Toward Habitable Assistance from Spoken Dialogue Systems

Susan L. Epstein[1,2], Rebecca J. Passonneau[3], Tiziana Ligorio[1], and Joshua Gordon[3]

[1]Hunter College and [2]The Graduate Center of The City University of New York, New York, NY, USA

[3]Columbia University, New York, NY, USA

susan.epstein@hunter.cuny.edu, becky@cs.columbia.edu, tligorio@hunter.cuny.edu, joshua@cs.columbia.edu

Abstract

Spoken dialogue is increasingly central to systems that assist people. As the tasks that people and machines speak about together become more complex, however, users' dissatisfaction with those systems is an important concern. This paper presents a novel approach to learning for spoken dialogue systems. It describes *embedded wizardry*, a methodology for learning from skilled people, and applies it to a library whose patrons order books by telephone. To address the challenges inherent in this application, we introduce RFW+, a domain-independent, feature-selection method that considers feature categories. Models learned with RFW+ on embedded-wizard data improve the performance of a traditional spoken dialogue system.

Introduction

Across a broad range of real-world applications, people increasingly use spoken dialogue systems (*SDSs*). Fielded SDSs typically assume *system initiative*, that is, they control the path of the dialogue, whose turn it is to speak, and even indicate what the person (*user*) can say in response (e.g., "Say 1 for hours, 2 for sales…"). As the tasks SDSs address become more difficult, however, users' language can become more complex, and it is far more challenging for the system designer to predict and control the dialogue. Thus, an SDS's ability to understand its users' goals becomes central to its development. Our results demonstrate that such understanding relies on dynamic synergy among a variety of data sources available to the system at runtime.

This paper reports on a large-scale project that learns to support habitable human-machine dialogue through the implementation of three systems on which we collected extensive data. The next section describes a challenging application and *CheckItOut*, an SDS constructed for it within a traditional architecture. To support research into dialogue management strategies for good SDS performance across different levels of speech recognition, CheckItOut's speech recognition is deliberately poor. It often asks users to repeat, and terminates 20% of their calls.

People, however, adeptly match automated transcription of similar speech to CheckItOut's databases. We therefore used *embedded wizardry* to develop a second system, *CheckItOutW,* which collects data both about CheckItOut's computations and about how people make decisions when restricted to CheckItOut's input and actions. The complexity of the resultant corpus drove the development of *RFW+,* a feature selection method that supports machine learning with knowledge about categories of data available to a system at runtime. A new SDS, *CheckItOut+*, used models learned under RFW+ on wizard data to reduce and address errors. We describe three extensive experiments, one for each of CheckItOut, CheckItOutW, and CheckItOut+. CheckItOut+ demonstrates how multiple knowledge sources already available to an SDS can support a system's ability to understand, and thereby improve its accuracy. It also provides insight on SDS performance evaluation.

The Application: Library Book Orders

Our application domain is book orders at The Andrew Heiskell Braille and Talking Book Library, a branch of the New York Public Library and a Regional Library of the National Library Service. Heiskell patrons typically order several books during a telephone call with a librarian. Call volume from 5000 active patrons is heavy, and is supplemented by a voice mail service. Our copy of the patron database is sanitized to protect patrons' privacy.

Traditional SDSs use a natural language understanding (*NLU*) *pipeline* to process user utterances. When an SDS receives acoustic data, its automated speech recognition (*ASR*) module produces a text string of words. ASR is challenged by a large vocabulary, many-word utterances, a broad variety of human speakers, and noisy environments and data channels. Our application confronts all of these.

Our copy of Heiskell's book database has 71,166 titles and 28,031 authors, in a vocabulary of 54,448 distinct words. Book titles are less predictable than ordinary speech, and longer (2 – 34 words) than user utterances to most commercial SDSs. In every data collection, users (several with native languages other than English) called a dedicated VOIP line from mobile or landline telephones at locations they chose. Although we adapted ASR acoustic models of Wall Street Journal dictation speech with 8 hours of conversational speech for book orders, the ASR word error rate (*WER*) averaged 50%. As a result, the ASR these systems confront is as difficult to match to the database as the ASR for titles shown in Figure 1.

