Automatic annotation of COMMUNICATOR dialogue data for learning dialogue strategies and user simulations

Kallirroi Georgila, Oliver Lemon, and James Henderson
Human Communication Research Centre
School of Informatics, University of Edinburgh
{kgeorgil,olemon,jhender6}@inf.ed.ac.uk

Abstract

We present and evaluate an automatic annotation system which builds "Information State Update" (ISU) representations of dialogue context for the COMMUNICATOR (2000 and 2001) corpora of human-machine dialogues (approx 2300 dialogues). The purposes of this annotation are to generate training data for reinforcement learning (RL) of dialogue policies, to generate data for building user simulations, and to evaluate different dialogue strategies against a baseline. The automatic annotation system uses the DIPPER dialogue manager. This produces annotations of user inputs and dialogue context representations. We present a detailed example, and then evaluate our annotations, with respect to the task completion metrics of the original corpus. The resulting data has been used to train user simulations and to learn successful dialogue strategies.

1 Introduction

We present and evaluate an automatic annotation system which builds "Information State Update" (ISU) representations of dialogue context (Larsson and Traum, 2000; Bos et al., 2003; Lemon and Gruenstein, 2004) for the COMMUNICATOR (2000 and 2001) corpora of human-machine dialogues (2331 dialogues) (Walker et al., 2001). The purpose of this annotation is to generate enough training data for a reinforcement learning (RL) approach to dialogue management, and also to be able to build user simulations, and to evaluate different dialogue strategies against a baseline. In general, for such an approach we require data that has either been generated and logged by ISU systems or that has been subsequently annotated (or a mixture of both).

A particular problem is that although the COMMUNICATOR corpus (recently released by the LDC) is the largest corpus of speech-act-annotated dialogues that we know of, it does not meet our requirements on corpus annotation for dialogue strategy learning and user simulation. For example, the user dialogue inputs were not annotated with speech act classifications, and no representation of dialogue context was annotated. We explain how we addressed such problems by building an automated annotation system which extends the COMMUNICATOR corpus, and we evaluate the resulting annotations. Note that prior work on ISU annotations (Poesio et al., 1999) was not automated, and was not suitable for large-scale annotations. We first

survey basic principles for annotating dialogue data with feature values for learning approaches. Section 2 describes the annotation system and section 3 presents our evaluation of the automatic annotations.

1.1 The DATE annotation scheme

The DATE (Dialogue Act Tagging for Evaluation) scheme (Walker et al., 2001) was developed for providing quantitative metrics for comparing and evaluating the 9 different DARPA COMMUNICATOR spoken dialogue systems. The scheme employs three orthogonal dimensions of utterance classification:

- *conversational domain*: about_task, about_communication, situation_frame
- *task-subtask*: top_level_trip (origin, destination, date, time, airline, trip_type, retrieval, itinerary), ground (hotel, car)
- *speech act*: request_info, present_info, offer, acknowledgement, status_report, explicit_confirm, implicit_confirm, instruction, apology, opening/closing.

The conversational domain dimension categorises each utterance as belonging to a particular "arena of conversational action". About_task refers to the domain task (in COMMUNICATOR this is air travel, hotel, and car-rental booking), and about_communication refers to conversational actions managing the communication channel (e.g. "are you still there?"). Situation_frame utterances manage the "culturally relevant framing expectations" in the dialogue (e.g. that the conversation will be in English, or that the system cannot issue airline tickets).

The task-subtask dimension relates to a model of the domain tasks that the dialogue system is designed to support. In COMMUNICATOR there were 2 main tasks: booking a flight ("top_level_trip"), and "ground" which was to determine whether the user also wanted to book a car rental and/or a hotel. The subtasks were elements such as finding the dates and times of the flights.

The speech_act dimension relates to the utterance's communicative goal. The speech acts used are relatively standard, and are described in detail in (Walker et al., 2001). Note that in the COMMUNICATOR data only the system's side of the dialogue is already annotated using the DATE scheme.

