Bootstrapping Spoken Dialog Systems with Data Reuse

Giuseppe Di Fabbrizio Gokhan Tur Dilek Hakkani-Tür

AT&T Labs-Research,
Florham Park, NJ, 07932
{pino,gtur,dtur}@research.att.com

Abstract

Building natural language spoken dialog systems requires large amounts of human transcribed and labeled speech utterances to reach useful operational service performances. Furthermore, the design of such complex systems consists of several manual steps. The User Experience (UE) expert analyzes and defines by hand the system core functionalities: the system semantic scope (call-types) and the dialog manager strategy which will drive the human-machine interaction. This approach is extensive and error prone since it involves several non-trivial design decisions that can only be evaluated after the actual system deployment. Moreover, scalability is compromised by time, costs and the high level of UE know-how needed to reach a consistent design. We propose a novel approach for bootstrapping spoken dialog systems based on reuse of existing transcribed and labeled data, common reusable dialog templates and patterns, generic language and understanding models, and a consistent design process. We demonstrate that our approach reduces design and development time while providing an effective system without any application specific data.

1 Introduction

Spoken dialog systems aim to identify intents of humans, expressed in natural language, and take actions accordingly, to satisfy their requests (Gorin et al., 2002). In a natural spoken dialog system, typically, first the speaker's utterance is recognized using an automatic speech recognizer (ASR). Then, the intent of the speaker is identified from the recognized sequence, using a spoken language understanding (SLU) component. This step can be framed as a classification problem for goal-oriented call routing systems (Gorin et al., 2002; Natarajan et al., 2002, among others). Then, the user would be engaged in a dialog via clarification or confirmation prompts if necessary. The role of the dialog manager (DM) is to interact in a natural way and help the user to achieve the task that the system is designed to support.

In our case we only consider automated call routing systems where the task is to reach the right route in a large call center, which could be either a live operator or an automated system. An example dialog from a telephone-based customer care application is given in Figure 1. Typically the design of such complex systems consists of several manual steps, including analysis of existing IVR (Interactive Voice Response) systems, customer service representative (CSR) interviews, customers' testimonials, CSR training material, and, when available, human-machine unconstrained speech recordings. Based on these heterogeneous requirements, the User Experience (UE) expert analyzes and defines by hand the system core functionalities: the system semantic scope (call-types) and the dialog manager strategy which will drive the human-machine interaction. Once the UE expert designs the system, large amounts of transcribed and labeled speech utterances are needed for building the ASR and SLU models.

In our previous work, we have presented active and unsupervised (or semi-supervised) learning algorithms in order to reduce the amount of labeling effort needed while building ASR and SLU systems (Tur et al., 2003; Tur and Hakkani-Tür, 2003; Riccardi and Hakkani-Tür, 2003). There we focus on a single application, and only the ASR and SLU components. In this study, we aim to exploit the labeled and transcribed data and common reusable dialog templates and patterns obtained from similar previous applications to bootstrap the whole spoken dialog system with ASR, SLU, and DM components.

The organization of this paper is as follows. Sec-

- **System:** How may I help you?
- **User:** Hello?
- *Call-type: Hello*
- **System:** Hello, how may I help you?
- **User:** I have a question.
- *Call-type: Ask(Info)*
- **System:** OK, What is your question?
- **User:** I would like to know my account balance.
- *Call-type: Request(Account_Balance)*
- **System:** I can help you with that. What is your account number?
- **User:** ...

Figure 1: An example natural language dialog

tion 2 describes briefly the AT&T Spoken Dialog System, which we use in this study, and its main components, ASR, SLU, and DM. In Section 3 we present our method to bootstrap ASR, SLU, and DM for a new application. Section 4 presents our experiments using real data from a customer care application.

2 AT&T Spoken Dialog System

Once a phone call is established, the dialog manager prompts the caller either with a pre-recorded or synthesized greetings message. At the same time, it activates the top level ASR grammar. The caller speech is then translated into text and sent to the SLU which replies with a semantic representation of the utterance. Based on the SLU reply and the implemented dialog strategy, the DM engages in a mixed initiative dialog to drive the user towards the goal. The DM iterates the previously described steps until the call reaches a final state (e.g. the call is transferred to a CSR, an IVR or the caller hangs up).

2.1 ASR

Robust speech recognition is a critical component of a spoken dialog system. The speech recognizer uses trigram language models based on Variable N-gram Stochastic Automata (Riccardi et al., 1996). The acoustic models are subword unit based, with triphone context modeling and variable number of gaussians (4-24). The output of the ASR engine (which can be the 1-best or a lattice) is then used as the input of the SLU component.