An NLU pipeline sends ASR output to a semantic parser that detects referenced concepts (e.g., a title or an author in a book request) and binds them to values. The parse describes concepts as attribute slots and their values, where a slot name corresponds roughly to a database attribute. For example, a BookRequest parse could include one or more title, author, or catalog-number slots. A confidence annotator then scores the system's certainty about user input.

The NLU pipeline delivers its best parse to a *dialogue manager* that decides what to do (e.g., query a database) or say next. A query seeks to match a concept attribute value to an object in the database. For robustness, we rank returns to a query by *R/*O (Ratcliff and Metzener, 1988), a simple string-similarity metric. If the top return has a high enough R/O score, it is offered to the user. For example:

SDS: Next book?
User: BOOK NUMBER SIX SEVEN FOUR THREE TWO
SDS: *A Work in Progress* is available. Next book?

For multiple perfect matches (e.g., an author with several titles), the system offers the books one at a time to the user.

Even with perfectly-recognized speech, however, the dialogue manager may confront an ambiguous or unknown situation. For example, a user may request a title that is not unique or not in the database. A *non-understanding* occurs when the SDS cannot produce a semantic interpretation of what it has just heard. A *misunderstanding* occurs when the SDS binds a concept incorrectly (e.g., identifies the wrong book). Typically the SDS becomes aware of a misunderstanding only when the user signals it. *Error handling* seeks to correct misunderstandings and to address non-understandings as they arise.

CheckItOut is an SDS for the Heiskell domain, built within the Olympus/Ravenclaw architecture (Bohus and Rudnicky, 2009). CheckItOut uses the PocketSphinx speech recognizer (Huggins-Daines et al., 2008), the Phoenix parser (Ward and Issar, 1994), and the Helios confidence annotator (Bohus and Rudnicky, 2002). Once the pipeline arrives at a single parse for the user's most recent utterance, the RavenClaw dialogue manager applies a domain-specific hierarchy that explicitly specifies domain-dependent goals (e.g., identify the book) and provides the ability to query databases.

ASR	True title
INTO THAN 9	*Into the Night*
HELEN AND TEACH DISTORT TELL UNTIL AN AM SULLIVAN MACY	*Helen and Teacher: The Story of Helen Keller and Anne Sullivan Macy*
NAH DON'T BONES	*Map of Bones*
ELUSIVE TOTAL MAN	*I Lived to Tell it All*

Figure 1: ASR encountered by CheckItOut for book titles.

For error handling and recovery, RavenClaw separates domain-specific dialogue strategies (e.g., ask for the next book) from domain-independent ones (e.g., ask the user to repeat). Once the dialogue manager decides what to say, a template-based natural language generator converts the semantic representation to an orthographic string, which is then sent to the Kalliope/Swift text-to-speech generator. That speech is forwarded to the audio manager, which conveys it back to the user.

We emphasize that this is a difficult task. CheckItOut must determine whether a request is for a title, an author, or a catalog number. Indeed, our users were given a pre-specified signal for misunderstanding:

User: MY PERSONAL BEST
SDS: *Persuasion* is available. Next book?
User: THAT'S NOT WHAT I SAID
SDS: Sorry, I must have misunderstood you, let's try again. Next book?

In earlier studies, we had shown that people without library training were surprisingly adept at matching noisy ASR to the correct Heiskell book title at WERs as high as 83% (Ligorio et al., 2010; Passonneau et al., 2009a; Passonneau et al., 2009b; Passonneau et al., 2010). Thus we chose to learn from people how an SDS should do it.

Embedded Wizardry

A Wizard of Oz (*WOz*) study inserts a person (the *wizard*) as the decision maker in a system whose human users believe they are interacting with a computer. WOz studies were originally designed to anticipate the kinds of experiences a system could expect to encounter. Our WOz study, however, embeds a wizard within a functioning system, to capture data available to the system when the wizard made a decision. The assumption was that a proficient human wizard could be modeled to improve CheckItOut.

