LET’S BUY BOOKS: FINDING EBOOKS USING VOICE SEARCH

Cheongjae Lee¹, Alexander Rudnicky², Gary Geunbae Lee¹

¹POSTECH, Republic of Korea ²Carnegie Mellon University, USA

ABSTRACT

We describe Let’s Buy Books, a dialog system that helps users search for eBook titles. In this paper we compare different vector space approaches to voice search and find that a hybrid approach using a weighted sub-space model smoothed with a general model provides the best performance over different conditions and evaluated using both synthetic queries and queries collected from users through questionnaires.

Index Terms— spoken dialog systems; voice search; book search; Olympus;

1. INTRODUCTION

The book shopping domain presents interesting issues for spoken dialog systems as the core operation involves search for an often under-specified item, a book for which the user may have incomplete or incorrect information. Thus the system needs to first identify a likely set of candidates for the target item then efficiently reduce this set until it contains that item or items originally targeted by the user. The first part of the process is characterized as “voice search” and several such systems have been described (see below). For the second part of the process a graphic interface can be used to offer selections, alternately the search can be interactively modified to generate progressively better solution sets or to support exploration over a set of potential targets. In this paper we focus on the voice search part of the process and specifically on two sources of difficulty: users not having an exact specification for a target, and queries being degraded through speech recognition errors.

A common problem that a user may not know the exact values for their desired book’s attributes, corresponding to slots in a spoken dialog system form. For example, if the book title is "ALICE'S ADVENTURES IN WONDERLAND AND THROUGH THE LOOKING-GLASS", it is difficult for a user to remember the entire title. In this case, the user might say "I don't know the whole title but it's something like ALICE ANDVENTURE". As we will see below, about 33% respondents in our survey did not have the complete information, while 32% of respondents did not know the exact title of the book they were trying to find. Moreover, many titles are too long to say the whole title even though the user knows the exact string; for example, in our database the maximum title length was 38 words. Thus, users often say a few keywords instead of its exact value. In addition, some titles contain its author’s name and category. There are additional peculiarities. For example, the title “MISS PARLOA'S NEW COOK BOOK” is sort of a book by Ms. Parloa in the cook category, but this title also contains its author’s name and category. This problem may cause performance degradation in spoken language understanding (SLU) because the mapping on input to form slots may be ambiguous. The problem is exacerbated by the large number of eBooks1 that have been published as well as inconsistencies in how the information may be stored.

This paper addresses some practical issues in the development of the Let’s Buy Books system, in particular the development of an effective voice-search component (Section 3).

2. RELATED WORK

Voice search [1] has been used in various applications: automated directory assistance system [2], consumer rating system [3], and multimedia search [4]. Early voice search systems primarily focused on issues of ASR and search problems in locating business or residential phone listings [2]. Recent voice search systems have been applied to search for entries in large multimedia databases [4]. Book search dialog systems have recently been studied by Passoneau et al. [5,6]. These studies have focused on Wizard-of-Oz (WoZ) experiments to design a book search dialog system for use in the public library system. Passoneau et al are able to assume that patrons have exact information for the title, author, or catalogue number because they receive monthly newsletters for new book lists. In addition, a simple Ratcliff/Obersher technique [7] was used to measure the similarity of an ASR string to book titles in the database.

3. THE LET’S BUY BOOKS SYSTEM

We implemented the Let’s Buy Books system using the Olympus/RavenClaw framework [9,10]. To date, this framework has been used to develop and deploy various dialog systems spanning different applications [10].

---

¹ 629,649 eBooks are currently available in the Amazon Kindle Store (retrieved July 5th, 2010).
3.1. Backend Database

The backend of our system contains a relational database consisting of 15,088 eBooks, sampled randomly from Amazon Kindle Books (http://www.amazon.com/Books-Kindle/b?node=154606011) by web crawling. First, each book record was extracted automatically including 17 attributes about its title, authors’ names, categories, price, sales rank, customer review rating, publisher, etc. Abbreviations (e.g. ED., REV., VOL., NO., etc) were expanded into the corresponding full terms. Slots containing non-alphabetic characters such as C++, $2, etc, were transliterated into spoken forms, e.g. C PLUS PLUS and TWO DOLLARS.

