INTERSPEECH 2007, pp. 2709-2712, Antwerp, Belgium, 2007

Automated Directory Assistance System - from Theory to Practice

Dong Yu, Yun-Cheng Ju, Ye-Yi Wang, Geoffrey Zweig, and Alex Acero

Microsoft Research, One Microsoft Way, Redmond, WA 98052, USA

{dongyu, yuncj, yeyiwang, gzweig, alexac}@microsoft.com

Abstract

The automated directory assistance system (ADAS) is traditionally formulated as an automatic speech recognition (ASR) problem. Recently, it has been formulated as a voice search problem, where a spoken utterance is firstly converted into text, which in turn is used to search for the listing. In this paper, we focus on the design and development of the utterance-to-listing component of ADAS. We show that many theoretical and practical issues need to be resolved when applying the basic idea of voice search to the development of ADAS. We share our experiences in addressing these issues, especially in pre-processing the listing database, generating a high performance LM, and developing efficient, accurate, and robust search algorithms. Field tests of our prototype system indicate that an 81% task completion rate can be achieved.

Index Terms: speech recognition, directory assistance, voice search, TFIDF, spoken dialog system, vector space model

1. Introduction

An automated directory assistance system (ADAS) [1] [2] [3] [5] [6] is a spoken dialog system that provides the caller with the phone number and/or address of the business or residential listing he/she requests. It is a very complicated system that involves automatic speech recognition (ASR), listing lookup, disambiguation, and dialog design. The core element of the ADAS is the utterance-to-listing (U2L) component that maps an utterance o to a listing $l \in \mathbb{L}$, where $\mathbb{L} = \{l_i | i = 1, \dots, L\}$ is the set of listings known to the system. An important design goal for the ADAS is to find the decision rule $l = d(o)$ that minimizes the average error rate

$$R = \sum_{j=1}^{N} \delta\left(l_j, d(o_j)\right) \quad (1)$$

where l_j is the reference listing of the jth utterance o_j, N is the total number of utterances in the evaluation set, and

$$\delta(x, y) = \begin{cases} 0 & \text{if x=y} \\ 1 & otherwise \end{cases}. \quad (2)$$

The U2L task was traditionally formulated merely as an ASR problem, where the utterance o is directly mapped to the listing l by the ASR [1] [2] [3]. In other words

$$\begin{aligned} d(o) &= \arg\max_l p(l|o) \\ &= \arg\max_l p(o|l)\, p(l), \end{aligned} \quad (3)$$

where $p(o|l)$ is the acoustic model (AM) probability and $p(l)$ is the language model (LM) probability. In its simplest form, the LM is just the listing unigram and is constructed as a probabilistic context free grammar (PCFG).

It has been observed, however, the automation rate of the ADAS with this simple formulation is extremely low due to the fact that callers usually don't know, don't remember, or don't say the exact name listed in the directory [1] [2]. Instead, different callers may refer the same listing in many different ways. For example, the listing *Kung Ho Cuisine of China* usually is referred as *Kung Ho*, *Kung Ho Restaurant*, or *Kung Ho Chinese Restaurant*. In [4], this has been dealt with through the use of "unique signatures" that are different word sequences which uniquely identify a listing, but we have found that even this method is somewhat fragile.

A natural improvement to the ADAS design is to model the different expressions callers actually use in referring a listing. This leads to the decision rule

$$\begin{aligned} d(o) &= \arg\max_l p(l|o) \\ &= \arg\max_l \sum_w p(o|w)p(w|l)\, p(l) \\ &\approx \arg\max_l \max_w p(o|w)p(w|l)\, p(l) \\ &= \arg\max_l \max_w p(o|w)p(w, l), \end{aligned} \quad (4)$$

where w is the word sequence callers use to ask for the listing l, and the joint probability $p(w, l)$ is taken as the LM [1].

This formulation leads to two difficult issues. First, the estimation of the joint probability $p(w, l)$ is not trivial and usually requires collecting a huge number of real calls. Second, the approach is not scalable [2] given that there are numerous ways in which different callers may refer the same listing and there are more than 18M listings in the US yellow page alone. As a result, this approach is practical only when the listing database is small (e.g., when the system only automates the frequently requested listings [1]).

