Vol.2011-SLP-85 No.5
2011/2/4

スロットベースのベクトル空間モデルの組合せによる音声書籍検索

李　清宰[†1]　Rudnicky Alexander[†2]
河　原　達　也[†1]

本稿では、音声認識・言語理解・ベクトル空間モデルに基づいて、電子書籍を検索するための音声対話システム Let's Buy Books について述べる。オンラインサービスの Amazon Mechanical Turk (Mturk) を用いて検索文を収集することにより、文法構築のための発話パターンを分析したり、検索の評価実験を行うことができた。音声書籍検索のためにスロットベースの新たなベクトル空間モデルを提案し、重み付きのスロット毎のモデルと全体のモデルを組み合わせることにより、人工的なデータと Mturk で収集したデータの双方において最も高い検索性能が得られることを確認した。

Combining Slot-based Vector Space Models for Voice Book Search

CHEONGJAE LEE,[†1] ALEXANDER RUDNICKY[†2]
and TATSUYA KAWAHARA[†1]

This paper first presents the development of Let's Buy Books, a spoken dialog system that helps users search for eBooks using the Olympus framework that provides the Pocket-Sphinx for speech recognition, the Phoenix for language understanding, and the RavenClaw for dialog management. To develop this system, we use Amazon Mechanical Turk (Mturk), an on-line marketplace for human workers, to collect queries that are used to predict the possible utterance patterns and to evaluate the book search performance. We also compare different vector space approaches to voice book search and find that combining slot-based vector space models using a weighted sub-space model smoothed with a general model provides the best performance over evaluations using both synthetic queries and real queries collected from users through Mturk.

†1 Academic Center for Computing and Media Studies, Kyoto University, Japan.
Cheongjae Lee is a research fellow of the Japan Society for Promotion of Science (JSPS).
†2 Computer Science Department, Carnegie Mellon University, USA

1. Introduction

The book shopping domain poses interesting challenges for spoken dialog systems as the core interaction 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 matches 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 (Section 2). 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.

One of the major problems is that a user may not know the exact values for their designated book's attributes, corresponding to slots in a spoken dialog system form. For example, if the book title is "ALICE'S ADVENTURES IN WONDER-LAND 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 ADVENTURE". 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 even though the user knows the exact name; for example, in our database the maximum title length was 38 words. Thus, users often say a few keywords instead of its exact value. 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 input to form slots may be ambiguous. The problem is exacerbated by the large number of eBooks [⋆1] 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,

⋆1 800,126 eBooks are currently available in the Amazon Kindle Store (retrieved January 6th, 2011)

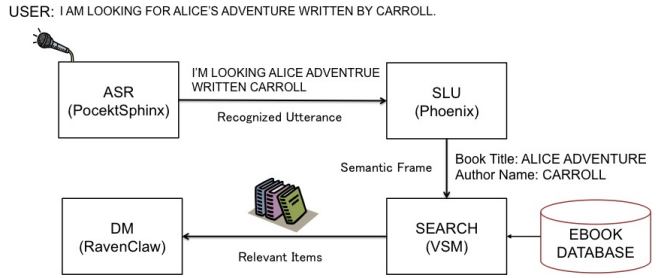

図 1 Overall architecture of Let's Buy Books system.

in particular the models for an effective voice-search component.

2. Related Work

Voice search 8) has been used in various applications: automated directory assistance system 9), consumer rating system 10), and multimedia search 6). Early voice search systems primarily focused on issues of ASR and search problems in locating business or residential phone listings 9). Then, it has been extended to general web search such as Google's. Recent voice search systems have been applied to search for entries in large multimedia databases 6).

A book search dialog system has recently been studied by Passoneau et al. 4). This study has focused on Wizard-of-Oz (WoZ) experiments to design a book search dialog system for use in the public library system. Passoneau et al. assumed 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/Obsersher technique 5) was used to measure the similarity of an ASR output string to book titles in the database.

3. The Let's Buy Books System

We have implemented the Let's Buy Books system using the Olympus/RavenClaw framework 1). To date, this framework has been used to develop and deploy various dialog systems spanning different applications. We also extend this framework to a voice book search system (Fig. 1). In our system, a user's speech is converted into a textual query using the PocketSphinx speech recognizer, and then some task-specific slots, such as book title, author's name, and book category, are extracted from the textual query by the Phoenix semantic parser. After the slots are extracted, the book search module returns relevant items using slot-based vector space models. In this section, we briefly describe how to prototype our system for voice book search applications.

表 1 Statistics of the book database.

Slots	Title	Author	Category
Max. Length	38	10	5
Avg. Length	6.99	2.25	1.53
#Values	13982	9598	1131
Voca. Size	13708	8159	1002

3.1 Backend Database

The backend of our system contains a relational database consisting of 15,088 eBooks, sampled randomly from Amazon Kindle Book[⋆1] by web crawling. First, each book record was extracted automatically with 17 attributes including its title, authors, categories, price, sales rank, customer review rating, publisher, etc. 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 and Language Understanding

The 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.

