HELPR: A Framework to Break the Barrier across Domains in Spoken Dialog Systems

Ming Sun, Yun-Nung Chen and Alexander I. Rudnicky

Abstract People usually interact with intelligent agents (IAs) when they have certain goals to be accomplished. Sometimes these goals are complex and may require interacting with multiple applications, which may focus on different domains. Current IAs may be of limited use in such cases and the user needs to directly manage the task at hand. An ideal personal agent would be able to learn, over time, these tasks spanning different resources. In this paper, we address the problem of cross-domain task assistance in the context of spoken dialog systems, and describe our approach about discovering such tasks and how IAs learn to talk to users about the task being carried out. Specifically we investigate how to learn user activity patterns in a smartphone environment that span multiple apps and how to incorporate users' descriptions about their high-level intents into human-agent interaction.

Key words: cross-domain; user intention; spoken dialog systems

1 Introduction

Smart devices, such as smartphones or TVs, allow users to achieve their goals (intentions) through verbal and non-verbal communication. The intention sometimes can be fulfilled in one single domain (i.e., an app). However, the user's intention is possible to span multiple domains and requires information coordination among these domains. A human user, with the global context at hand, can well-organize the functionality provided by apps and coordinate information efficiently. On the other

Ming Sun
School of Computer Science, Carnegie Mellon University, e-mail: mings@cs.cmu.edu

Yun-Nung Chen
School of Computer Science, Carnegie Mellon University, e-mail: yvchen@cs.cmu.edu

Alexander I. Rudnicky
School of Computer Science, Carnegie Mellon University, e-mail: air@cs.cmu.edu

hand, although intelligent agents can be configured by developers to passively support (limited) types of cross-domain interactions, they are not capable of actively managing apps to satisfy a user's potentially complex intentions, because they do not consider the repeated execution of activities in pursuit of user intentions.

Currently, most human-machine interactions are carried out via touch-screen. Although the vocabularies of recognizable gestures have been expanded during the past decade [8], interactive expressions are still restricted due to the limit of gestures and displays. This limit may affect usability, especially for certain populations, such as older users or users with visual disabilities. By contrast, spoken language can effectively convey the user's high-level and complex intentions to a device. However, the challenges are: 1) understanding both at the level of individual apps and at the level of activities that span apps; and 2) communicating a task-level functionality between user and agent. Our previous work focused on predicting user's follow-up action at app level [25] or understanding the current app-level intention [4]. This paper mainly addresses the high-level intention-embedded language understanding. For example, our proposed model understands that "*plan a dinner with Alex*" is composed of several domains such as YELP, OPENTABLE and MESSENGER. We also enable the system to verbally communicate its understanding of users intentions, in order to maintain a transparent communication channel.

Multi-domain dialog systems have been studied in the past [14, 19], where a classic architecture contains multiple models developed independently for different domains and allows corresponding apps to handle user requests [11, 18, 3, 4]. Given a spoken utterance, a domain detector selects 1) a single domain [10, 18, 25, 4] or 2) several domains based on the functionality in the user request [20, 21]. However, neither of the two approaches considered the user intention behind the multi-domain interaction (i.e., why the user needs this set of domains). Our method bridges the low-level surface forms in cross-domain interactions and the high-level intention in the user's mind to enable systems to support intention realization. Moreover, considering a personal assistant's perspective, we compare personalized models with generic ones based on personal data availability.

The rest of the paper is organized as follows: we first briefly describe a data collection process to gather user's real-life multi-domain tasks. Then we discuss the methodology to discover, recognized and realize user intentions. Two user studies are described later as end-to-end and component-wise evaluation.

2 Data Collection

We undertook a data collection during which the participants in our study agreed to provide a continuous record of their smartphone use over an extended period of time, in the form of operating system events (e.g. app invoked, phone number dialed, etc). To do this we implemented an Android app that logs each event, together with its date/time and the phone's location (if GPS is enabled).

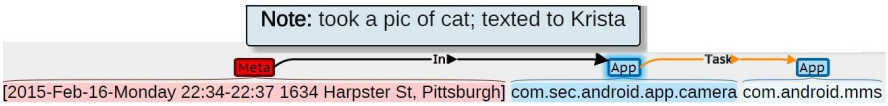

Fig. 1: Example of user annotation

Initial analysis of the data indicated that phone usage could be segmented into episodes consisting of interaction events closely spaced in time. In our pilot data, we found 3 mins of inactivity could be used to group events. Although this parameter appeared to vary across users, we used a single value for simplicity. Participants were asked to upload their log on a daily basis. A privacy step allowed them to delete episodes that they might not wish to share.