Throughout our work, subjects (both users and wizards) were balanced for gender, and recruited from among students at local colleges. Users called over several days, and were periodically asked to complete an electronic survey that described their experience on a Likert scale of 1 to 5. For each call, the user retrieved from a website a new, randomly-generated scenario: user identification data, and

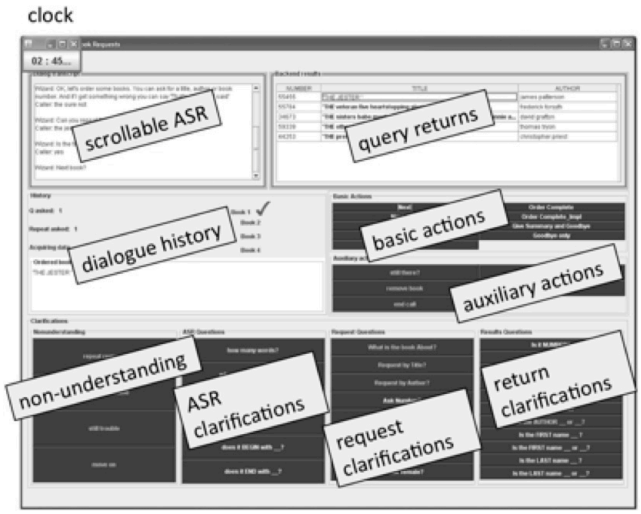

Figure 2: The wizard GUI for book requests in CheckItOutW.

four books, each with its title, author, and catalog number. Users were asked to order one book by title, one by author, one by catalog number, and one by any of those methods. A new, randomly-selected set of 3000 books provided each experiment with data for the scenarios, the ASR language model, and the semantic grammar rules. On each call, the SDS was expected to greet the user, identify her, accept four book requests, offer an optional order summary, and sign off. The focus of this paper is on book requests, the most difficult part. Baseline data for CheckItOut was collected in an experiment where each of 10 users was asked to make 50 calls; 562 calls were collected.

CheckItOutW's interface for a human wizard ablated both the wizard's input and her output as she made her own NLU and DM decisions. (CheckItOut's recognition and parse output was generated but withheld from the wizard.) Instead of speech, the wizard saw ASR output on a graphical user interface (*GUI*), labeled for content in Figure 2. The wizard could query the Heiskell databases through the GUI, which displayed the author, title, and catalog number of the best matches in descending order of their (concealed) R/O score. The wizard's behavior was restricted to a rich set of 43 actions (with black backgrounds in Figure 2). Three actions queried the database, by author, title, or catalog number. Others offered a book to the user, moved on to the next book, offered an order summary, or signed off. Many actions were questions to be generated by the system and then spoken to the user. Questions sought to clarify what the user wanted (e.g., "Did you ask for a title?" or "Did you say — ?").The wizard used mouse clicks to conduct a dialogue to help the user order books.

Nine volunteers trained as wizards; we selected six who appeared the most adept and motivated. Each of 10 subjects then made 90 scheduled calls to CheckItOutW (unknowingly, at least 15 to each wizard). We explained to CheckItOutW users that the system was highly experimental, and likely to be slow because it was considering many alternatives. To preserve our subjects' positive attitudes we also had the clock on the GUI change color after six minutes. That alerted the wizard to finish the current book request and sign off. During these calls, we collected data on 163 features that described knowledge that would be available to support dialogue management at runtime.

The 913 collected calls to CheckItOutW covered 3394 book requests and 20,378 utterances. These were assembled into 17,288 adjacency pairs. An *adjacency pair* is the speech that begins with a system prompt and ends just before the next one. About a third of all adjacency pairs included at least one query. Query returns included the intended book 58% of the time if it was for a catalog number, but only 33% for author queries, and 28% for title queries. Nonetheless, our wizards successfully identified 92% of the requests. (Only 22 books were ever countermanded by THAT'S NOT WHAT I SAID.) The 6-minute time limit, however, forced termination of 63% of the calls before all 4 books were ordered. This was clearly a model of expertise worth learning.