To solve these problems, a voice search formulation of the U2L task has been proposed by Natarajan et al. [2] recently. In this formulation, a spoken utterance is firstly converted into text, which in turn is used to search for the listing. This new formulation provides a promising framework in developing a scalable high performance ADAS.

In this paper, we focus on the design and development of the U2L component in the ADAS targeting on the business listings. We show that many theoretical and practical issues need to be resolved when applying the voice search technique to the development of ADAS. We share our experiences in addressing these issues, especially in pre-processing the listing database, generating a high performance LM, and developing efficient, accurate, and robust search algorithms.

This paper is organized as follows. In Section 2, we formulate the U2L task as a voice search problem. In Section 3, we discuss the listing database pre-processing strategies. We describe the LM generation approaches in Section 4 and illustrate the search algorithm in Section 5. In Section 6, we show the field test results. We conclude the paper in Section 7.

2. ADAS - Voice Search Formulation

As indicated in Section 1 and in [1] [2], the motivation behind the voice search formulation of the ADAS is to avoid the estimation of the joint probability $p(w, l)$ and to alleviate the scalability problem. The basic idea of the voice search formulation is to use the stochastic N-gram LM to generalize different ways of referring listings and to make the LM compact. This is similar to using the N-gram LM in the dictation task. Since the goal of the ADAS is to find the

listing, a second step of mapping the recognized text to the listing is needed as we will discuss next.

With the voice search formulation, the decision rule (4) becomes

$$d(o) \approx \underset{l}{\operatorname{argmax}} \max_{w} p(o|w)p(w,l) = \underset{l}{\operatorname{argmax}} \max_{w} p(o|w)p(w)p(l|w), \quad (5)$$

where $p(w)$ is a domain specific N-gram LM that is not associated with listing l, and $p(l|w)$ is a search model that maps the word sequence w to the listing l. One benefit of this formulation is that the ASR and the search engines can be decoupled and formed into a pipeline where the N-best results from the ASR are fed into the search engine for processing. If only the top ASR result is used (esp. if the ASR result has been confirmed by the user), the decision rule (5) simplifies to

$$d(o) \approx \underset{l}{\operatorname{argmax}}\, p(l|\hat{w}), \quad (6)$$

where

$$\hat{w} = \underset{w}{\operatorname{argmax}}\, p(o|w)\, p(w) \quad (7)$$

is the top ASR result. In practice, (6) can be applied to each entry in the ASR N-best list to form the final N-best result.

Figure 1 depicts the architecture of the U2L component in the ADAS with this formulation. In this architecture, the utterance is firstly converted into ASR N-best results. The ASR results are then fed into the search component and the N-best list that combines both the ASR score and the search score is returned. The results may need to be disambiguated if the utterance is not sufficient to identify a single listing l.

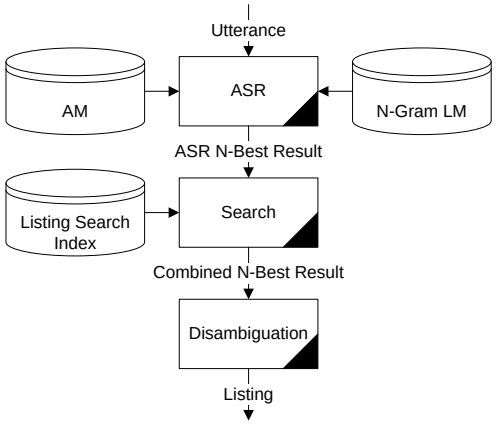

Figure 1: architecture of the U2L component in the ADAS.

3. Automated Data Pre-Processing

Before building the LM and the search index, it's extremely important to correctly pre-process the entries in the listing database for the following reasons.

First, the entries in the listing database are usually optimized for visual presentation, which is different from the way that callers would ask for them. For example, the listing *Gates Mason MD* is often asked by callers as *Doctor Mason* or *Doctor Gates Mason*.

