J. Arguello, F. Diaz, and J. Callan. Learning to Aggregate Vertical Results into Web Search Results. In Proc of the 20th ACM Conference on Information and Knowledge Management (CIKM'11), 2011.
2. S. Baccianella, A. Esuli, and F. Sebastiani. SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining. In Proc. Of the 7th Conf. on Language Resources and Evaluation (LREC'10), 2010.
3. T. Joachims. Training Linear SVMs in Linear Time. In Proc. of the ACM Conf. on Knowledge Discovery and Data Mining (KDD'06), 2006.
4. S.E. Robertson, S. Walker, S. Jones, M.M. Hancock-Beaulieu, M. Gatford. "Okapi at TREC-3." In Proc. of the Third Text REtrieval Conf. (TREC'94), 1994.