Due to multi-tasking, episodes might consist of multiple activities, each corresponding to a specific intent. For example one might be communicating with a friend but at the same time playing a game or surfing the web. We invited participants to our lab on a regular basis (about once a week) to annotate their submitted logs to decouple multiple tasks in the same episodes and also describe the nature (intent) of the tasks (see details below). Note that some activities might also span episodes (for example making plans with others); we did not examine these.

2.1 Smartphone Data Annotation

Participants were presented with episodes from their log and asked to group events into sequences corresponding to individual activities [13] (which we will also refer to as tasks). Meta-information such as date, time, and street location, was shown to aid recall. Participants were asked to produce two types of annotation, using the Brat server-based tool [23]: 1) **Task Structure**: link applications that served a common goal/intention; 2) **Task Description**: type in a brief description of the goal or intention of the task.

For example, in Fig 1, the user first linked two apps (one about camera and another about text message) together since they were used for the goal of *sharing a photo*, and wrote a description "*took a pic of* ___". Some of the task descriptions were quite detailed and provided the actual app sequence executed (see example in Fig 1). However, others were quite abstract, such as "look up math problems" or "schedule a study session". In this paper, we took **task descriptions** as transcribed intent-embedded user utterances since these descriptions are usually abstract. We used these descriptions as data for our intention understanding models.

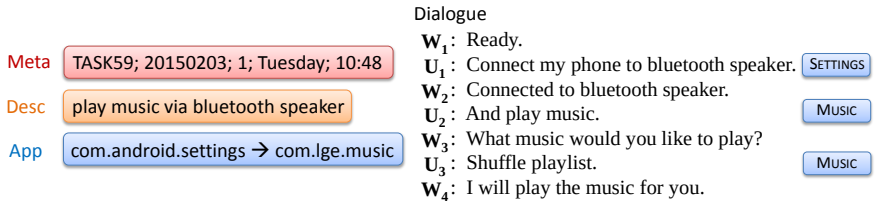

Fig. 2: Multi-app task dialog example. Meta, Desc, App were shown to the participant. Utterances were transcribed manually or via Google ASR. Apps were manually assigned to utterances.

Table 1: Corpus characteristics. Age informally indicates young and old. A native Korean and Spanish speaker participated; both were fluent in English. #Apps is the average number of unique apps. #Multi is the number of tasks which involves multiple user turns.

Category	#Participants	Age	#Apps	#Tasks	#Multi
Male	4	23.0	19.3	170	133
Female	10	34.6	19.1	363	322
Age <25	6	21.2	19.7	418	345
Age ≥ 25	8	38.9	18.8	115	110
Native	12	31.8	19.3	269	218
Non-native	2	28.5	18.0	264	237
Overall	14	31.3	19.1	533	455

2.2 Interactive Dialog Task

We also asked users to talk to a Wizard-of-Oz dialog system to reproduce ("reenact") their multi-domain tasks using speech, instead of the GUI, in a controlled laboratory environment. The users were shown 1) apps used; 2) task description they provided earlier; 3) meta data such as time, location to help them recall the task (see left part in Fig 2). The participants were not required to follow the order of the applications used on the smartphones. Other than for remaining on-task, we did not constrain expression. The wizard (21-year-old male native English speaker) was instructed to respond directly to a participant's goal-directed requests and to not accept out-of-domain inputs. An example of a transcribed dialog is shown in Fig 2.

This allowed us to create parallel corpora[1] of how people would use multiple apps to achieve a goal via both smartphone (touch screen) and language. We recruited 14 participants and collected 533 parallel interactions, of which 455 involve multiple user turns (see Table 1).

[1] Dataset: `http://www.cs.cmu.edu/~mings/data/MultiDomain.tar.gz`

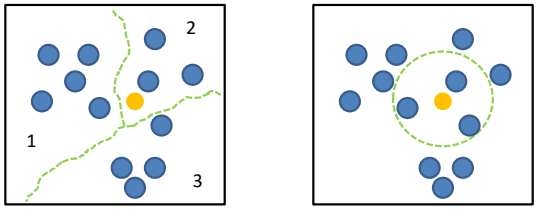

Fig. 3: Illustration of static intention vs. dynamic intention. Blue circles denote training examples and the yellow circle is a testing example.