Second, listing entries may contain abbreviations such as *Co*, *LLC*, *MD*, *RN*, *CPA*, *Corp*, *&*, *St.*, and *Ave*. These abbreviations usually need to be normalized for the ASR engine to generate the correct pronunciations with some exceptions such as *LLC*. Note that some of the abbreviations need to be normalized differently under different contexts. For example, *St.* in *St. Paul Cathedral* should be normalized as *Saint*, while *St.* in *First St. Cafe* should be normalized as *Street*.

Third, listing entries may contain spelling and typographical errors which need to be fixed. For example, there can be more than 30 different spellings of *locksmith* in the listing database. Having all these different spellings would greatly hurt the performance of the ASR.

Fourth, the same entity may be listed under different names in the listing database. For example, the coffee store *Starbucks Coffee* sometimes is listed as *Starbucks* and sometimes is listed as *Starbucks Coffee*.

The requirement of the normalization is different for the ASR and for the search. For the ASR, the only requirement is to make sure the listing names and street addresses are normalized to the spoken form. In other words, we care about how to rearrange words in the listing, how to expand or drop off abbreviations, and how to identify and convert misspelled words to the correct form. Since normalization rules are context dependent, it is usually conducted using a finite state transducer (FST) [11], with which the output symbols are determined based on the state and the input symbols. Figure 2 illustrates an example FST that normalizes the abbreviation *St.* to different forms depending on the context. In this FST State 1 is both the start and the end state. While in State 1, the FST changes to State 2 after accepting the input *St.*. It remains at State 1 and output whatever observed otherwise. While in State 2, the FST outputs *Saint [name]* if a *[name]* is accepted and outputs *Street [word]* otherwise. After accepting an input at State 2, the FST changes the state to State1. The number of states in the FST can be as high as hundreds of thousands in a typical ADAS system.

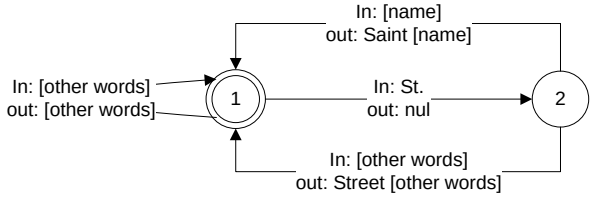

Figure 2: *an example FST for text normalization.*

For the search, the goal of the normalization is to make sure the correct listing can be found no matter how the caller would refer the listing, even when there are ASR errors. The key here is to normalize listings into canonical forms that take into consideration the behavior of the ASR engines. For example, *[word]'s*, *[word]s*, and *[word]* should all be normalized into *[word]*, *Wal-Mart* and *Walmart* should all be normalized as *Wal Mart*, and *JCPenny* and *J C Penny* should all be normalized as *J C Penny*. Here we want to emphasize the importance of breaking the compound words, which occur a lot in the listing database (e.g., *townhouse*, *townhome*, *Sportsworld*, and *Drycleaner*). As an example, there is a listing called *Lionsgate Townhomes* which might be referred by callers as *Lionsgate Townhouses*. Without word breaking, the match score is low even if there is no ASR error since *Townhomes* is different from *Townhouses* or *Town houses*. When there are ASR errors the match score is even lower. For these listings, breaking the compound words in both the search index and the ASR output can greatly increase the search accuracy. We have developed an efficient prefix-tree based automatic word breaking algorithm, where the parent word is the prefix of all its child words. Put it in another way, each child word might be split into two words if the part without the prefix is also a word. If a word has two or more possible breaking possibilities, the one with the highest bigram score is chosen. Word breaking provided us with more than 3% overall accuracy improvement in our offline testing.

4. Language Model Generation

Ideally, the LM should be built from the transcripts of real calls, which demonstrate not only the different ways callers use to refer business listings but also the probability of each such ways. Unfortunately, since there are more than 18M (12M unique names) business listings in the US, it's extremely hard to collect enough real calls to provide a good coverage, especially during the early development phase. For this reason, we estimate the LM probability as

$$p(w) = \lambda p_t(w) + (1-\lambda)p_l(w), \quad (8)$$

where $p_t(w)$ is the LM model built using the transcripts of the real calls, $p_l(w)$ is the LM built using the listing database, and λ is the interpolation weight. λ is set to 0 when there is no real calls available, and increases as the number of real calls increase. The exact value of λ can be determined by minimizing the perplexity of a validation set collected under real usage scenario.