Table 1 shows the statistics of the book database used in Let’s Buy Books system. There are 13982 unique titles, 1131 unique categories, and 9589 unique authors in the book database. The average title length is 6.99 words, and the average author length is 2.25 words. These contribute 20882 words to the system vocabulary.

3.2. Speech Recognition & Language Understanding

Let’s Buy Books uses the PocketSphinx decoder, configured with a statistical n-gram language model. The resulting hypothesis is parsed by Phoenix, a robust parser configured with an extended context free grammar [11].

One of the most important challenges in building speech recognition and language understanding modules for the book search system is to define a habitable user language prior to the point at which a prototype system is available to collect actual user data. Often the procedure consists of the developer and a few other volunteers generating likely inputs as language data. Such an approach necessarily introduces a sampling bias. We sought to improve the diversity of this sample by using the Amazon Mechanical Turk (Mturk) service to obtain user utterances at a low cost. See http://www.mturk.com for a description MTurk is an online marketplace for human workers (turkers) who perform tasks, called human intelligence tasks (HITs), in exchange for a small sum of money [8].

MTurk can be used to collect diverse answers to specific questions; we published some HITs to collect the possible utterances given the metadata about a title, author’s names, and a category in response to the question “how can I help you?” as might be posed by a human clerk in a book store. These utterances include one or more slot values of the corresponding book. The basic grammars for Phoenix were then manually written based on an analysis of these utterances. In addition, we augmented the grammar using domain-specific grammars for continuing further dialogs such as searching different books, specifying the books, manipulating a shopping cart and so on. These grammars were independent of the book database.

Some sub-grams for slot values need to be automatically generated from the book database because updating new items periodically is necessary operation to maintain the accuracy of the book search system. To define the slot values for books, their titles were tokenized into a bag of words; title queries have many combinations of words regardless of their orders and grammars because users do not say content words (e.g. ‘ALICE’, ‘ADVENTURE’, ‘WONDERLAND’, etc) without functional words (e.g. ‘IN’, ‘OF’, ‘THROUGH’, etc). The author’s names were divided into the first name, the middle name, and the last name because users can say either the full name or the partial name. For example, either ‘LEWIS’, ‘CARROLL’, or ‘LEWIS CARROLL’ could be said when users want to find some books written by ‘LEWIS CARROLL’.

The n-gram language model in our system was trained by using 100,000 sampled sentences automatically generated using these grammars. This is a routine process and consequently the Let’s Buy Books system can be easily maintained to reflect changes in the book database.

3.3. Dialog Manager

The dialog manager in Let’s Buy Books is implemented using RavenClaw, an agenda-based dialog manager that uses a predefined hierarchical task structure (Figure 1) to control interaction. In our dialog strategy, users speak a query after the system prompts “how can I help you?” The users can search for any books by three slots such as a title, a category, and an author. If the system still has no filled slots after the initial query is provided, the system attempts to fill slots by asking the user to provide the missing information.

Table 1. Statistics of the book database

<table>
<thead>
<tr>
<th>Slots</th>
<th>Title</th>
<th>Author</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max. Length</td>
<td>38</td>
<td>10</td>
<td>5</td>
</tr>
<tr>
<td>Avg. Length</td>
<td>6.99</td>
<td>2.25</td>
<td>1.53</td>
</tr>
<tr>
<td>#Values</td>
<td>13982</td>
<td>9589</td>
<td>1131</td>
</tr>
<tr>
<td>#Vocabularies</td>
<td>13708</td>
<td>8159</td>
<td>1002</td>
</tr>
</tbody>
</table>