1.2 Annotation principles for ISU systems

The question arises of what types of information should ideally be logged or annotated for the purposes of building simulated users and optimising ISU dialogue systems via RL (Young, 2000). We can divide the types of information required into 5 main levels: *dialogue-level*, *task-level*, *low-level*, *history-level*, and *reward-level*. We also divide the logging and annotations required into information about *utterances*, and information about *states*. Utterances (by humans or systems) will have dialogue-level, task-level, and low-level features, while dialogue states will additionally contain some history-level information (see figure 2). Entire dialogues will be assigned reward features, e.g. taken from questionnaires filled by users.

A notable constraint on the information to be useful for machine learning is that all captured features should in principle be available to a dialogue system at runtime – so that a dialogue system using a learned policy can compute a next action in any state. This excludes, for example, word error rate from the state information usable for RL, since it can only be computed after transcription of user speech. In this case, for example, automatic speech recognition (ASR) confidence scores should be used instead. It also means that we need to annotate the ASR hypotheses of the systems, rather than the transcribed user utterances.

We now present an extended version of the

DATE scheme, producing sequences of dialogue information states that feed into learning algorithms and user simulations.

2 The automated annotation system

The annotation of the COMMUNICATOR data with information states was implemented using DIPPER (Bos et al., 2003) and OAA (Cheyer and Martin, 2001). Several OAA agents have been developed:

The first OAA agent *(readXMLfile)* is used for reading the original COMMUNICATOR corpus XML files, which contain information about dialogues, turns, utterances, transcriptions, and so on. When the agent reads information from an XML file, a corresponding DIPPER update rule fires and the dialogue information state is updated accordingly. Each information state corresponds to an utterance in the COMMUNICATOR data and a turn may contain several utterances.

A second OAA agent *(saveISsequence)* appends the current information state values to the file that will finally contain the whole sequence of information states (a DTD defining the format of IS-sequence files is available).

2.1 Confidence scoring

Ideally we would have dialogue data that contains ASR confidence scores. Unfortunately the COMMUNICATOR data does not have this information. However, the COMMUNICATOR data contains both the output of the speech recognition engine for a user utterance and a manual transcription of the same utterance carried out by a human annotator. We consider the word error rate (WER) to be strongly related to confidence scores and thus each time a user utterance is read from the XML file a third agent is called to estimate error rates (the *ComputeErrorRates* agent). Four different error rates are estimated: classical WER, WER-noins, SER, and KER.

WER-noins is WER without taking into account insertions. The distinction between WER and WER-noins is made because WER shows the overall recognition accuracy whereas WER-noins shows the percentage of words correctly recognised. The sentence error rate (SER) is computed on the whole sentence. All the above error estimations have been performed using the HResults tool of HTK (Young et al., 2002), which is called by *ComputeErrorRates*. Finally the keyword error rate (KER) is also computed by *ComputeErrorRates* (after the utterance has been parsed) and shows the percentage of the correctly recognised keywords (cities, dates, times, etc.). This is also a very important metric regarding the efficiency of the dialogues.

2.2 Interpreting user utterances, extending DATE

Even though all the above agents play a crucial rule in the annotation task, the most important subtask is to interpret the user's input and find its effect on the dialogue, or in other words to associate the user utterances with the correct speech acts and tasks. Multiple levels of parsing are thus required and are performed using Prolog clauses (part of the DIPPER *.resources* file).

Unfortunately, in the original COMMUNICATOR data XML files there is no distinction between the origin and destination cities in multiple-leg trips. That is, the tag "dest_city" could be used for any type of destination, regardless of whether the trip is single or multiple-leg. However, we believe that it is important to annotate these distinctions so that there is no overwriting of the values in filled slots such as "dest_city", "depart_date", etc. Moreover, the COMMUNICATOR data does not distinguish between departure and arrival dates or times, and sometimes it has times labelled as dates.

We use the following extended tasks and speech acts for annotating user utterances. These are in addition to the DATE scheme (Walker et al., 2001) used for the system

prompts annotation:

- Tasks which take values: continue_dest_city, depart_date, continue_depart_date, return_depart_date, arrive_date, continue_arrive_date, return_arrive_date, depart_time, continue_depart_time, return_depart_time, arrive_time, continue_arrive_time, return_arrive_time.