Building $p_t(w)$ is straight forward. The difficult part is building $p_l(w)$ since the entries in the listing database do not reflect different ways callers actually use to ask for the listings, even if the listings are text-normalized. We have given the example of *Kung Ho Cuisine of China* in Section 1. A simpler example is *Microsoft Corporation* which is usually referred as *Microsoft* by callers. If we build $p_l(w)$ directly from the listing database $p(\langle/s\rangle \mid Microsoft)$ (where $\langle/s\rangle$ means the end of the sentence) is extremely low. With this LM *Microsoft* may be misrecognized as *Micro Song* (or something else sounds similar to *Microsoft*) by the ASR.

To improve the quality of $p_l(w)$, we need to predict how the callers will phrase their queries given a listing. A typical approach is to collect many real calls, transcribe them (or to elicit alternative user expressions through a game [10]), and either manually or automatically create rules [6] (sometimes called a *variation model*) from the transcripts. This rule-based approach is usually costly; the rule coverage is highly restricted by the data available; and the rules may be over-generalized without careful crafting.

In our system, we have used a statistical variation model based on the following observations. First, callers usually remember and say the first several words in the listing name but likely to forget or omit the words at the end. Second, caller often say the words that can distinguish the listing from others but omit words that cannot. Third, callers may voluntarily provide information that they think can help identifying the listing, e.g., category or address information.

The basic idea of our approach is to adjust the N-gram counts as follows. First, each listing is weighted using priors retrieved from the query log or search hit if this information is available. Second, each word in the listing name is associated with a skip probability that is assumed to be proportional to the importance of the word in the listing. The importance of the word is determined by its position in the listing and its discriminative ability. Each word at the position i has a position importance of w_l^i, where $0 < w_l \leq 1$ is the position weight. The discriminative ability of the word is determined using the inverse document frequency (which will be covered in Section 5), maximum entropy (MaxEnt) model [7], or similar technologies. Third, each word has a probability to transfer to words that carry the category information. The transfer probability is assumed to be correlated to the word importance and the category (e.g., *restaurant*) indication ability (based on the mutual information) of the words.

There are several other issues to consider when building the LM. Due to the large number of business listings in the US, it's a good idea to build an LM for each city to reduce the perplexity. When building the city specific LMs, listings in the surrounding cities should also be included and/or the LM of the whole US should be interpolated since many times the callers do not know the exact boundary of the cities. Note that using city specific LM requires the ADAS to ask the caller the city name first. This is completely acceptable since city name is important disambiguation information and is likely to be asked later if not at the first dialog turn.

Another issue to consider is whether to use a bigram LM or a trigram LM. We have noticed that bigram LM is a good choice for $p_l(w)$ since most listing names are short and the callers may rephrase the names. We have also noticed that using the garbage model [8] can provide additional robustness since side talks and carrier phrases can be filtered.

As a final note in this section, we want to emphasize that no data is better than more real data. The tricks we have described here can bring the performance of the ADAS to a high level. However, further performance improvement may only be achieved by collecting more real calls.

5. Listing Search

The search component in the ADAS is quite non trivial. It needs to be robust to the noisy input because the callers may use carrier phrases and often refer listings by names that are different than that appear in the listing database, and the ASR may introduce errors. Moreover, it has to be very efficient computationally since the listing database to be searched is large and the callers expect that the response is instantaneous. We have tried using the MaxEnt [7] and the term frequency - inverse document frequency (TFIDF) [9] algorithms. We noticed that the MaxEnt algorithm is not practical at this stage due to the training complexity on the 18M listings.