Figure 1. Partial view of task structure in Let’s Buy Books
Once a back-end query can be formed (at least one slot is filled), the system returns relevant books using its book search algorithm. Otherwise, the system can either return best-selling books sorted by sales rank, or provide error messages, e.g. "I am sorry that I could not find any books with your query." if no books are returned. If multiple lists are found, the system informs the user of the titles for up to five books, and then suggests one book among them. After the system lists the returned books, the user can reply with a follow-on query including selecting the next rank, searching different books, specifying the relevant books, asking the details of the suggested item, and adding the suggested item to the shopping cart. For example, the user can search for other books by using either the same title, same author, or same category. If the user says "same author", the system retrieves the new lists by the same author again. In addition, if the user needs to narrow the relevant books, the system suggests possible slot values for unfilled slots, and then the user can specify a new slot value to select different items among the books returned previously. Figure 2 illustrates a sample interaction with the Let’s Buy Books system.

4. BOOK SEARCH ALGORITHM

The search problem in Let’s Buy Books is to return relevant books given noisy queries. In this section, we describe how to define a set of slots and how to search for relevant books for the book search dialog system.

4.1. Defining a Set of Slots for Book Search

The book database in Let’s Buy Books consists of various fields including title, authors, category, subject, price, file size, printed pages, and publisher. Although many slots can be used to search for appropriate book, it is not necessarily practical to handle all possible fields. Therefore, a partial set of slots should be defined for use in the book search system. To define the set of slots for use in the system, we surveyed 221 turkers, those who had previously bought eBooks, on which information they typically have when they buy eBooks. The distribution is shown in Figure 3. The top three were title (31.99%), authors (26.10%), and category (14.73%). Consequently only these three slots are considered in our book search.

4.2. Vector Space Model

Vector space models have been widely used to search for appropriate items given user’s query in many voice search systems [2,3]. In our system, we also adopted different vector space models to address the book search problem. Our vector-space search engine uses the idea of a term space, where each book is represented as a vector with specific weights in a high-dimensional space ($v_l$). A query ($q$) is also represented as the same kind of vector ($v_q$). The relevant book lists are identified by calculating the cosine similarity, $s(v_q, v_l)$, between two vectors as follows:

$$s(v_q, v_l) = \frac{v_q \cdot v_l}{\|v_q\| \|v_l\|}$$

If the vectors are normalized, it is possible to compute the cosine similarity as the dot product between the unit vectors.

$$s(\hat{v_q}, \hat{v}_l) = \hat{v_q} \cdot \hat{v}_l$$
This is also important for speedy search because there are many vectors to be computed at every query.

We explored the best vector space model for the book search task, comparing three different models: single vector space model (SVSM), multiple vector space model (MVSM), and hybrid vector space model (HVSM). First, in SVSM, all slots are indexed together over a single term space where every term is equally weighted regardless of its slot name. In this model, slot names may be not necessary for the book search because all query terms are integrated into a single query vector. This model can be robust against SLU errors in which the slot names are incorrectly extracted. This model may be adequate for books in which the title includes its author’s name and category. However, this model cannot capture the relationship between slots. For example, when some users who have no exact book in mind want to find any suitable books, and then to choose one of them for purchase, this type of model cannot take into account the user’s preferences. The next model considered is MVSM in which each slot is independently indexed over different spaces, and all models for slot-type $i$ are interpolated with slot-specific weights, as follows:

$$I^* = \arg\max_i \sum_j w_i^j s_j (v_q^j, v_j^i)$$

Although the interpolation weights can usually be set empirically or by using held-out data, these weights can be modified based on a user’s preferences or on confidence scores derived from recognition. Thus, MVSM can be easily tuned to dynamically improve the odds of generating relevant lists. Nevertheless, incorrect slot names may be critical to the search performance compared to SVSM since MVSM relies on correct input word to slot mapping. We consequently also evaluated a hybrid model, HVSM, in which SVSM is interpolated into MVSM with a specific weight. This model can compensate for the individual drawbacks of the SVSM and MVSM models at the cost of additional computation.

4.3. Index & Term Weight

In representing book information and queries as term vectors, we use stemming to improve search performance, but we do not eliminate stop words because some stop words are necessary and meaningful for identifying relevant books. For example, some titles consist of only stop words such as “YOU ARE THAT” and “IT”. They will not be indexed correctly if stop words are filtered out.