3 Methodology

For an agent to interact with users at the level of intention, it should 1) **understand** an intention expressed by speech; and 2) be able to **convey** its understanding of the intention via natural language. For example, once the user says "I'd like to plan a farewell party for my lab-mate", the agent needs to know the intention behind this spoken input as well as be able to assist user to find a restaurant (YELP) and schedule time with other lab-mates (MESSENGER). On the other hand, the agent may reveal its inner state of understanding to the user, especially in clarification process. For instance, it may say "I think we are going to plan an evening event, right?" Channel-maintenance with such verbal cues (either implicit or explicit) is helpful in conversation [2]. We first describe modeling intention understanding, then describe the process by which the agent can verbally convey its inner state.

3.1 Models for Intention Understanding

What is user intention? We consider two possibilities. Observed interactions in the intention semantic space may be clustered into K_C groups, each representing a specific intention. We refer to this as the **static** intention. On the other hand, we can also define **dynamic** intention, which is a collection of local neighbors (seen interactions) of the input speech. See Fig 3 as an example. In the static intention setting, the agent is aware of the existence of K_C intentions and their semantics prior to invocation. However, in the dynamic setting, intention is implicitly defined by the K_N nearest neighbors during execution. In both cases, a realization process using the members of the recognized intention set maps the user utterance into a sequence/set of apps to support the user activity.

We anticipate two major differences between statically and dynamically based intentions. First, the static approach can use potentially richer information than just intention-embedded utterances when discovering basic intentions — it could use post-initiate features such as apps launched or user utterances in the spoken dialog. Ideally, this may yield a better semantic space to categorize seen interactions. How-

ever, during execution, the input feature is the same as in the dynamic approach, i.e., task description. Second, the static approach has hard boundaries between intentions. Instances close to the boundaries may not be well characterized by their cluster members.

In both cases the agent will need to map an intention-embedded utterance into steps (i.e., sequence of apps/domains). Several techniques are available. We can combine the individual app sequences of the set members into a single app sequence that represents a common way of surfacing the intention (denoted as REPSEQ). Alternately, we can use a classifier that assigns multiple labels (apps ids) to the input (denoted as MULTLAB). Compared with the MULTLAB strategy, the advantage of REPSEQ is that it can preserve the order of the app sequence. However, once the intention is classified, the representative app sequence will always be the same, regardless of variations in the input. This could be a potential problem for statically based intentions. Arguably, during this process, we could weight each set member by its closeness to the input; we did not investigate this possibility. To evaluate, we compare the set of apps predicted by our realization model with the actual apps launched by the user and compute an F_1 score[2].

There are two types of users—ones for which historical data are available, and the others. New users or users with privacy concerns will not have sufficient data. Thus, a *generic* model trained from large user community can be used instead of *personalized* model. We expect that a sufficiently well-trained generic model can provide reasonable performance; as history is accumulated performance will improve.

The building of intention understanding models may be impacted by intra- and inter-user inconsistency in the language/apps. We may encounter the problem of *vocabulary-mismatch* [13, 22], where interactions related with the same intention have non-overlapping 1) spoken terms (words), even caused by minor differences such as misspellings, morphologies, etc; 2) apps, e.g., people may use different apps — MESSENGER or EMAIL with essentially similar functionality. Below we describe two techniques to overcome potential language- and app-mismatch.

3.1.1 Language Mismatch

We can consider a user's input utterances (e.g., "schedule a meeting") as a query to the intention model. To manage language inconsistency, we used a two-phase process — 1) text normalization where only verbs and nouns in the query are preserved and further lemmatized (e.g., "took" → "take"); 2) query enrichment (QryEn) which expands the query by incorporating words related to it semantically. QryEn can reduce the likelihood of seeing sparse input feature vector du to out-of-vocabulary [24] words. In this work, we used word2vec [17] with gensim[3] toolkit on the pre-trained GoogleNews word2vec[4] model. The proposed QryEn algorithm is described in Al-

[2] $F_1 = 2 \times Precision \times Recall / (Precision + Recall)$

[3] Toolkit: `https://radimrehurek.com/gensim/models/word2vec.html`

[4] Model: `https://drive.google.com/file/d/0B7XkCwpI5KDYNlNUTTlSS21pQmM/edit?usp=sharing`

gorithm 1. In short, each word w_i in the lemmatized query Q yields mass increases for N semantically close words in the feature vector f.