2.2 SLU

In a natural spoken dialog system, the definition of "understanding" depends on the application. In this work, we focus only on goal-oriented call classification tasks, where the aim is to classify the intent of the user into one of the predefined call-types. As a call classification example, consider the utterance in the previous example dialog *I would like to know my account balance*, in a customer care application. Assuming that the utterance is recognized correctly, the corresponding intent or the call-type would be *Request(Account_Balance)* and the action would be prompting for the account number and then telling the balance to the user or routing this call to the Billing Department.

Classification can be achieved by either a knowledge-based approach which depends heavily on an expert writing manual rules or a data-driven approach which *trains* a classification model to be used during run-time. In our current system we consider both approaches. Data-driven classification has long been studied in the machine learning community. Typically these classification algorithms try to train a classification model using the features from the training data. More formally, each object in the training data, $x_i \in \mathcal{X}_j$, is represented in the form (F_{ij}, C_{ij}), where $F_{ij} \subset \mathcal{F}_j$ is the feature set and the $C_{ij} \subset \mathcal{C}_j$ is the assigned set of classes for that object for the application j. In this study, we have used an extended version of a Boosting-style classification algorithm for call classification (Schapire, 2001) so that it is now possible to develop hand written rules to cover low frequent classes or bias the classifier decision for some of the classes. This is explained in detail in Schapire *et al.* (2002). In our previous work, we have used rules to bootstrap the SLU models for new applications when no training data is available (Di Fabbrizio et al., 2002).

Classification is employed for all utterances in all dialogs as seen in the sample dialog in Figure 1. Thus all the expressions the users can utter are classified into predefined call-types before starting an application. Even the utterances which do not contain any specific information content get a special call-type (e.g. *Hello*). So, in our case objects, $\mathcal{X}_j$ are utterances and classes, $\mathcal{C}_j$, are call-types for a given application j.

In the literature, in order to determine the application-specific call-types, first a "wizard" data collection is performed (Gorin et al., 1997). In this approach, a human, i.e. wizard, acts like the system, though the users of the system do not know about this. This method turned out to be better than recording user-agent (human-human) dialogs, since the responses to machine prompts are found to be significantly different than responses to humans, in terms of language characteristics.

2.3 DM

In a mixed-initiative Spoken Dialog System, the Dialog Manager is the key component responsible for the human-machine interaction. The DM keeps track of the specific discourse context and provides disambiguation and clarification strategies when the SLU call-types are

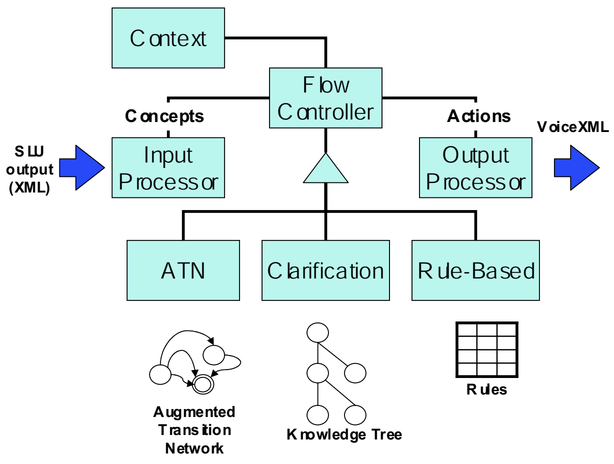

Figure 2: Dialog Manager Architecture

ambiguous or have associated low confidence scores. It also extracts other information from the SLU response in order to complete the information necessary to provide a service.

Previous work on dialog management (Abella and Gorin, 1999) shows how an object inheritance hierarchy is a convenient way of representing the task knowledge and the relationships among the objects. A formally defined Construct Algebra describes the set of operations necessary to execute actions (e.g. replies to the user or motivators). Each dialog motivator consists of a small processing unit which can be combined accordingly to the object hierarchy to build the application. Although this approach demonstrated effective results in different domains (Gorin et al., 1997; Buntschuh et al., 1998), it proposes a model which substantially differs from the *call flow* model broadly used to specify the human-machine interaction.