- Tasks which are either present or absent: no_continue_trip, return_trip, no_return_trip, accept_hotel_offer, reject_hotel_offer, accept_flight_summary, reject_flight_summary, accept_car_offer, reject_car_offer, accept_ground_offer, reject_ground_offer, accept_flight_offer, reject_flight_offer, hotel_city, car_interest, car_rental, rental_company, no_airline_preference, change_airline, flight_interest, send_itinerary, price_itinerary, id_number, number, continue, request_help, request_repetition, request_stop, bye, nonstop_flight, start_over.

- User speech acts: provide_info, reprovide_info, correct_info, reject_info, yes_answer, no_answer, question, command.

2.3 Computing grounding information

Part of our task is also to compute dialogue context information from the existing COMMUNICATOR annotations – for example, which utterances are grounded.

During processing the user's utterance the automatic annotation system will take into account the history of the dialogue and the labels on previous system utterances and then decide (via processing based on "patterns", see below) whether one or more slots should be filled, grounded, or even emptied if the slot is not confirmed. We define a piece of information as "grounded" (according to the system's perspective) only if it has been positively confirmed. Thus grounding processing can only take place after system utterances labelled as explicit_ or implicit_ confirmation. Here one could worry that our computation of grounding information is based on the assumption that the COMMUNICATOR systems had some notion of grounding in their algorithms, but that this kind of information is not included in the original COMMUNICATOR corpus. Nevertheless, the fact that we attempt to ground a slot only when the system attempts confirmation makes our assumption "safe". Moreover, only the speech recognition output is used for processing and deciding on the slots that will be filled or grounded. The human transcription of the user's input is only considered for computing error rates as explained above. This also ensures that we do not base our annotation on information that the systems did not have in runtime.

2.3.1 Example ISU annotation

We now examine an extract from the original COMMUNICATOR 2001 data (figure 1) and its new ISU annotation (figure 2). System utterances are marked with "S(n)" and user utterances as "U(n)" where n is the number of the utterance. For the system utterances the speech act and task pairs are given, for the user utterances only the speech recognition output is provided.

In utterance (U3) the user gives the departure date and time. However, the speech recognition output "october three first" was not considered by the system to be a valid date so the system understands only the time "late morning" and tries to confirm it in (S6). As we see in (S6) the speech act is "implicit_confirm" and the task is tagged as "depart_arrive_date" instead of "depart_arrive_time". Similar phenomena cause problems for correctly annotating the dialogues. In the above example, in (U3) our automatic annotation system will fill slot "de-

```
(S1) what city are you leaving from?
              (request_info, orig_city)
(U1) <CITY>hartford connecticut</CITY>
(S2) a flight from <CITY>hartford</CITY>
              (implicit_confirm, orig_city)
(S3) where would you like to go?
              (request_info, dest_city)
(U2) <CITY>orlando florida</CITY>
(S4) traveling to <CITY>orlando</CITY>
              (implicit_confirm, dest_city)
(S5) on what date would you like to travel?
              (request_info, depart_arrive_date)
(U3) <DATE_TIME>october three first
                          late morning</DATE_TIME>
(S6) traveling <DATE_TIME>late morning</DATE_TIME>
              (implicit_confirm, depart_arrive_date)
(S7) on what date would you like to travel?
              (request_info, depart_arrive_date)
(U4) <DATE_TIME>october thirty one</DATE_TIME>
```

Figure 1: Original COMMUNICATOR data.

part_time" with the value "late morning" and it will ground the "dest_city" slot. Then it reads the next system utterance (S6). Note that if it considers only the task label (depart_arrive_date) it will attempt to ground the wrong slot "depart_arrive_date", or in other words it will try to ground a slot that has not been filled yet. Therefore routines have been implemented so that the system can distinguish between valid dates or times.

In figure 2 we can see the automatically annotated Information State[1] corresponding to the dialogue context after U3 (the actual system output is in XML, but we do not show that here). Note especially the grounding of "dest_city" information in this move, and the history level of the annotation, which contains the sequences of speech acts and filled and grounded slots for the entire dialogue.