As a result, in our current system the search is conducted using an improved TFIDF algorithm, where each query and listing is represented as a vector of weights $v_i = tf_i \cdot idf_i$. Here, the term frequency

$$tf_i = \frac{n_i}{\sum_k n_k}, \quad (9)$$

with n_i being the number of the occurrences of the term t_i in the query or the listing, and the inverse document frequency

$$idf_i = log\frac{D}{d_i}, \quad (10)$$

with d_i being the number of listings where the term t_i appears, and D being the total number of listings in the database. The weight v_i indicates that terms that occurs frequently in a specific listing and rarely seen in other listings should be weighted higher since they contain more information in identifying the listing from others. For example, in the listing *Microsoft Corporation*, the term *Microsoft* is more important than the term *Corporation* in identifying the listing from others. The similarity between the query vector $\boldsymbol{v}_q$ and the listing vector $\boldsymbol{v}_l$ is defined as

$$s(\boldsymbol{v}_q, \boldsymbol{v}_l) = \frac{\boldsymbol{v}_q \cdot \boldsymbol{v}_l}{\|\boldsymbol{v}_q\|\|\boldsymbol{v}_l\|}, \quad (11)$$

where $\cdot$ is the inner product of vectors. This is the famous vector space model [9] widely used in information retrieval.

Note that $s(\boldsymbol{v}_q, \boldsymbol{v}_l)$ is not a probability and strictly speaking cannot be directly plugged into (5) or (6). However, the decision rule (6) can be changed to

$$d(o) = \underset{l}{\operatorname{argmax}}\, s(\hat{\boldsymbol{v}}_q, \boldsymbol{v}_l) \quad (12)$$

as long as the decision boundaries are the same, i.e.,

$$\underset{l}{\operatorname{argmax}}\, p(l|\hat{w}) = \underset{l}{\operatorname{argmax}}\, s(\hat{\boldsymbol{v}}_q, \boldsymbol{v}_l)\,. \quad (13)$$

To apply the TFIDF to the listing search problem, we need to determine what the terms should be. We have tried using just the unigram words as the terms as well as adding bigram terms. The bigram terms help in some cases and hurt in other cases due to the ASR error and the word order swap in the caller's phrases. Overall it provides about 2% absolute error rate reduction in our offline testing with doubled search time compared to using the unigram terms alone.

To further improve the performance of the search algorithm we have proposed additional enhancements to the basic TFIDF. The first enhancement is about the way to handle the duplicate terms in the callers' queries. Note that since queries are short, each term in the query carries important information, i.e., a repeated term in the query or the listing name is a key indicator of the listing. The second occurrence of the same term should not be treated the same as its first occurrence. Our approach is to give each term in the listing and the query a unique ID. For example, the second 5 in the listing *5 star 5* is noted as *5_2* to distinguish it from the first *5*. With this enhancement, the query *big 5* matches better to the listing *big 5 sporting goods* than the listing *5 star 5*.

The second enhancement is about the category information. Many times the caller voluntarily provides the category information which can be recognized based on the mutual information between the words and the categories. The category can be considered as a term in the TFIDF calculation or another information source. With the addition of the category information, our system can rank the listing *Calabria Ristorante Italiano* higher than the listing *Calabria Electric* when the query is *Calabria Restaurant*.

The third enhancement is about the search efficiency. Given that each term is unique in both the query and the listing in the enhanced TFIDF, we can simplify (12) to

$$\begin{aligned} d(o) &= \underset{l}{\operatorname{argmax}} \frac{\hat{\boldsymbol{v}}_q \cdot \boldsymbol{v}_l}{\|\hat{\boldsymbol{v}}_q\|\|\boldsymbol{v}_l\|} \\ &= \underset{l}{\operatorname{argmax}} \frac{\hat{\boldsymbol{v}}_q \cdot \boldsymbol{v}_l}{\|\boldsymbol{v}_l\|} \\ &= \underset{l}{\operatorname{argmax}} \sum_{tf_{i,q}=tf_{i,l}=1} \frac{idf_i^2}{\|\boldsymbol{v}_l\|}, \end{aligned} \tag{14}$$

where $idf_i^2/\|\boldsymbol{v}_l\|$ is non-negative and can be pre-computed. This allows us to develop an efficient A* like algorithm that only expands and stores the most promising search path. The search takes about 2 ms on average on a 2GHz CPU 2GB RAM Windows XP computer. Note that TFIDF score is also a good feature for confidence calculation in ADAS.