Algorithm 1 Query Enrichment

Require: lemmatized words of the query $Q = \{w_1, ..., w_{|Q|}\}$ and their counts $C = \{c_1, ..., c_{|Q|}\}$; training vocabulary V; bag-of-word feature vector $Q_f = \{f_1, ..., f_{|V|}\}$ constructed on Q; the word semantic relatedness matrix M; the number of semantically similar words N to be extracted for each word in Q;

Ensure: an enriched bag-of-word feature vector

1: **for** each $w_i \in Q$ **do**
2: Use M to find the N closest words $V_N = \{v_1, ..., v_N\} \in V$;
3: **for** each $v_j \in V_N$ **do**
4: $f_j = f_j + M_{i,j} \times c_i$
5: **end for**
6: **end for**
7: **return** f;

3.1.2 App Mismatch

When a generic model is used, recommended apps may not match the apps available on a specific user's device. For example, the recommended app, BROWSER should be converted to CHROME if that is the only (or preferred) app in this user's phone that can browse the Internet. Therefore, similarity metrics among apps are needed.

There are several ways to compute **app similarity** (AppSim). First, based on the edit distance between app (package) names, for example com.lge.music is similar to com.sec.android.app.music since both contains the string "music". Second, we can project an app to a vector space. Ideally, apps with similar functionalities will appear close to each other. Possible resources to use are 1) app descriptions in app stores; 2) language associated with each app when users verbally command the app (see example in Fig 2). Third, app-store category may indicate functionality-wise similarity. However, we found Google Play category too coarse. In this work, we used the first method with 16 fillers (e.g., "android", "com", "htc") removed from package names. Examples are shown in Table 2. We found this simple method significantly improved system performance (described later).

Table 2: Most similar apps for Accuweather and Music among 132 apps in our data collection

order	com.accuweather.android	com.lge.music
1	com.sec.android.widgetapp.ap.hero.accuweather	com.google.android.music
2	com.jrdcom.weather	com.sec.android.app.music
3	com.weather.Weather	com.spotify.music

3.2 Conveying Intention Understanding

IAs may need to communicate with the user in language cast at the level of intention, especially as part of a clarification process. For example, the IA may launch a short sub-dialog by saying "are you trying to *share a picture*?" This involves a template ("are you trying to ___?") and some content ("share a picture"). Instead of echoing content directly extracted from the user's current input, we abstract the semantics of similar previous interactions to provide language material indicating that the agent (though a paraphrase) indeed understands the user's intention.

4 Study

4.1 Intention Interpretation and Realization

To evaluate intention modeling, we focus on three comparisons: 1) **intention**: static vs. dynamic models; 2) **source**: personalized vs. generic setups; 3) **method**: REPSEQ vs. MULTLAB realization strategies. We used the chronologically first 70% of each user's data for training the personalized model, in principle mirroring actual data accumulation. The remaining 13 users' first 70% data was combined to train the generic model. The number of intentions K_C for the static intention model and the number of nearest neighbors K_N for the dynamic model can be varied. We adapted K_C using gap statistics [26], an unsupervised algorithm, to select the optimal K_C from 1 to 10 before KMeans. K_N was set to the square root of the number of training examples [5]. For REPSEQ we used ROVER to collapse multiple app sequences into one [6]. For MULTLAB, we used SVM with linear kernel.

We show system performance in Table 3. This prediction task is difficult since on average each user has 19 unique apps and 25 different sequences of apps in our data collection. The upper part corresponds to static intention model and the lower part to dynamic intention. Within either approach, different intention realization strategies (QryEn and AppSim) and their combination are also shown. We performed a balanced ANOVA test of F_1 score on the factors mentioned above: **intention**, **source** and **method**. The test indicates that the performance differs significantly ($p < 0.05$).

As noted earlier, the static model has the flexibility to incorporate richer information (post-initiate features) when used to discover the basic K_C intentions. As shown in Table 3, adding more post-initiate information (denoted with $\star$ and †) improves personalized models since users have behavioral patterns. However, it does not necessarily improve generic models, mainly due to the inter-user difference in language and apps.