Building and maintaining large-scale voice-enabled applications requires a more direct mapping between specifications and programming model, together with authoring tools that simplifies the time consuming implementation, debugging, and testing phases. Moreover, the DM requires broad protocols and standard interfaces support to interact with modern enterprise backend systems (e.g. databases, http servers, email servers, etc.). Alternatively, VoiceXML (vxm, 2003) provides the basic infrastructure to build spoken dialog system, but the lack of SLU support and offline tools compromises the use in a data-driven classification applications.

Our approach proposes a general and scalable framework for Spoken Dialog Systems. Figure 2 depicts the logical DM framework architecture. The *Flow Controller* (FC) implements an abstraction of pluggable dialog strategy modules. Different algorithms can be implemented and made available to the DM engine. Our DM provides three basic algorithms. Traditional call routing systems are better described in terms of ATNs (Augmented Transition Networks) (Bobrow and Fraser, 1969). ATNs are attractive mechanisms for dialog specification since they are (a) an almost direct translation of call flow specifications, (b) easy to augment with specific mixed-initiative interactions, (c) practical to manage extensive dialog context. Complex knowledge-based tasks could be synthetically described by a variation of knowledge trees. Plan-based dialogs are effectively defined by rules and constraints.

The FC provides a synthetic XML-based language to author the appropriate dialog strategy. Dialog strategy algorithms are encapsulated using object oriented paradigms. This allows dialog authors to write sub-dialogs with different algorithms, depending on the nature of the task and use them interchangeably and exchanging variables through the local and global contexts. A complete description of the DM is out of the scope of this publication and will be covered elsewhere. We will focus our attention on the ATN module which is the one used in our experiments. The ATN engine operates on the semantic representation provided by the SLU and the current dialog context to control the interaction flow.

3 Bootstrapping a Spoken Dialog System

This section describes how we bootstrap the main components of a spoken dialog system, namely the ASR, SLU, and DM. For all modules, we assume no data from the application domain is available.

3.1 Unsupervised Learning of Language Models

State-of-the-art speech recognition systems are generally trained using in-domain transcribed utterances, preparation of which is labor intensive and time-consuming. In this work, we re-train only the statistical language models, and use an acoustic model trained using data from other applications. Typically, the recognition accuracy improves by adding more data from the application domain to train statistical language models (Rosenfeld, 1995).

In our previous work, we have proposed active and unsupervised learning techniques for reducing the amount of transcribed data needed to achieve a given word accuracy, for automatic speech recognition, when some data (transcribed or untranscribed) is available from the application domain (Riccardi and Hakkani-Tür, 2003). Iyer and Ostendorf (1999) have examined various similarity techniques to selectively sample out-of-domain data to enhance sparse in-domain data for statistical language models, and have found that even the brute addition of out-of-domain data is useful. Venkataraman and Wang (2003) have used maximum likelihood count estimation and document similarity metrics to select a single vocabulary from many corpora of varying origins and characteristics. In these studies, the assumption is that there is some domain data (transcribed and/or untranscribed) available, and its n-gram distributions are used to extend that set with additional data.

In this paper, we focus on the reuse of transcribed data from other resources, such as human-human dialogs (e.g. Switchboard Corpus, (Godfrey et al., 1992)), or human-machine dialogs from other spoken dialog applications, as well as some text data from the web pages of the application domain. We examine the style and content similarity, when out-of-domain data is used to train statistical language models and when no in-domain human-machine dialog data is available. Intuitively, the domain web pages could be useful to learn domain-specific vocabulary. Other application data can provide stylistic characteristics of human-machine dialogs.

3.2 Call-type Classification with Data Reuse

The bottleneck of building reasonably performing classification models is the amount of time and money spent for high quality labeling. By "labeling," we mean assigning one or more predefined label(s) (*call-type(s)*) to each utterance.

In our previous work, in order to build call classification systems in shorter time frames, we have employed active and unsupervised learning methods to selectively sample the data to label (Tur et al., 2003; Tur and Hakkani-Tür, 2003). We have also incorporated manually written rules to bootstrap the Boosting classifier (Schapire et al., 2002) and used it in the AT&T HelpDesk application (Di Fabbrizio et al., 2002).

In this study, we aim to reuse the existing labeled data from other applications to bootstrap a given application. The idea is forming a library of call-types along with the associated data and let the UE expert responsible for that application exploit this information source.