There are several different ways of assigning term weights. One of the best known schemes is TFxIDF (term frequency and inverse document frequency), but we found this scheme did not work well for book search because most values and queries are too short to estimate reliable weights over the true distribution. A simple term count weight was used to represent term vectors in which the weights indicate the counts of term occurrences.

4.4. Search

Queries against SVSM can be generated by concatenating slot values. For example, if the book title was ‘ALICE ADVENTURE’ and the author’s name was ‘LEWIS’, then the query would be ‘ALICE ADVENTURE LEWIS’. A set of relevant books is returned by using a single query. On the other hand, three queries in MVSM can be used to find relevant books in each vector space model such as title, author, and category. In this case, we have to consider unfilled slots because they return no results. Therefore, the weights are renormalized dynamically according to the current slot-filling coefficient $(f)$ that is assigned a value of one if the slot name $i$ is already filled, and zero otherwise, as follows:

$$\hat{w}_j = \frac{f_i w_j}{\sum_j f_j w_j}$$

Finally, HVSM requires all four queries used in SVSM and MVSM. In all models, the dot product is used to measure the similarity between the normalized vectors.

5. SEARCH EVALUATION

5.1. Evaluation Metrics

To evaluate the book search algorithms, we defined two evaluation metrics widely used in information retrieval systems. One is Precision at $n$ ($P@n$), which represents the number of correct queries among the top $n$ relevant lists divided by the total number of queries. For example, $P@100$ means how many queries contain the correct answer by search in the top 100 relevant books. The other is mean reciprocal rank (MRR), which indicates the average of the reciprocal ranks of results for a sample of queries $Q$ [12].

$$MRR = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \frac{1}{rank_i}$$

In reality, there may be multiple correct answers in the lists when users do not have the exact book. For example, some users can search for any fictions without an exact book. However, because it is difficult to automatically determine the relevance relationship between the queries and the lists, we assumed that the lists include a single correct book corresponding to the query.

5.2. Evaluation on Synthetic Queries
We first evaluated the book search algorithms using 2000 synthetic queries which are automatically generated with all slot names. These include three kinds of queries: exact query, partial query, and noisy query (see Table 2). Exact queries are the queries including exact values from the book record. Partial queries consist of some terms randomly selected, up to five, from the exact queries. Noisy queries are generated by using a simple ASR error simulator [13] applied to the partial queries. This error simulator can make errors systematically when given both a specific word error rate (WER) and error type (e.g. insertion, deletion, and substitution) distribution. Thus, various experiments can be performed for preliminarily evaluating the search performance under different WER conditions although simulated queries may differ from real queries produced in real ASR. In our experiment, the weights for MVSM and HVSM were assigned based on our survey result (Figure 3) to reflect user’s reported preferences when they buy eBooks (Table 3).

We first examined the robustness of different models to simulated ASR errors for the book search (Figure 4). Our search engines are fairly robust to ASR error; however, MVSM shows lower accuracy than the other models. However, the performances between SVSM and HVSM were not significantly different. This indicates that SVSM may be a useful way to compensate for recognition errors.

Table 4 also shows how many items returned (#ReturnedItems) by the book search are appropriate for the book search. In this experiment, P@100 and MRR have increased with the number of returned items; however, they have been nearly saturated at 100 returned items. This is encouraging as it means that the search engine does not need to return too many candidates as the desired target answer is likely to occur near the top. This in turn reduces the cost of the subsequent dialog interaction.

The effect of the query length (#QueryTerms) was also investigated as shown in Table 5; increase in the query length improves search performance. This implies that encouraging users to provide longer queries, consisting of over 4 terms, in the book search system can be effective and may be necessary to enhance the search performance, despite the fact that spoken queries are usually shorter than textual queries.