But we do not observe superior performance for the static model over the dynamic one, even when richer information incorporated ($\star$ and †) . For REPSEQ strategy, the dynamic model is much better than the static one. It is possible that REPSEQ is sensitive to the selection of similar interactions. Arguably, an input may fall close to the intention boundary in a static setting, which indeed is closer to some interactions on the other side of the boundary as opposed to the ones within the same

Table 3: Weighted average F_1 score (%) on test set across 14 participants, using **bag-of-words**. Average K_C in static condition is 7.0 ± 1.0 for generic model, and 7.1 ± 1.6 for personalized model. The static condition was run 10 times and the average is reported. K_N in the dynamic condition is 18.5 ± 0.4 for the generic model and 4.9 ± 1.4 for the personalized model. $\star$ indicates both descriptions and user utterances are used in clustering and † indicates apps are used as well.

	REQSEQ		MULTLAB	
	Personalized	Generic	Personalized	Generic
Static (baseline)	42.8	10.1	55.7	23.8
+QryEn	44.6	11.2	56.3	27.9
+AppSim	42.8	15.1	55.7	27.8
+QryEn+AppSim	44.6	16.1	56.3	36.1
+QryEn+AppSim$\star$	44.9	18.0	57.5	**37.1**
+QryEn+AppSim†	**45.8**	**18.1**	**57.6**	35.9
Dynamic (baseline)	50.8	23.8	51.3	19.1
+QryEn	54.9	26.2	57.0	22.9
+AppSim	50.8	30.1	51.3	22.7
+QryEn+AppSim	**54.9**	**32.5**	**57.0**	**28.0**

intention cluster. On the other hand, the MULTLAB approach shows relatively consistent performance in both static and dynamic settings, indicating robustness and self-adaptability with respect to the choice of interactions of similar intention.

In Table 3, the fact that QryEn improves the F_1 score in all conditions indicates that semantic similarity among words can effectively address the *language-mismatch* problem. On the other hand, although AppSim has no effect on the personalized model, it addresses the *app-mismatch* issue in generic models intuitively ($p < 0.05$ when comparing with the baseline in an balanced ANOVA on additional two factors: **intention**, **method**). Combining QryEn and AppSim methods together (denoted as "+QryEn+AppSim") consistently achieves the highest F_1 score. As we expected, generic intention model is consistently inferior to the personalized model.

4.2 Intention Representation in Natural Language

It should be possible to automatically abstract the semantics of the recognized intention cluster (or neighbors): Text summarization may be used to generate high-level description of the intention cluster [7, 12]. Keyphrase extraction provides another alternative [27, 15, 1]. Note that, even if the automatic generation of semantic summarization is not precise, it may still be sufficiently meaningful in context.

In this study, we used the Rapid Automatic Keyword Extraction (RAKE[5]) algorithm [1], an unsupervised, language-independent and domain-independent extraction method. This method has been reported to outperform other unsupervised

[5] Toolkit: `https://www.airpair.com/nlp/keyword-extraction-tutorial`

Table 4: Mean number of phrases generated using different resources

MANUAL	ASR	DESC	DESC+ ASR	DESC+ MANUAL
20.0	20.3	11.3	**29.6**	29.1

methods such as TextRank [16] and [9] in both precision and F score. In RAKE, we required that 1) each word have 3 or more characters; 2) each phrase have at most 3 words; and that 3) each key word appear in the text at least once. We did not investigate tuning these parameters. We use 3 individual resources and 2 combinations, reflecting constraints on the availability of different contexts in real-life. The three individual resources are manual transcription of user utterances in their dialogs (MANUAL) and their ASR transcriptions (ASR) and high-level task descriptions (DESC). The average number of key phrases generated by each resource (or their combination) is shown in Table 4.