Learning a New Dialogue Manager

Except for its dialogue manager, CheckItOut+ is identical to CheckItOut. When it has a single confident parse or knows it has made an error, CheckItOut+ behaves the way CheckItOut would. Otherwise, CheckItOut+'s *wizard module* selects an action. The intent is to replace domain-independent strategies with ones learned from the wizards, to make CheckItOut+ more flexible in the face of uncertainty. We partitioned the CheckItOutW data to address three important decision points, each with its own model. Without a single confident parse, the *voice search model* decides whether or not to seek a title, an author, and a catalog number that partially matches the ASR (i.e., do *voice search* instead of semantic parsing). In these CheckItOut+ examples, periods delimit unconfident words. Here, no parse was generated, but voice search successfully matched the correct book:

User: .DUDLEY. SAFARI
SDS: Is the title *Deadly safari*?
User: YES

After voice search, the *yes-query model* decides whether to confirm the best return with the user, as it did above, or to ask a question informed by that return as it did here:

User: .FOR. .NONDRIVERS. .SCHULLER. CHAPMAN
SDS: Is the author Fern Schumer Chapman?
User: YES

Finally, if voice search was not performed, the *no-query model* decides whether to indicate non-understanding or to ask about the ASR or the request. Note the switch here from a prompt for the author to a prompt for the title:

Table 1: Feature categories and examples under varying granularity for 163 features collected in the wizard experiment.

Coarse (3)	Medium (5)	Fine (10)	Examples
Adjacency pair (36)	Adjacency pair (36)	Adjacency pair history (23)	# books ordered so far
		Adjacency pair (13)	# database searches in this adjacency pair
Input (99)	Recognition (45)	Recognition history (10)	Average language model score on call
		Recognition (35)	Normalized acoustic model score
	Parse (46)	Parse history (12)	Average # grammar slots on this book
		Parse (34)	# words not covered by the best parse
Query (28)	Query (16)	Query history (8)	# title queries in this call
		Query (8)	# query results
	Confidence (20)	Confidence history (14)	Average Helios confidence on this book
		Confidence (6)	Helios confidence

SDS: Sorry, I misunderstood. Can you repeat the author?
User: .ANDRES. MORTIMER
SDS: Did you say John Mortimer?
User: NO
SDS: Sorry, I misunderstood. Can you repeat the author?
User: .ANDREW. .VACHSS. NO DIDN'T
SDS: Can you give me the book title please?

The no-query model moves on to the next book after three consecutive non-understandings on the current one.

The CheckItOutW dataset covers the system's runtime knowledge thoroughly but redundantly. On such data, machine-learning algorithms often benefit from feature selection. Feature selection methods are either *filters* that select relevant features before learning (independently from the intended learning algorithm), or they are *wrappers* that iteratively test feature subsets on a specific learning algorithm. Two new wrappers were developed for this work: RFW and RFW+ (Ligorio, 2011).

RFW (Randomized Feature Weighting) explores the possible feature subsets to learn a feature-relevance criterion tailored to both the data and the learner. Unlike most wrappers, however, RFW ranks features well at low computational cost, in part because a parameter *x* determines the size of the feature subset RFW considers. RFW consistently performs at least as well as state-of-the art filters (e.g., CFS(Hall, 2000)) and wrappers (e.g., Greedy Forward Selection (Kohavi and John, 1997)). Although slower than a filter, on a variety of datasets RFW has proved orders of magnitude faster than Greedy Forward Selection for logistic regression and decision trees, and faster for support vector machines (Ligorio, 2011).

To ensure that every aspect of available knowledge contributed to decisions, we developed RFW+, which augments RFW-selected features to address under-represented feature categories. Table 1 categorizes the 163 features in the dataset at three levels of granularity. When applied to such categories, RFW+ seeks to improve RFW's initial subset by the inclusion of some features from each category. Under medium granularity, therefore, RFW+ seeks to include every pipeline stage. Figure 3 compares RFW and RFW+ to baselines as logistic regression learns whether or not to perform voice search on data from all wizards.