Note that full dialogues are also annotated with reward level features (e.g. actual task completion) from the PARADISE evaluations (Walker et al., 2000). These are used in reinforcement learning with the data.

In order to further explain the "patterns" we use to compute grounding, consider a variation on the above example. Imagine that in U3 the user does not give the departure date

```
DIALOGUE LEVEL
Turn: user
TurnStartTime: 988306674.170
TurnEndTime: 988306677.510
TurnNumber: 5
Speaker: user
UtteranceStartTime: 988306674.170
UtteranceEndTime: 988306677.510
UtteranceNumber: 5
ConvDomain: [about_task]
SpeechAct: [provide_info]
AsrInput: <date_time>october three first late
          morning</date_time>
TransInput: <date_time>october thirty first late
          morning</date_time>
System Output:

TASK LEVEL
Task: [depart_time]
FilledSlotValue: [late morning]
FilledSlot: [depart_time]
GroundedSlot: [dest_city]

LOW LEVEL
WordErrorRatenoins: 20.00
WordErrorRate: 20.00
SentenceErrorRate: 100.00
KeyWordErrorRate: 50.00

HISTORY LEVEL
SpeechActsHist: [yes_answer],opening_closing,[],
  opening_closing,instruction,request_info,
  [provide_info],implicit_confirm,request_info,
  [provide_info],implicit_confirm,request_info,
  [provide_info]
TasksHist: [null],meta_greeting_goodbye,[],
  meta_greeting_goodbye,meta_instruct,orig_city,
  [orig_city],orig_city,dest_city,[dest_city],
  dest_city,depart_arrive_date,[depart_time]
FilledSlotsHist: [null],[],[orig_city],[dest_city],
                 [depart_time]
FilledSlotsValuesHist: [yes],[],[hartford connecticut],
                  [orlando florida], [late morning]
GroundedSlotsHist: [],[],[],[orig_city],[dest_city]
```

Figure 2: Information State after U3.

but instead only replies to the confirmation prompt about the destination city (S4). There are 6 general ways the user could reply [2] : yes-class, e.g. "yes"; no-class, e.g. "no"; yes-class, city, e.g. "yes, orlando"; no-class, city, e.g. "no, boston"; no-class, city, city, e.g. "not orlando, boston"; city, e.g. "orlando".

In the first 5 cases it is easy for the annotation system to infer that there is positive or negative confirmation and thus ground the slot or not accordingly because of the appearance of "yes-class" or "no-class". However, in the last case the annotation system should compare the user's utterance with

[1] Items appearing between [brackets] are user inputs (sometimes not annotated) and other items are system actions.

[2] The "yes-class" corresponds to words or expressions like "yes", "okay", "right", "correct", etc. In the same way "no-class" stands for "no", "wrong", and so on.

the previous system's prompt for confirmation in order to decide whether the slot should be grounded or not. If the user says "orlando" he *re-provides* information and the slot "dest_city" is grounded whereas if he/she utters "boston" he/she corrects the system (correct_info), which means that the slot "dest_city" is not grounded and therefore its current value will be removed. In the "no-class, city, city" case the user rejects the value of the slot and corrects it at the same time. These are examples of the patterns used to compute grounding.

2.3.2 Confirmation strategies

When computing grounding it is important to take into account the different ways in which COMMUNICATOR systems ground information through various types of confirmation. In general all the COMMUNICATOR systems follow one of 2 general confirmation strategies. In the first strategy the system asks the user to fill a slot, then asks for confirmation (explicit or implicit), and moves to the next slot if the user confirms, or may keep asking for confirmation if the user does not cooperate. In the second strategy the system asks the user to fill several slots and then attempts to confirm them in one single turn. That means that the system's turn could consist of several utterances labelled as "explicit_confirm" or "implicit_confirm". A third strategy, which is a variation of the second strategy is when the system tries to confirm several slots in a single action, e.g. "explicit_confirm, trip", "implicit_confirm, orig_dest_city". Before confirmation the slots could be filled either in a single turn or in multiple turns.