Assume that there is an oracle which categorizes all the possible natural language sentences which can be uttered in any spoken dialog application we deal with. Let us denote this set of universal classes with $\mathcal{C}$, such that the call-type set of a given application is a subset of that, $\mathcal{C}_j \subset \mathcal{C}$. It is intuitive that, some of the call-types will appear in all applications, some in only one of them, etc. Thus, we categorize $\mathcal{C}_j$ into three:

1. **Generic Call-types:** These are the intents appearing independent of the application. A typical example would be a request for talking to a human instead of a machine. Call this set

$$\mathcal{CG} = \{C_i : C_i \in \mathcal{C}_k, \forall k\}$$

2. **Re-usable Call-types:** These are the intents which are not generic but have already been defined for a previous application (most probably from the same or similar industry sector) and have already had labeled data. Call this set

$$\mathcal{CR}_j = \{C_i : C_i \in \mathcal{C}_k, \exists k\}$$

3. **Specific Call-types:** These are the intents specific to the application, because of the specific business needs or application characteristics. Call this set

$$\mathcal{CS}_j = \{C_i : C_i \notin \mathcal{C}_k, \forall k \neq j\}$$

Now, for each application j, we have

$$\mathcal{C}_j = \mathcal{CG} \bigcup \mathcal{CR}_j \bigcup \mathcal{CS}_j$$

It is up to the UE expert to decide which call-types are specific or reusable, i.e. the sets $\mathcal{CR}_j$ and $\mathcal{CS}_j$. Given that not any two applications are the same, deciding on whether to reuse a call-type along with its data is very subjective. There may be two applications including the intent *Request(Account_Balance)* one from a telecommunications sector, the other from a pharmaceutical sector, and the wording can be slightly different. For example while in one case we may have *How much is my last phone bill*, the other can be *do I owe you anything on the medicine*. Since each classifier can tolerate some amount of language variability and noise, we assume that if the names of the intents are the same, their contents are the same. Since in some cases, this assumption does not hold, it is still an open problem to selectively sample the portions of data to reuse automatically.

Since $\mathcal{CG}$ appears in all applications by definition, this is the core set of call-types in a new application, n. Then, if the UE expert knows the possible reusable intents, $\mathcal{CR}_n$, existing in the application, they can be added too. The bootstrap classifier can then be trained using the utterances associated with the call-types, $\mathcal{CG} \bigcup \mathcal{CR}_n$ in the call-type library. For the application specific intents, $\mathcal{CS}_n$, it is still possible to augment the classifier with a few rules as described in Schapire *et al.* (2002). This is also up to the expert to decide.

Depending on the size of the library or the similarity of the new application to the existing ones, using this approach it is possible to cover a significant portion of the intents. For example, in our experiments, we have seen that 10% of the responses to the initial prompt is a request to talk to a human. Using this system, we have the capability to continue the dialog with the user and getting the intent before sending them to a human agent.

Using this approach the application begins with a reasonably well working understanding component. One can also consider this as a more complex wizard, depending on the bootstrap model.

Another advantage of maintaining a call-type library and exploiting them by reuse is that automatically ensures consistency in labeling and naming. Note that the design of call-types is not a well defined procedure and most of the time, it is up to the expert. Using this approach it is possible to discipline the *art* of call-type design to some extent.

After the system is deployed and real data is collected, then the application specific or other reusable call-types can be determined by the UE expert to get a complete picture.

3.3 Bootstrapping the Dialog Manager with Reuse

Mixed-initiative dialogs generally allow users to take control over the machine dialog flow almost at any time of the interaction. For example, during the course of a dialog aiming to a specific task, a user may utter a new intention (speech act) and deviate from the previously stated goal. Depending on the nature of the request, the DM strategy could either decide to shift to the different context (*context shift*) or re-prompt providing additional information. Similarly, other dialog strategy patterns such as *correction, start-over, repeat, confirmation, clarification, contextual help*, and the already mentioned *context shift*, are recurring features in a mixed-initiative system.

Our goal is to derive some overall approach to dialog management that would define templates or basic dialog strategies based on the call-type structure. For the specific call routing task described in this paper, we generalized dialog strategy templates based on the categorization of the call-type presented in 3.2 and on best practice user experience design.

Generic call-types, such as *Yes, No, Hello, Goodbye, Repeat, Help*, etc., are domain independent, but are handled in most reusable sub-dialogs with the specific dialog context. When detected in any dialog turn, they trigger context dependent system replies such as informative prompts (*Help*), greetings (*Hello*) and summarization of the previous dialog turn using the dialog history (*Repeat*). In this case, the dialog will handle the request and resume the execution when the information has been provided. *Yes* and *No* generic call-types are used for confirmation if the system is expecting a yes/no answer or ignored with a system re-prompt in other contexts.