We selected 6 users to first review their own clusters, by showing them all cluster members with 1) apps used in the member interaction; 2) dialog reproduced; 3) meta-data such as time, date, address, etc. We let them judge whether each individual phrase generated by the system summarized all the activities in the cluster (binary judgement). We used three Information Retrieval (IR) metrics to evaluate performance among different resources — 1) Precision at position K ($P@K$); 2) Mean Average Precision[6] at position K ($MAP@K$); 3) Mean Reciprocal Rank (MRR). The first two metrics emphasize on the quality of the top K phrases, MRR focuses on a practical goal — "how deep the user has to go down a ranked list to find one useful phrase?". Average MRR is 0.64, meaning that the user will find an acceptable descriptive phrase in the top 2 items shown; an ANOVA did not show significant differences between resources. With more sensitive $MAP@K$ and $P@K$ metrics, DESC+ASR and DESC+MANUAL do best. The improvement becomes significant as K increases: having a user-generated task description is very useful.

Participants also were asked to suggest carrier phrases that the agent could use to refer to activities; we found these to be unremarkable. Among the 23 phrases collected, "do you want to ___" and "would you like to ___" were the most popular.

To conclude, if the IA can observe a user's speech commands or elicit descriptions from the user (ideally both), it can generate understandable activity references and might avoid less efficient interactions (e.g. lists).

5 Conclusion and Future Work

We present a framework, HELPR, that is used to learn to understand a user's intention from a high-level description of goals (e.g., "go out with friends") and to link these to specific functionality available on a smart device. The proposed agent solicits descriptions from the user. We found that the language used to describe activities is sufficient to group together similar activities. Query enrichment and app similarity help with language- and domain-mismatch problems, especially when a

[6] $MAP@K$ computed as: $\sum_{k=1}^{K} precision(k) * relevance(k)/K$

generic model is used. We demonstrated that an agent could use data from large user community while also learning user-specific models.

The long-term goal of our work is to create agents that observe recurring human activities, understand the underlying intentions and support the task through spoken language interaction. The agent must communicate on the level of intentions instead of, or in addition to, individual apps. And it needs to manage the context of the activity so that its state can be shared between different apps.

The value of such an agent is that it would operate on a level higher than provided by app-specific interfaces. It would moreover allow the user to effectively build their own applications by composing the functionality in existing apps. We have shown that it is possible to infer user intentions; the next challenge is to capture meaningful context and actively apply it across different apps.

6 Acknowledgement

This work was supported in part by YAHOO! InMind, and by the General Motors Advanced Technical Center. We thank Zhenhao Hua for implementing the logger application, and Yulian Tamres-Rudnicky and Arnab Dash for collecting data.

References

1. Michael W. Berry and Jacob Kogan. 2010. Text mining: applications and theory.
2. Dan Bohus and Alexander I Rudnicky. 2005. Sorry, I didn't catch that!-An investigation of non-understanding errors and recovery strategies. In *SIGdial Workshop on Discourse and Dialogue (SIGDIAL)*.
3. Yun-Nung Chen and Alexander I. Rudnicky. 2014. Dynamically supporting unexplored domains in conversational interactions by enriching semantics with neural word embeddings. In *Proceedings of 2014 IEEE Spoken Language Technology Workshop (SLT)*. IEEE, 590–595.
4. Yun-Nung Chen, Ming Sun, and Alexander I. Rudnicky. 2015. Leveraging Behavioral Patterns of Mobile Applications for Personalized Spoken Language Understanding. In *Proceedings of 2015 International Conference on Multimodal Interaction (ICMI)*.
5. Richard Duda, Peter Hart, and David Stork. 2012. *Pattern Classification*. John Wiley and Sons.
6. Jonathan G Fiscus. 1997. A Post-Processing System to Yield Reduced Word Error Rates: Recognizer output voting error reduction (ROVER). In *Proceedings of Automatic Speech Recognition and Understanding Workshop (ASRU)*. 347–352.
7. Kavita Ganesan, Chengxiang Zhai, and Jiawei Han. 2010. Opinosis: a graph-based approach to abstractive summarization of highly redundant opinions. In *Proceedings of the 23rd international conference on computational linguistics (COLING)*. ACL, 340–348.
8. Chris Harrison, Robert Xiao, Julia Schwarz, and Scott E. Hudson. 2014. TouchTools: leveraging familiarity and skill with physical tools to augment touch interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2913–2916.
9. Anette Hulth. 2003. Improved automatic keyword extraction given more linguistic knowledge. In *Proceedings of the 2003 conference on Empirical methods in natural language processing (EMNLP)*. ACL, 216–223.
10. Qi Li, Gokhan Tur, Dilek Hakkani-Tur, Xiang Li, Tim Paek, Asela Gunawardana, and Chris Quirk. 2014. Distributed open-domain conversational understanding framework with domain

independent extractors. In *Spoken Language Technology Workshop (SLT), 2014 IEEE*. IEEE, 566–571.