For the first and third confirmation strategies it proves adequate to look only 1 or 2 steps backwards in the history of system utterances, whereas for the second strategy looking further back is required. We consider only the following speech acts: *request_info*, *explicit_confirm*, *implicit_confirm*, and *offer*. Other utterances (e.g. instructions) are not taken into account because they do not affect whether a slot will be filled or grounded.

Note that first the annotation system extracts the speech acts and possible tasks related to the current user utterance and then attempts to ground based on this information. Any kind of disambiguation required , e.g. to decide whether the speech act should be tagged as "provide_info" or "reprovide_info", is done before grounding. We deal with possible task ambiguity simultaneously with grounding e.g. if the user uttered a "city" name we cannot be sure whether it refers to an origin or destination city until we consider the context. The reason for this sequential procedure is that we want grounding to be computed exactly in the same way for both our annotation and simulation systems, so that we are able to straightforwardly compare our simulated dialogues with COMMUNICATOR data (Henderson et al., 2005). In simulation the only information we have is the list of tasks and speech acts for the user's input and not the ASR or real utterance transcription. For the first two confirmation strategies the annotation system should check whether the tasks extracted by parsing the user's utterance are included in the task labels of the previous explicit- or implicit- confirmation system prompts. The "explicit_confirm, trip" case adds further difficulty to grounding calculation because of the general task "trip". Thus the annotation system has to parse the system prompt to detect the slots that the system attempts to confirm. Then according to the type of speech act (reprovide_info, provide_info, correct_info, etc.) the system grounds one or more previously filled slots or fills one or more new ones.

3 Evaluating the automatic annotations

We evaluated our automatic annotation system by automatically comparing its output

with the actual (ATC) and perceived (PTC) task completion metrics as they are given in the COMMUNICATOR corpus. Our evaluation is restricted in the 2001 corpus because no such metrics are available for the 2000 data collection. If the final state of a dialogue – that is, the information about the filled and grounded slots – agrees with the ATC and PTC for the same dialogue, this indicates that the annotation is consistent with the task completion metrics. We consider only dialogues where the tasks have been completed successfully – in these dialogues we know that all slots have been correctly filled and grounded[3] and thus the evaluation process is simple to automate. This automatic method cannot give us exact results – it only indicates whether the dialogue is annotated more or less correctly.

We have applied our automatic evaluation method on the flight-booking portions of the automatically annotated COMMUNICATOR corpora. The results are that, for dialogues where ATC or PTC is marked as "1" or "2" (i.e. where the flight booking portion of the dialogue was successful or was considered to be successful), the current automatic annotations for the whole corpus showed 88.47% of the required slots to be filled ("filled slots accuracy") and 71.56% of the slots to be grounded ("grounded slots accuracy"). Detailed results are depicted in table 1.

The IBM system avoided confirmation and therefore we could not obtain results for the "grounded slots accuracy". In cases where the system attempts to confirm more than one slots in a single turn (second and third confirmation strategies), if the user gives a simple "no_answer" there is no way for the annotation system to detect the slot that the "no_answer" refers to. This can lead to fewer slots being grounded. One of the rules that the annotation system uses in grounding calculation is that only filled slots can be grounded, mostly to ensure that the system policies trained with the COMMUNICATOR annotated corpus (e.g. using RL) will be reasonable. This rule can cause problems in cases where for example the system knows the user's residence and therefore does not ask for the "orig_city" but in the sequel tries to confirm it, or when the user gives a negative confirmation to a filled slot value (thus the filled slot is emptied) but the system performs a second confirmation request with an alternative slot value. Now even if the user gives a "yes_answer" the slot will not be grounded because it is not filled anymore. The above observations explain the low scores of the Colorado and ATT systems (and to a lesser extent CMU) for "grounded slots accuracy".

System	Number of dialogues	Filled slots	Confirmed slots
ATT	122	91.15	65.31
BBN	126	86.17	84.96
CMU	114	80.09	69.08
COL	152	84.83	55.40
IBM	165	94.51	-
LUC	127	93.44	78.90
MIT	159	89.19	74.42
SRI	85	87.08	78.28
ALL	1050	88.47	71.56

Table 1: ISU annotation accuracy for COMMUNICATOR 2001 data.