Call-types are further categorized as *vague* and *concrete*. A request like *I have a question* will be classified as vague *Ask(Info)* and will generate a clarification question: *OK. What is your question?* Concrete call-types categorize a clear routing request and they activate a confirmation dialog strategy when they are classified with low confidence scores. Concrete call-type can also have associated mandatory or optional attributes. For instance, the concrete call-type *Request(Account_Balance)* requires a mandatory attribute *AccountNumber* (generally captured by the SLU) to complete the task.

We generalized sub-dialogs to handle the most common call- type attributes (telephone number, account number, zip code, credit card, etc.) including a *dialog container* that implements the optimal flow for multiple inputs. A common top level dialog handles the initial open prompt requests. Reusable dialog templates are implemented as ATNs where the actions are executed when the network arcs are traversed and passed as parameters at run-time. Disambiguation of multiple call-types is not supported. We only consider the top scoring call-type assuming that multiple call-types with high confidence are rare events.

4 Experiments and Results

For our experiments, we selected an application from the pharmaceutical domain to bootstrap. We have evaluated the performances of the ASR language model, call classifier, and dialog manager as described below.

4.1 Speech Recognition Experiments

To bootstrap a statistical language model for ASR, we used human-machine spoken language data from two previous AT&T VoiceTone spoken dialog applications (App. 1 (telecommunication domain) and App. 2 (medical insurance domain)). We also used some data from the application domain web pages (Web). Table 1 lists the sizes of these corpora. “App. Training Data” and “App. Test Data” correspond to the training and test data we have for the new application and are used for controlled experiments. We also extended the available corpora with human-human dialog data from the Switchboard corpus (SWBD) (Godfrey et al., 1992).

Table 2 summarizes some style and content features of the available corpora. For simplification, we only compared the percentage of pronouns and filled pauses to show style differences, and the domain test data out-of-vocabulary word (OOV) rate for content variations. The human-machine spoken dialog corpora include much more pronouns than the web data. There are even further differences between the individual pronoun distributions. For example, out of all the pronouns in the web data, 35% is “you”, and 0% is “I”, whereas in all of the human-machine dialog corpora, more than 50% of the pronouns are “I”. In terms of style, both spoken dialog corpora can be considered as similar. In terms of content, the second application data is the most similar corpus, as it results in the lowest OOV rate for the domain test data. In Table 3, we show further reductions in the App. test set OOV rate, when we combine these corpora.

Figure 3 shows the effect of using various corpora as training data for statistical language models used in the

	App. 1	App. 2	Web Data	App. Training Data	App. Test Data
No. of Utterances	35,551	79,792	NA	29,561	5,537
No. of Words	329,959	385,168	71,497	299,752	47.684

Table 1: ASR Data Characteristics

	App. 1	App. 2	SWBD	Web Data	In-Domain Training Data
Percentage of Pronouns	15.14%	9.16%	14.8%	5.30%	14.5%
Percentage of Filled Pauses	2.66%	2.27%	2.74%	0%	3.26%
Test Set OOV Rate	9.79%	1.99%	2.64%	13.36%	1.02%

Table 2: Style and Content differences among various data sources.

Corpora	Test Set OOV Rate
App 1 + App 2 Data	1.53%
App 1 + App 2 + Web Data	1.22%
App 1 + App 2 + Web + SWBD Data	0.88%

Table 3: Effect of training corpus combination on OOV rate of the test data.

recognition of the test data. We also computed ASR run-time curves by varying the beam-width of the decoder, as the characteristics of the corpora effects the size of the language model. Content-wise the most similar corpus (App. 2) resulted in the best performing language model, when the corpora are considered separately. We obtained the best recognition accuracy, when we augment App. 2 data with App. 1 and the web data. Switchboard corpus also resulted in a reasonable performance, but the problem is that it resulted in a very big language model, slowing down the recognition. In that figure, we also show the word accuracy curve, when we use in-domain transcribed data for training the language model.

Once some data from the domain is available, it is possible to weight the available out-of-domain data and the web-data while reusing, to achieve further improvements. When we lack any in-domain data, we expect the UE expert to reuse the application data from the most similar sectors and/or combine all available data.

4.2 Call-type Classification Experiments

We have performed the SLU tests using the Boostexter tool (Schapire and Singer, 2000). For all experiments, we have used word n-grams of transcriptions as features and iterated Boostexter 1,100 times. In this study we have assumed that all candidate utterances are first recognized by the same automatic speech recognizer (ASR), so we deal with only text input of the same quality, which corresponds to the recognitions obtained using the language model trained from App. 1, App. 2, and Web data.