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 this sample diversity by using the Amazon Mechanical Turk (Mturk) service to obtain user utterances at a low cost. Mturk[⋆2] is an on-line marketplace for human workers (turkers) who perform tasks, called human

⋆1 http://www.amazon.com/Books-Kindle/b?node=154606011
⋆2 See http://www.mturk.com/ for a Mturk description

intelligence tasks (HITs), in exchange for a small sum of money 3).

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, authors, 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 task-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-grammars for slot values need to be automatically generated from the book database because updating new items periodically is a 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 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 (Fig. 2) to control interaction. In our dialog strategy, a user speaks a query after the system prompts ”how can I help you?” The user 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. Once a back-end query can be formed (at least one slot is filled), the system returns relevant books using its book search algorithm. Fig. 3 illustrates a sample interaction with the Let’s Buy Books system.

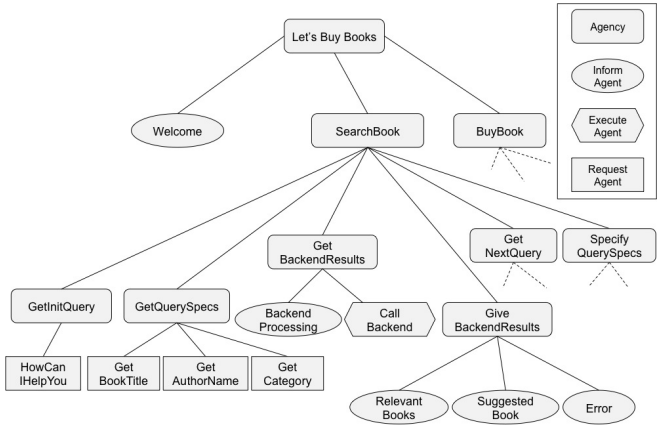

図 2 Partial view of task structure in Let’s Buy Books.

S: WELCOME TO LET’S BUY BOOKS!
S: HOW CAN I HELP YOU?
U: ***I AM LOOKING FOR SOME BOOKS BY LEWIS CARROLL***
S: WHAT IS THE BOOK CATEGORY?
U: ***MAYBE FICTION***
S: JUST A MINUTE. LET ME CHECK THAT FOR YOU
S: 5 BOOKS WERE FOUND SUCH AS ”ALICE IN WONDERLAND AND THROUGH THE LOOKING-GLASS”, ”ALICE’S ADVENTURES IN WONDERLAND THROUGH THE LOOKING GLASS”, · · ·
S: I SUGGEST ”ALICE IN WONDERLAND AND THROUGH THE LOOKING-GLASS” BY LEWIS CARROLL IN FICTION CATEGORY. THE PRICE IS 0.99 DOLLARS.
S: PLEASE SAY WHAT YOU DO NEXT.
[· · · · · ·]

図 3 A sample dialog of Let’s Buy Books.

4. Book Search Algorithm

The search problem in the 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 the 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 books, 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 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 a user query in many voice search systems 9)–10). 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_i). A query (q) is also represented as the same kind of vector (v_q). The relevant book list is identified by calculating the cosine similarity, $s(v_q, v_i)$ between two vectors as follows:

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

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

$$s(v_q, v_i) = \hat{v}_q \cdot \hat{v}_i$$

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 not be 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, i, is independently indexed over subspaces, and each slot-based model is interpolated with slot-specific weights, w_i, as follows:

$$l^* = \underset{l}{\operatorname{argmax}} \sum_i w_i s_i(v_q, v_i)$$

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 speech recognition. Thus, MVSM can be easily tuned to improve the performance of generating relevant lists. Nevertheless, incorrect slot names may be critical to the search performance compared to SVSM since MVSM relies on the correct 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 Term Indexing and 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 Database 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_i) that is assigned a value of one if the slot name i is already filled, and zero otherwise, as follows:

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

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 7).

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

In reality, there may be multiple correct answers in the lists when users do not have the exact book in their mind. For example, some users can search for any fictions without an exact book in their mind. 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 (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 2) 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 (Section 4.1) to reflect user's reported preferences when they buy eBooks.

表 2 Examples of synthetic query; *sub* and *del* represent substitution error and deletion error, respectively.