6. Field Evaluation

We have deployed our completely automated prototype ADAS (for the 18M US businesses) built upon Microsoft Speech Server to public to evaluate its task completion rate, which is one of the most important measurements of the successfulness of an ADAS. The results are summarized in Table 1. Out of 808 legitimate calls we have listened to, 655 were successful. This translates to 81% task completion rate. Note that some users retried several times at the listing search turn with the total number of turns tried being 1072, which indicates a 61% turn accuracy rate. About 67% of the failures are due to the ASR errors (esp. because of accents, noise, and out of grammar words) and 33% of the failures are due to the search errors (mostly because the listing is not in our feed or the surrounding cities we have included). We have noticed that if the system cannot provide the caller with the correct listings in the first two trials, it's very unlikely that the call will be successful even if the caller tries it many times.

Table 1. *Field evaluation result.*

	Total	Success	Success Rate
Tasks	808	655	81%
Turns	1072	655	61%

7. Conclusion

Building an ADAS is a challenging task. It involves a combination of acoustic modeling, language modeling, dialog strategy, listing search, confidence measurement, and disambiguation. In this paper we focused our discussion on the U2L component in the ADAS. We described the motivation of formulating the ADAS as a voice search problem and demonstrated many theoretical and practical issues to be resolved to successfully apply the basic idea of voice search to the development of the ADAS. We summarized our experiences in addressing these issues and proposed many practical solutions to data pre-processing, LM generation, and listing search. The 81% task completion rate in the field test is a good indication of the success of our approaches. We believe that the performance of our system will continue to improve as we start to collect more and more real calls.

Although our current system focuses on the business listing search, the experiences we have gain can be easily applied to many other large scale spoken dialog systems where information lookup is part of the task.

8. References

[1] Levin, E., and Mane, A.M., "Voice User Interface Design for Automated Directory Assistance", in Proc. INTERSPEECH, 2005, vol. 3, pp. 2509-2512.

[2] Natarajan, P., Prasad, R., Schwartz, R., and Makhoul, J., "A Scalable Architecture for Directory Assistance Automation", in Proc. ICASSP, 2002, vol. 1, pp. 21-24.

[3] Billi, R., Canavesio, F., and Rullent, C., "Automation of Telecom Italia Directory Assistance Service: Field Trial Results", in Proc. IVTTA, 1998, pp. 11-16.

[4] Jan, E.E., Maison, B., Mangu, L., and Zweig, G., "Automatic construction of Unique Signatures and Confusable sets for Natural Language Directory Assistance Application", In Proc. Eurospeech 2003, pp. 1249-1252.

[5] Seide, F., and Kellner, A., "Towards an Automated Directory Information System", in Proc. Eurospeech, 1997, vol. 3, pp. 1327–1330.

[6] Scharenborg, O., Sturm, J., Boves, L., "Business Listings in Automatic Directory Assistance", in Proc. Eurospeech 2001, pp. 2381-2384.

[7] Berger, A. L., Pietra, S. A. D., and Pietra, V. J. D., "A Maximum Entropy Approach to Natural Language Processing", Computational Linguistics, 1996, 22(1):39-71.

[8] Yu, D., Ju, Y.-C., Wang, Y.-Y., and Acero, A., "N-Gram Based Filler Model for Robust Grammar Authoring", in Proc. ICASSP, 2006, vol. 1, pp. 565-568.

[9] Salton, G., Introduction to Modern Information Retrieval. McGraw-Hill, 1983.

[10] Paek, T., Ju, Y.-C., Meek, C., "People Watcher: A Game for Eliciting Human-Labeled Data for Automated Directory Assistance", Interspeech 2007.

[11] Sproat, R., Black, A., Chen, S., Kumar, S., Ostendorf, M., and Richards, C., "Normalization of non-standard words", Computer Speech and Language, 15(3):287–333, 2001.