One way to infuse learning with knowledge is to select relevant features, as RFW and RFW+ do; another is to select the best examples of the target behavior available. Two of our six wizards, WA and WB, were particularly skilled. They had more *fully successful calls*, where they identified all four books correctly (33% and 31% compared to 28% for all wizards), and had the fewest *failed calls*, where no books were ordered (7% and 12% compared to 17% for all wizards). WA and WB also had different conversational styles. WA was persistent; she asked the most questions per book request, made the most database queries, and shifted query type when necessary (e.g., after some diffi-

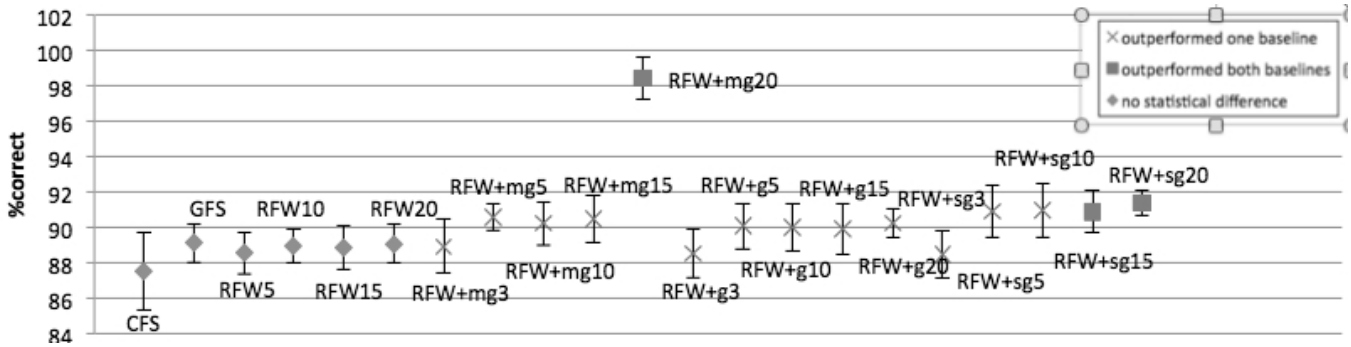

Figure 3: Percentage of instances correctly classified by voice query models to match an ASR hypothesis learned with RFWx and RFW+ *x* for all wizard data, where *x* is the size of the feature subset. Granularity for RFW+ is coarse (mg), medium (g), or fine (sg). CFS and Greedy Forward Selection (GFS) are baselines.

culty with the title, she clicked on "Who is the author?"). In contrast, WB queried the database extensively and "spoke" the least of any wizard. He quickly moved on to the next book when he experienced difficulty; the user could return to it later if time remained. For these reasons, we learned voice search and yes-query models on the data for WA from her 3161 and 1159 instances, respectively, and the no-query model on the 714 instances for WB.

Under 10-fold cross validation on these reduced datasets, speed and accuracy were roughly equivalent for learners that used decision trees, support vector machines, or logistic regression. Granularity impacted RFW+ models' performance. (This differs from performance across all wizards, shown in Figure 3.) Thus the wizard module implemented in the dialogue manager for CheckItOut+ was learned by RFW+ for logistic regression with medium granularity for the voice-search and no-query models, but with fine granularity for the yes-query model. Table 2 summarizes the features in the learned wizard module.

Performance with CheckItOut+

To evaluate the system under the new dialogue manager, 10 subjects each made 50 calls to CheckItOut+ under the same conditions as those used for CheckItOut. During these 505 calls, the wizard module triggered 4.72 times per call, and was responsible for at least 38% of the books correctly ordered by CheckItOut+. Calls averaged about 13 seconds longer with CheckItOut+, an increase of only 6.19%. (All comparisons here are at $p < 0.05$.)

CheckItOut+ ordered more books, and more correct books. CheckItOut+ also ordered fewer incorrect books and elicited fewer misunderstanding signals (THAT'S NOT WHAT I SAID). CheckItOut+ averaged 65.57 seconds per correct book, 25% faster than CheckItOut's 87.89 seconds. When calls required fewer turns (an indication that the user was readily understood), CheckItOut+ identified more books correctly, and costs (e.g., number of system and user turns per ordered book) decreased. When calls had more turns, CheckItOut+ persisted, while CheckItOut hung up.

Despite CheckItOut+'s greater task success and higher throughput, users' survey responses showed no preference for one SDS over the other. Only 1 of the 11 questions about how users perceived the system registered a difference: "I had to pay close attention while using the system" elicited different responses. CheckItOut+ users indicated that they had had to work harder; its persistent questions may have revealed too much about the system's uncertainty, unlike the binary responses (understood or not) of commercial SDSs. Demands on users' attention apparently overshadowed CheckItOut+'s increased task success, and influenced their level of satisfaction.