Title	ALICE'S ADVENTURES IN WONDERLAND AND THROUGH THE LOOKING-GLASS
Exact Query	ALICE'S ADVENTURES IN WONDERLAND AND THROUGH THE LOOKING-GLASS
Partial Query	ALICE'S ADVENTURES THROUGH GLASS
Noisy Query	ALEX$_{sub}$ ADVENTURES THROUGH$_{del}$ GLASS

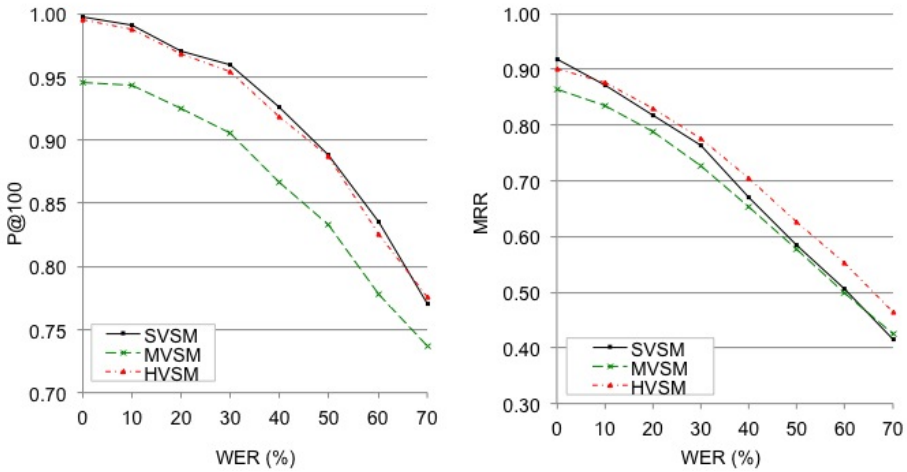

図 4 $P@100$ and MRR of different models.

We examined the robustness of different models to simulated ASR errors for the book search (Fig. 4). Our search engines are fairly robust to ASR errors; however, MVSM shows lower accuracy than the other models. However, the performances between SVSM and HVSM were not

表 3 Search results on real queries (WER=0%).

Models	SVSM	MVSM	HVSM
$P@n$	0.8699	0.8747	0.9030
MRR	0.6634	0.6917	0.7227

significantly different. This indicates that SVSM may be a useful way to compensate for recognition errors.

5.3 Evaluation on Real Queries

To evaluate the search performance on the real queries collected through Mturk, 623 queries (text input) were parsed by Phoenix using the task grammar. Among them, 203 queries had no parse results due to the lack of coverage; the rest (420 queries) 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 turkers had used a partial set of slots. Table 3 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 work well in the book search system because this model can not only reflect the slot information but also fall back to SVSM in case of SLU errors in which the slot names are incorrectly extracted.

6. Summary and 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. Some experiments were conducted using synthetic queries from an ASR error simulator. The experimental results on synthetic queries have shown that SVSM and HVSM can outperform MVSM under various WER conditions. However, HVSM shows the best performance on real queries obtained through Mturk. 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 need to resolve a re-ranking problem in displaying the top lists for users because the current search can consider only the lexical similarity between query and attributes' values in the book database.

Finally, additional issues have yet to be resolved before deploying the application in the real world. One of them is that the book search must be improved to be robust to high WER. To improve the robustness, various ASR hypothesis 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.

参 考 文 献

1) Bohus, D. and Rudnicky, A.I.: The RaveClaw dialog management framework: Architecture and systems, *Computer Speech and Language*, Vol.23, No.3, pp.332–361 (2009).
2) Jung, S., Lee, C., Kim, K. and Lee, G.G.: Data-driven user simulation for automated evaluation of spoken dialog systems, *Computer Speech and Language*, Vol.23, No.4, pp.479–509 (2009).
3) Marge, M., Banergee, S. and Rudnicky, A.I.: Using the Amazon Mechanical Turk for transcription of spoken language, *Proc. ICASSP*, pp.5270–5273 (2010).
4) Passonneau, R.J., Epstein, S.L., Ligorio, T., Gordon, J.B. and Bhutada, P.: Learning about Voice Search for Spoken Dialogue Systems, *Proc. NAACL*, pp.840–848 (2010).
5) Ratcliff, J.W. and Metzener, D.E.: Pattern Matching: The Gestalt Approach, *Dr. Dobb's Journal*, Vol.46 (1988).
6) Song, Y.-I., Wang, Y.-Y., Ju, Y.-C., Seltzer, M., Tashev, I. and Acero, A.: Voice Search of Structured Media Data, *Proc. IEEE ICASSP*, pp.3941–3944 (2009).
7) Voorhees, E.M. and Tice, D.M.: The TREC-8 question answering track evaluation, *Proc. Text Retrieval Conference TREC-8*, pp.83–105 (1999).
8) Wang, Y.-Y., D.Yu, Ju, Y.-C. and Acero, A.: An Introduction to Voice Search, *IEEE Signal Processing Magazine*, Vol.25, No.3, pp.29–38 (2008).
9) Yu, D., Ju, Y.-C., Wang, Y.-Y., Zweig, G. and Acero, A.: Automated Directory Assistance System, *Proc. INTERSPEECH*, pp.2709–2712 (2007).
10) Zweig, G., Nguyen, P., Ju, Y.-C., Wang, Y.-Y., Yu, D. and Acero, A.: The Voice-Rate Dialog System for Consumer Ratings, *Proc. INTERSPEECH*, pp.2713–2716 (2007).