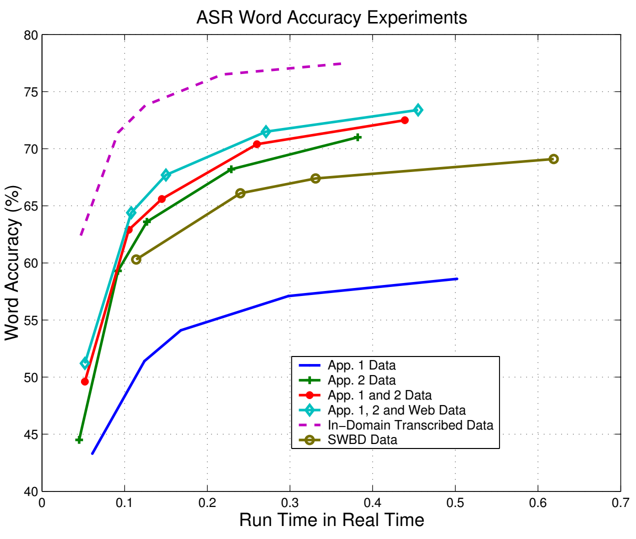

Figure 3: The word accuracy of in-domain test data, using various language models and pruning thresholds.

As the test set, we again used the 5,537 utterances collected from a pharmaceutical domain customer care application. We used a very limited library of call-types from a telecommunications domain application. We have made controlled experiments where we know the true call-types of the utterances. In this application we have 97 call-types with a fairly high perplexity of 32.81.

If an utterance has the call-type which is covered by the bootstrapped model, we expect that call-type to get high confidence. Otherwise, we expect the model to reject it by assigning the special call-type *Not(Understood)* meaning that the intent in the utterance is known to be not covered or some call-type with low confidence. Then we compute the *rejection accuracy* (RA) of the bootstrap model:

$$RA = \frac{Number\ of\ correctly\ rejected\ utterances}{Number\ of\ all\ utterances}$$

		Transcriptions		ASR Output	
	Coverage	Rejection Acc.	Classification Acc.	Rejection Acc.	Classification Acc.
In-Domain Model	100.00%	78.27%	78.27%	61.73%	61.73%
Generic Model	45.78%	88.53%	95.38%	85.55%	91.08%
Bootstrapped Model	70.34%	79.50%	79.13%	71.86%	68.40%

Table 4: SLU results using transcriptions and ASR output with the models trained with in-domain data, only generic call-types, and also with call-types from the library and rules.

In order to evaluate the classifier performance for the utterances whose call-types are covered by the bootstrapped model, we have used *classification accuracy* (CA) which is the fraction of utterances in which the top scoring call-type is one of the true call-types assigned by a human-labeler and its confidence score is more than some threshold:

$$CA = \frac{Number\ of\ correctly\ classified\ utterances}{Number\ of\ all\ utterances}$$

These two measures are actually complementary to each other. For the complete model trained with all the training data, where all the intents are covered, these two metrics are the same.

In order to see our upper bound, we first trained a classifier using 30,000 labeled utterances from the same application. First row of Table 4 presents these results using both the transcriptions of the test set and using the ASR output with around 68% word accuracy. As the confidence threshold we have chosen a hypothetical value of 0.3 for all experiments. As seen, 78.27% classification (or rejection) accuracy is the performance using all training data. This reduces to 61.73% when we use the ASR output. This is mostly because of the unrecognized words which are critical for the application. This is intuitive since ASR language model has not been trained with domain data.

Then we trained a generic model using only generic call-types. This model has achieved better accuracies as seen in the second row, since we do not expect it to distinguish among the reusable or specific call-types. Furthermore, for classification accuracy we only use the portion of the test set whose call-types are covered by the model and the call-types in this model are definitely easier than the specific ones. The drawback is that we only cover about half of the utterances. Using the ASR output, unlike the in-domain model case, did not hurt much, since the ASR already covers the utterances with generic call-types with great accuracy.

We then trained a bootstrapped model using 13 call-types from the library and a few simple rules written manually for three frequent intents. Since the library consists of an application from a fairly different domain, we could only exploit intents related to billing, such as *Request(Account_Balance)*. While determining the call-types to write rules, we actually played the expert which has previous knowledge on the application. This enabled us to increase the coverage to 70.34%.