Related Work

Common paradigms for dialogue managers learn a policy from a finite-state model (e.g., (Singh et al., 2002), follow a plan (Allen et al., 1995), or use frames (e.g., (Seneff et al., 1998) or information-state updates (e.g.,(Traum and Larsson, 2003)). Other WOz studies have investigated multimodal clarification (Rieser and O., 2011), domain-dependent strategies (Skantze, 2005), and tried to predict responses to non-understanding (Bohus, 2007). In contrast, our wizards used voice search to understand noisy ASR.

CheckItOut+ scores well on dialogue efficiency, but other work suggests that users may prefer systems that offer explicit confirmation, rather than implicit confirmation or none at all, both of which were available through our wizard module (Litman et al., 1999). PARADISE correlates user satisfaction (averaged here across all survey questions) with 20 cost and success measures, and performs linear regression on the most relevant (Walker et al., 2000). A PARADISE-style evaluation showed that users of both CheckItOut and CheckItOut+ were sensitive to task success. CheckItOut user satisfaction was negatively correlated with the number of prompts that dictated what they could say, while CheckItOut+ user satisfaction was negatively correlated with the number of incorrect books.

Table 2: Features in learned models with categories from Table 1, denoted as A (adjacency pair), R (recognition), P (parse), Q (query), and C (confidence). Italics indicate features used in more than one model.

Voice-search	Yes-query	No-query
# of pairs in request (A)	# times moved on in call (A)	# user utterances in pair (A)
# question prompts in call (A)	# new book requests in pair (A)	# explicit confirms in request (A)
Last prompt was non-understanding (A)	Mean acoustic model score in call (R)	*Last prompt was explicit confirmation* (A)
Last prompt was explicit confirmation (A)	Mean R/O score on voice search returns (Q)	*Mean word confidence in top* (R)
Mean word confidence in top (R)	σ of R/O score on voice search returns (Q)	Maximum word confidence in top (R)
# words in best parse for top (P)		Acoustic model score for top (R)
Top grammar slot for top (P)		≥ 1 title slot in parse (P)
# parses for top (P)		Some words omitted by top (P)
# database queries in request (Q)		# author queries this request (Q)
# title queries in request (Q)		*# title queries in request* (Q)
Mean Helios/RO in request (C)		Helios confidence in top (C)

Discussion and Current Work

Although most people are no longer surprised to find their telephone call answered by a machine, they are rarely pleased — this work moves toward amelioration of that response. Much of this work is application-independent, including SDS actions and most features, RFW, and RFW+. The best of our wizards' problem-solving strategies (e.g., recognize when no search return is likely) also appear to be application-independent.

We emphasize that the performance improvement chronicled here can only be attributed to changes in the decision manager learned from the wizard data; CheckItOut and CheckItOut+ are otherwise identical. The features learned for CheckItOut+'s wizard module were deliberately drawn by RFW+ from different modules in the pipeline. They improved both the system's accuracy and its speed, which suggests that understanding benefits from less rigid interaction among SDS pipeline modules. In separate work, we have begun development of a new SDS architecture that further exploits the rich wizard corpus we collected.

There is an inherent tension among accuracy, speed, and flexibility for an SDS. Developers rely on system initiative to prevent errors and move the dialogue forward. A user with latitude may provide a familiar property (e.g., a book's author) that is not a unique identifier, and therefore requires further discussion. Our emerging system can use a database of domain-specific objects as a source of confirmation and information to supplement what it has heard and what it has computed, and then develop reasonable interpretations for it to assist the user.

Acknowledgements

The National Science Foundation supported this work under awards IIS-084966, IIS-0745369, and IIS-0744904.

References

Allen, J. F., L. K. Schubert, G. Ferguson, P. Heeman, C. H. Hwang, T. Kato, M. Light, N. G. Martin, B. W. Miller and M. Poesio 1995. The TRAINS Project: A Case Study in Defining a Conversational Planning Agent. Journal of Experimental and Theoretical AI 7: 7-48.