11. Bor-shen Lin, Hsin-min Wang, and Lin-shan Lee. 1999. A distributed architecture for cooperative spoken dialogue agents with coherent dialogue state and history. In *Proceedings of 1999 IEEE Workshop on Automatic Speech Recognition and Understanding Workshop (ASRU)*, Vol. 99. 4.
12. Fei Liu, Jeffrey Flanigan, Sam Thomson, Norman Sadeh, and Noah A. Smith. 2015. Toward Abstractive Summarization Using Semantic Representations. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.
13. Claudio Lucchese, Salvatore Orlando, Raffaele Perego, Fabrizio Silvestri, and Gabriele Tolomei. 2011. Identifying task-based sessions in search engine query logs. In *Proceedings of the fourth ACM international conference on Web search and data mining*. ACM, 277–286.
14. Jean-Michel Lunati and Alexander I. Rudnicky. 1991. Spoken language interfaces: The OM system. *CHI91 Human Factors on Computing Systems* (1991).
15. Olena Medelyan. 2009. Human-competitive automatic topic indexing. In *Thesis Dissertation, University of Waikato*.
16. Rada Mihalcea and Paul Tarau. 2004. TextRank: Bringing order into texts. In *ACL*.
17. Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *Proceedings of Workshop at International Conference on Learning Representations (ICLR)*.
18. Mikio Nakano, Shun Sato, Kazunori Komatani, Kyoko Matsuyama, Kotaro Funakoshi, and Hiroshi G Okuno. 2011. A two-stage domain selection framework for extensible multi-domain spoken dialogue systems. In *SIGdial Workshop on Discourse and Dialogue (SIGDIAL)*. Association for Computational Linguistics, 18–29.
19. Alexander I Rudnicky, Jean-Michel Lunati, and Alexander M Franz. 1991. Spoken language recognition in an office management domain. In *Proceedings of International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*. IEEE, 829–832.
20. Seonghan Ryu, Donghyeon Lee, Injae Lee, Sangdo Han, Gary Geunbae Lee, Myungjae Kim, and Kyungduk Kim. 2012. A Hierarchical Domain Model-Based Multi-Domain Selection Framework for Multi-Domain Dialog Systems. In *Proceedings of the 24th International Conference on Computational Linguistics(ACL)*.
21. Seonghan Ryu, Jaiyoun Song, Sangjun Koo, Soonchoul Kwon, and Gary Geunbae Lee. 2015. Detecting Multiple Domains from Users Utterance in Spoken Dialog System. In *Proceedings of the International Workshop on Spoken Dialogue Systems (IWSDS)*.
22. Xuehua Shen, Bin Tan, and ChengXiang Zhai. 2005. Implicit user modeling for personalized search. In *Proceedings of the 14th ACM international conference on Information and knowledge management*. ACM, 824–831.
23. Pontus Stenetorp, Sampo Pyysalo, Goran Topić, Tomoko Ohta, Sophia Ananiadou, and Jun'ichi Tsujii. 2012. BRAT: a web-based tool for NLP-assisted text annotation. In *Proceedings of the Demonstrations at the 13th Conference of the European Chapter of the Association for Computational Linguistics (EACL)*. Association for Computational Linguistics, 102–107.
24. Ming Sun, Yun-Nung Chen, and Alexander I. Rudnicky. 2015a. Learning OOV through Semantic Relatedness in Spoken Dialog Systems. In *16th Annual Conference of the International Speech Communication Association (Interspeech)*.
25. Ming Sun, Yun-Nung Chen, and Alexander. I. Rudnicky. 2015b. Understanding User's Cross-Domain Intentions in Spoken Dialog Systems. In *NIPS workshop on Machine Learning for SLU and Interaction*.
26. Robert Tibshirani, Guenther Walther, and Trevor Hastie. 2001. Estimating the number of clusters in a data set via the gap statistic. In *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*. 411–423.
27. Ian H. Witten, Gordon W. Paynter, Eibe Frank, Carl Gutwin, and Craig G. Nevill-Manning. 1999. KEA: Practical automatic keyphrase extraction. In *Proceedings of the fourth ACM conference on Digital libraries*. 254–255.