The most impressive result of these experiments is that, we have got a call classifier which is trained without any in-domain data and can handle most utterances with almost same accuracy as the trained with extensive amounts of data. Noting the weakness of our current call-type library we expect even better performances as we augment more call-types from on-going applications.

4.3 Dialog Level Evaluation

Evaluation of spoken dialog system performances is a complex task and depends on the purpose of the desired dialog metric (Paek, 2001). While ASR and SLU can be fairly assessed off-line using utterances collected in previous runs of the baseline system, the dialog manager requires interaction with a real motivated user who will cooperate with the system to complete the task. Ideally, the bootstrap system has to be deployed in the field and the dialogs have to be manually labeled to provide accurate measure of task completion rate. Usability metrics also require direct feedback from the caller to properly measure the user satisfaction (specifically, task success and dialog cost) (Walker et al., 1997). However, we are more interested in automatically comparing the bootstrap system performances with a reference system, working on the same domain and with identical dialog strategies.

As a first order of approximation, we reused the 3,082 baseline test dialogs (5,537 utterances) collected by the live reference system and applied the same dialog turn sequence to evaluate the bootstrap system. According to the reference system call flow, the 97 call-types covered by the reference classifier are clustered into 32 DM categories (DMC). A DMC is a generalization of more specific intents.

The bootstrap system only classifies 16 call-types and 16 DMC accordingly to the bootstrapping SLU design requirements described in 4.2. This is only half of the reference system DMC coverage, but it actually addresses 70.34% of the total utterance classification task.

We simulate the execution of the dialog using data collected from a deployed system, with the following proce-

	DM Category		DM Route	
	Transcribed	ASR output	Transcribed	ASR output
concrete	44.65%	34.13%	47.18%	36.99%
concrete+conf	50.78%	42.20%	53.47%	44.32%
concrete+conf+vague/generic	67.27%	57.39%	70.67%	61.84%

Table 5: DM Evaluation results.

dure: for each dialog $\mathcal{D}_i$ in the reference data set, we pass the utterance j to the bootstrap classifier and select the result with the highest confidence score c_j. We use two confidence score thresholds, T_{high}, for acceptance, and T_{low}, for rejection. Call-types, whose confidence scores are in between these two thresholds are confirmed. Then:

1. the dialog is considered as successful if the following condition is verified:

   $$score(c_j) \geq T_{high} \wedge c_j \in \mathcal{DMC}_j \wedge c_j \in Concrete$$

   where T_{high} is the acceptance threshold, $\mathcal{DMC}_j$ is the manually labeled reference DM category set for the turn j, and $Concrete$ is the set of concrete call-types;

2. the dialog is considered as successful with confirmation if the following condition is verified:

   $$T_{low} \leq score(c_j) < T_{high}$$

   $$\wedge c_j \in \mathcal{DMC}_j \wedge c_j \in Concrete$$

   where T_{low} is the rejection threshold;

3. if in any turn of the dialog, a mismatching concrete call-type is found, the dialog is considered as unsuccessful:

   $$score(c_j) \geq T_{high} \wedge c_j \notin \mathcal{DMC}_j \wedge c_j \in Concrete$$

If none of the conditions above are satisfied, we apply steps 1 and 3 with a lower threshold and $c_j \in Vague \vee c_j \in Generic$, assuming that the dialog did not contain any relevant user intention.

A further experiment considers only the final routing destinations (e.g. specific type of agent or the automatic fulfillment system destination). Both reference and bootstrap systems direct calls to 12 different destinations, implying that a few DM categories are combined into the same destination. This quantifies how effectively the system routes the callers to the right place in the call center and, conversely, gives some metric to evaluate the missed automation and misrouted calls. The test has been executed for both transcribed and untranscribed utterances. Results are shown in Table 5. Even with a modest 50% DM Categories coverage, the bootstrap system shows an overall task completion of 67.27% in case of transcribed data and 57.39% using the output generated by the bootstrap ASR. When considering the route destinations, completion increases to 70.67% and 61.84% respectively. This approach explicitly ignores the dialog context, but it contemplates the call-type categorization, the confirmation mechanism and the final route destination, that would be missed in the SLU evaluation. Although a more completed evaluation analysis is needed, lower bound results are indicative of the overall performances.

5 Summary

This paper shows that bootstrapping a spoken dialog system reusing existing transcribed and labeled data from out-of-domain human-machine dialogs, common reusable dialog templates and patterns, is possible to achieve operational performances. Our evaluations on a call classification system using no domain specific data indicate 67% ASR word accuracy, 79% SLU call classification accuracy with 70% coverage, and 62% routing accuracy with 50% DM coverage. Our future work consists of developing techniques to refine the bootstrap system when application domain data become available.