Bohus, D. 2007. Error Awareness and Recovery in Task-Oriented Spoken Dialogue Systems. Computer Science. Pittsburgh, PA, Carnegie Mellon University. Ph.D.

Bohus, D. and A. Rudnicky 2002. Integrating Multiple Knowledge Sources for Utterance-Level Confidence Annotation in the CMU Communicator Spoken Dialogue System, Carnegie Mellon University.

Bohus, D. and A. I. Rudnicky 2009. The RavenClaw dialog management framework: Architecture and systems. Computer Speech and Language 23(3): 332-361.

Hall, M. A. 2000. Correlation-based Feature Selection for Discrete and Numeric Class Machine Learning. 17th International Conference on Machine Learning: 359-366.

Huggins-Daines, D., M. Kumar, A. Chan, A. W. Black, M. Ravishankar and A. Rudnicky 2008. Pocketsphinx: A Free, Real-Time Continuous Speech Recognition System for Hand-Held Device. International Conference on Acoustics, Speech and Signal Processing (ICASSP): 185-189.

Kohavi, R. and G. H. John 1997. Wrappers for feature subset selection. Artificial Intelligence 97(1-2): 273-324.

Ligorio, T. 2011. Feature Selection for Error Detection and Recovery in Spoken Dialogue Systems. Computer Science. The Graduate Center of The City University of New York. Ph.D.

Ligorio, T., S. L. Epstein, R. J. Passonneau and J. B. Gordon 2010. What You Did and Didn't Mean: Noise, Context, and Human Skill. Cognitive Science - 2010.

Litman, D. J., M. A. Walker and M. S. Kearns 1999. Automatic detection of poor speech recognition at the dialogue level. I. In Proceedings of Thirty Seventh Annual Meeting of the Association for Computational Linguistics (ACL), 309-316.

Passonneau, R. J., S. L. Epstein and J. B. Gordon 2009a. Help Me Understand You: Addressing the Speech Recognition Bottleneck. AAAI Spring Symposium on Agents that Learn from Human Teachers. Palo Alto, CA, AAAI.

Passonneau, R. J., S. L. Epstein, J. B. Gordon and T. Ligorio 2009b. Seeing What You Said: How Wizards Use Voice Search Results. IJCAI-09 Workshop on Knowledge and Reasoning in Practical Dialogue Systems. Pasadena, CA, AAAI Press.

Passonneau, R. J., S. L. Epstein, T. Ligorio, J. Gordon and P. Bhutada 2010. Learning About Voice Search for Spoken Dialogue Systems. NAACL HLT 2010: 840-848.

Ratcliff, J. W. and D. Metzener 1988. Pattern Matching: The Gestalt Approach, Dr. Dobb's Journal.

Rieser, V. and L. O. 2011. Learning and evaluation of dialogue strategies for new applications: Empirical methods for optimization from small data sets. Computational Linguistics 37: 153–96.

Seneff, S., E. Hurley, R. Lau, C. Pao, P. Schmid and V. Zue 1998. Galaxy II: A reference architecture for conversational system development. 5th International Conference on Spoken Language Systems (ICSLP-98). Sydney, Australia.

Singh, S., D. J. Litman, M. Kearns and M. Walker 2002. Optimizing dialogue management with reinforcement learning: Experiments with the njfun system. Journal of Artificial Intelligence Research.

Skantze, G. 2005. Exploring Human Error Recovery Strategies: Implications for Spoken Dialogue Systems Speech Communication, Special Issue on Speech Annotation and Corpus Tools 45(3): 207-359.

Traum, D. and S. Larsson 2003. The information state approach to dialogue management. Advances in Discourse and Dialogue. Kuppevelt, J. v. and R. Smith. Amsterdam, Kluwer.

Walker, M. A., C. A. Kamm and D. Litman, J. 2000. Towards developing general models of usability with PARADISE. Natural Language Engineering: Special Issue on Best Practice in Spoken Dialogue Systems 6(3-4): 363-377.

Ward, W. and S. Issar 1994. Recent improvements in the CMU spoken language understanding system. ARPA Human Language Technology Workshop. Plainsboro, NJ: 213-216.