### 5.3. Evaluation on Real Queries

To evaluate the search performance on the utterances collected through Mturk, 623 utterances were parsed using the task Phoenix grammar. 203 utterances among them had no parse results due to the lack of coverage, the rest (420 utterances) were evaluated for the book search. The current grammars cannot extract slot information perfectly although fully parsing the utterance (due to the ambiguities discussed earlier); therefore, even with correct input SLU may introduce errors. In addition, the queries did not contain all slot names because takers had used a partial set of slots. Table 6 shows the evaluation result on real queries with different search models. In this result, HVSM shows much higher accuracy than the others. We believe that HVSM can

<table>
<thead>
<tr>
<th>#ReturnedItems</th>
<th>10</th>
<th>50</th>
<th>100</th>
<th>150</th>
<th>200</th>
</tr>
</thead>
<tbody>
<tr>
<td>P@100</td>
<td>0.8390</td>
<td>0.9365</td>
<td>0.9545</td>
<td>0.9640</td>
<td>0.9725</td>
</tr>
<tr>
<td>MRR</td>
<td>0.7237</td>
<td>0.7642</td>
<td>0.7749</td>
<td>0.7834</td>
<td>0.7889</td>
</tr>
</tbody>
</table>

### Table 4. Search performance vs. the number of returned books (HVSM, WER=30%)

<table>
<thead>
<tr>
<th>#QueryTerms</th>
<th>&gt;4</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>7&lt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>P@100</td>
<td>0.8157</td>
<td>0.8944</td>
<td>0.9647</td>
<td>0.9771</td>
<td>0.9894</td>
<td>0.9885</td>
</tr>
<tr>
<td>MRR</td>
<td>0.4188</td>
<td>0.6631</td>
<td>0.7780</td>
<td>0.7780</td>
<td>0.8492</td>
<td>0.9020</td>
</tr>
</tbody>
</table>

### Table 5. Search performance vs. the number of query terms (HVSM, WER=30%)

<table>
<thead>
<tr>
<th>Models</th>
<th>SVSM</th>
<th>MVSM</th>
<th>HVSM</th>
</tr>
</thead>
<tbody>
<tr>
<td>P@100</td>
<td>0.8699</td>
<td>0.8747</td>
<td>0.9030</td>
</tr>
<tr>
<td>MRR</td>
<td>0.6634</td>
<td>0.6917</td>
<td>0.7227</td>
</tr>
</tbody>
</table>

### Table 6. Search results on real queries (WER=0%)
work well in the book search system because this model can not only reflect the slot information and but it is also robust to SLU errors in which the slot names are incorrectly extracted.

6. SUMMARY & DISCUSSION

We developed the Let’s Buy Books dialog system for eBook search using the Olympus/RavenClaw framework. Some issues in book search were also addressed. We implemented three modeling approaches: SVSM, MVSM, and HVSM, and various experiments were conducted using synthetic queries from ASR error simulator. The experimental results on synthetic queries have shown that SVSM and HVSM can outperform MVSM under various WER conditions. However, HVSM on real queries obtained through Mturk show the best performance. These results mean that slot information may be useful to search more precisely in an actual system and that SVSM, not considering slot names, may be a necessary adjunct to overcome SLU errors.

In addition, we surveyed which information was useful when you have selected one book if similar eBooks were found. We found that many users report that additional attributes such as price and customer review are also important in selecting a particular book among suggested books. Consequently, we still have a re-ranking to do in displaying the top lists for users because it can reflect only the lexical similarity between query and books in the book database.

Finally, additional issues have yet to be resolved before deploying the application in the real world. Dialog strategies can be also explored to efficiently find relevant books when a query has a few terms and noises. In our experiment, longer queries demonstrated better search performance than shorter queries. Therefore, when the query has a few terms, a query expansion might be used to help users by specifying additional slots. Finally, the book search must be improved to be robust to high WER. To improve the robustness, various ASR hypotheses structures (e.g. n-best list, confusion network, etc) can be incorporated. Some features, such as book synopsis and customer reviews, would also contribute to the effectiveness of the search engine by incorporating rich information.

7. ACKNOWLEDGEMENT

This work was supported by the Industrial Strategic technology development program, 10035252, Development of dialog-based spontaneous speech interface technology on mobile platform funded by the Ministry of Knowledge Economy (MKE, Korea).

8. REFERENCE