Our future work will focus on dealing with the above problems (e.g. by being more selective as to where some rules are applied). Moreover, we plan to perform manual evaluation of a portion of randomly selected annotated dialogues. Preliminary manual annotation has shown that not only the flight-booking portions of the data have been annotated with a high accuracy but also the hotel and car rental bookings.

As described in (Henderson et al., 2005), the first results of our supervised and rein-

[3]Error analysis showed that this assumption that the successful dialogues had all slots grounded (not just fi lled) is too strong.

forcement learning techniques trained with the this data are promising, which also indicates that a significant number of dialogues have been annotated accurately.

4 Conclusion

We explained that the original COMMUNICATOR data (2000 & 2001) is not sufficient for our purposes (of learning dialogue strategies and user simulations from a corpus) since it does not contain speech-act annotations of user utterances or representations of dialogue contexts. We briefly reviewed the DATE annotation scheme, and our extensions to it. We then described an automatic annotation system which uses DIPPER. This annotates user inputs and dialogue "information state" context representations. We presented an example, discussed grounding and confirmation strategies, and evaluated our annotations with respect to the task completion metrics of the original corpus. This resulting data has been used to learn successful dialogue strategies (Henderson et al., 2005), and to train user simulations (Georgila et al., 2005).

Finally, we think that this automatic annotation system could be extended and altered for use in producing ISU annotations for other dialogue corpora – in particular for human-machine dialogue corpora where the semantics of the system output is already logged by the dialogue system itself.

Acknowledgements

This work is funded by the European Commission's 6th framework project "TALK: Talk and Look, Tools for Ambient Linguistic Knowledge" (IST 507802). We thank Johanna Moore for proposing this data set.

References

Johan Bos, Ewan Klein, Oliver Lemon, and Tetsushi Oka. 2003. DIPPER: Description and Formalisation of an Information-State Update Dialogue System Architecture. In *4th SIGdial Workshop on Discourse and Dialogue*, pages 115–124, Sapporo.

Adam Cheyer and David Martin. 2001. The Open Agent Architecture. *Journal of Autonomous Agents and Multi-Agent Systems*, 4(1/2):143–148.

Kallirroi Georgila, James Henderson, and Oliver Lemon. 2005. Learning User Simulations for Information State Update Dialogue Systems. In *Eurospeech, (submitted)*.

James Henderson, Oliver Lemon, and Kallirroi Georgila. 2005. Hybrid Reinforcement/Supervised Learning for
Dialogue Policies from COMMUNICATOR data. In *IJCAI workshop on Knowledge and Reasoning in Practical Dialogue Systems, (submitted)*.

Staffan Larsson and David Traum. 2000. Information state and dialogue management in the TRINDI Dialogue Move Engine Toolkit. *Natural Language Engineering*, 6(3-4):323–340.

Oliver Lemon and Alexander Gruenstein. 2004. Multithreaded context for robust conversational interfaces: context-sensitive speech recognition and interpretation of corrective fragments. *ACM Transactions on Computer-Human Interaction (ACM TOCHI)*, 11(3):241– 267.

M. Poesio, R. Cooper, S. Larsson, C. Matheson, and D. Traum. 1999. Annotating conversations for information state update. In *Proceedings of Amstelogue'99 workshop on the semantics and pragmatics of dialogue.*

Marilyn A. Walker, Candace A. Kamm, and Diane J. Litman. 2000. Towards Developing General Models of Usability with PARADISE. *Natural Language Engineering*, 6(3).

Marilyn A. Walker, Rebecca J. Passonneau, and Julie E. Boland. 2001. Quantitative and Qualitative Evaluation of Darpa Communicator Spoken Dialogue Systems. In *Meeting of the Association for Computational Linguistics*, pages 515–522.

Steve Young, Gunnar Evermann, Dan Kershaw, Gareth Moore, Julian Odell, Dave Ollason, Dan Povey, Valtcho Valtchev, and Phil Woodland. 2002. *The HTK Book*. Cambridge University Engineering Department. (for HTK version 3.2).

SJ Young. 2000. Probabilistic methods in spoken dialogue systems. *Philosophical Transactions of the Royal Society (Series A)*, 358(1769):1389–1402.