Acknowledments

We would like to thank Liz Alba, Lee Begeja, Harry Blanchard, David Gibbon, Patrick Haffner, Zhu Liu, Mazin Rahim, Bernard Renger, Behzad Shahraray, and Jay Wilpon for many helpful discussions.

References

A. Abella and A. Gorin. 1999. Construct algebra: Analytical dialog management. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, Washington, D.C., June.

D. Bobrow and B. Fraser. 1969. An augmented state transition network analysis procedure. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pages 557–567, Washington, D.C, May.

B. Buntschuh, C. Kamm, G. Di Fabbrizio, A. Abella, M. Mohri, S. Narayanan, I. Zeljkovic, R. D. Sharp, J.Wright, S. MarcusU, J. Shaffer., R. Duncan., and

J. G. Wilpon. 1998. VPQ : A spoken language interface to large scale directory information. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Sydney, Australia, November.

G. Di Fabbrizio , D. Dutton, N. Gupta, B. Hollister, M. Rahim, G. Riccardi, R. Schapire, and J. Schroeter. 2002. AT&T help desk. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Denver, CO, September.

J. J. Godfrey, E. C. Holliman, and J. McDaniel. 1992. Switchboard: Telephone speech corpus for research and development. In *Proceedings of International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 1, pages 517–520, San Francisco, USA, March.

A. L. Gorin, G. Riccardi, and J. H. Wright. 1997. How May I Help You? . *Speech Communication*, 23:113–127.

A. L. Gorin, G. Riccardi, and J. H. Wright. 2002. Automated natural spoken dialog. *IEEE Computer Magazine*, 35(4):51–56, April.

R. Iyer and M. Ostendorf. 1999. Relevance weighting for combining multi-domain data for n-gram language modeling. *Computer Speech and Language*, 13:267–282.

P. Natarajan, R. Prasad, B. Suhm, and D. McCarthy. 2002. Speech enabled natural language call routing: BBN call director. In *Proceedings of the International Conference on Spoken Language Processing (ICSLP)*, Denver, CO, September.

T. Paek. 2001. Empirical methods for evaluating dialog systems. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL) Workshop on Evaluation Methodologies for Language and Dialogue Systems*, Toulouse, France, July.

G. Riccardi and D. Hakkani-Tür. 2003. Active and unsupervised learning for automatic speech recognition. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, Geneva, Switzerland, September.

G. Riccardi, R. Pieraccini, and E. Bocchieri. 1996. Stochastic automata for language modeling. *Computer Speech and Language*, 10:265–293.

R. Rosenfeld. 1995. Optimizing lexical and n-gram coverage via judicious use of linguistic data. In *Proceedings of European Conference on Speech Communication and Technology (EUROSPEECH)*, volume 2, pages 1763–1766, Madrid, Spain, September.

R. E. Schapire and Y. Singer. 2000. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 39(2/3):135–168.

R. E. Schapire, M. Rochery, M. Rahim, and N. Gupta. 2002. Incorporating prior knowledge into boosting. In *Proceedings of the International Conference on Machine Learning (ICML)*, Sydney, Australia, July.

R. E. Schapire. 2001. The boosting approach to machine learning: An overview. In *Proceedings of the MSRI Workshop on Nonlinear Estimation and Classification*, Berkeley, CA, March.

G. Tur and D. Hakkani-Tür. 2003. Unsupervised learning for spoken language understanding. In *Proceedings of the European Conference on Speech Communication and Technology (EUROSPEECH)*, Geneva, Switzerland, September.

G. Tur, R. E. Schapire, and D. Hakkani-Tür. 2003. Active learning for spoken language understanding. In *Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Hong Kong, May.

Anand Venkataraman and Wen Wang. 2003. Techniques for effective vocabulary selection. In *Proceedings of European Conference on Speech Communication and Technology (EUROSPEECH)*, pages 245–248, Geneva, Switzerland, September.

2003. Voice extensible markup language (VoiceXML) version 2.0. http://www.w3.org/TR/voicexml20/.

M. A. Walker, D. J. Litman, C. A. Kamm, and A. Abella. 1997. PARADISE : A framework for evaluating spoken dialogue agents. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)-Conference of the European Chapter of the Association for Computational Linguistics (EACL)*, Madrid, Spain, July.