Rapidly scaling dialog systems with interactive learning

Jason D. Williams*, Nobal B. Niraula*, Pradeep Dasigi*, Aparna Lakshmiratan, Carlos Garcia Jurado Suarez, Mouni Reddy, and Geoff Zweig

Abstract In personal assistant dialog systems, *intent models* are classifiers that identify the intent of a user utterance, such as to add a meeting to a calendar, or get the director of a stated movie. Rapidly adding intents is one of the main bottlenecks to *scaling* — adding functionality to — personal assistants. In this paper we show how *interactive learning* can be applied to the creation of statistical intent models. Interactive learning [10] combines model definition, labeling, model building, active learning, model evaluation, and feature engineering in a way that allows a domain expert — who need not be a machine learning expert — to build classifiers. We apply interactive learning to build a handful of intent models in three different domains. In controlled lab experiments, we show that intent detectors can be built using interactive learning, and then improved in a novel end-to-end visualization tool. We then applied this method to a publicly deployed personal assistant — Microsoft Cortana — where a non-machine learning expert built an intent model in just over two hours, yielding excellent performance in the commercial service.

1 Introduction

Personal assistant dialog systems are increasingly a part of daily life, with examples including Microsoft Cortana, Apple's Siri, Google Now, and Nuance Dragon Go.

Jason D. Williams, Aparna Lakshmiratan, Carlos Garcia Jurado Suarez, and Geoff Zweig
Microsoft Research, Redmond, WA, USA, e-mail: jason.williams@microsoft.com

Nobal B. Niraula†
University of Memphis, Memphis, TN, USA e-mail: nbnraula@memphis.edu

Pradeep Dasigi†
Carnegie Mellon University, Pittsburgh, PA, USA e-mail: pdasigi@cs.cmu.edu

† Work done while at Microsoft Research ·
* These authors contributed equally to this work

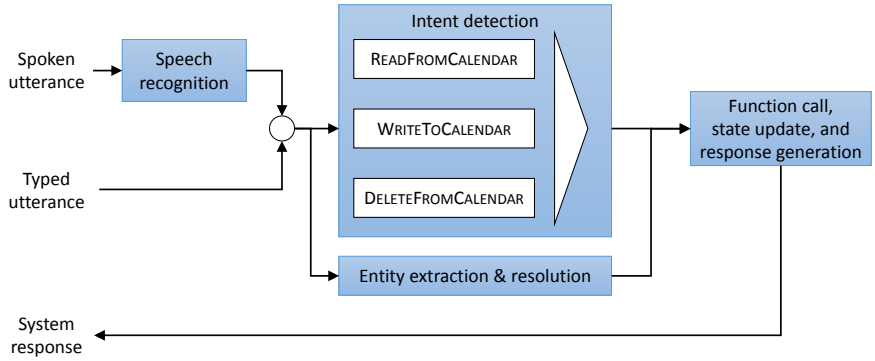

Fig. 1 Processing pipeline used for personal assistant dialog systems. First, utterances which are spoken are converted to text. Intent detection and entity extraction/resolution are then performed on the text. The resulting intent and entities are used to select and call a function. This function optionally updates an internal dialog state, then produces a response. The cycle can then repeat.

Figure 1 shows a high-level pipeline for these systems. First, spoken input is recognized using an open-domain automatic speech recognizer (ASR) and converted to words, such as "Am I free today at noon?". This step is skipped if input is provided by text. Next, the input words are processed by intent detectors to infer the user's intent, such as READFROMCALENDAR. In parallel, entity extraction identifies utterance substrings that contain entities such as the date "today" or "noon", and entity resolution maps those substrings to canonical forms such as `2014-09-11` or `12:00:00Z-08:00:00`. Finally, a function is called that takes the intent and entities as input, optionally updates an internal state, and produces a response as output. The cycle then repeats. Although variations exist — for example, intent detection and entity extraction/resolution may be done jointly — these are the high-level components of personal assistant dialog systems.

To add new functionality to state-of-the-art commercial systems, only certain stages of the pipeline need to be modified. On the one hand, the ASR service generally remains unchanged, because the coverage of modern ASR platforms is quite broad. Also, it is often the case that the function calls to produce a response are already available — for example, on most mobile phone platforms, APIs already exist for manipulating a user's calendar information. On the other hand, adding a new intent almost always requires building a new intent detector model. New entity extractors are only sometimes needed, because many entity types can be re-used for new intents: for example, the intents to *read*, *write*, and *delete* an appointment from a calendar all share the same entities: times, dates, locations, and so on.

In sum, quickly building new intent detector models is the key step in adding new functionality to a personal assistant dialog system. Yet building new intent detectors is often a slow process. One reason is that typically many people are involved, requiring coordination and scheduling. For example, a *data engineer* collects utterance data that contains instances of the target intent; a *user experience designer*

creates a labeling instruction document that explains the new intent; a *crowd-source engineer* creates a crowd-sourcing task where workers apply the labeling instructions to data; and a *machine-learning expert* uses the data to build an intent detection model. Another problem is that issues with the definition of the intent often surface only at the end of the process when model performance is measured, requiring the whole loop to be repeated. Overall, the entire process can take weeks.

In this paper we introduce a new method for building intent detector models that reduces the time required to a few hours. The work is done by a *single person*, who is an expert in the domain of interest, but is *not* an expert in machine learning. The key idea is to use *interactive learning* [10], which interleaves intent definition, active learning, model building, model evaluation, and feature engineering (described in detail in Section 3).

This paper is organized as follows. The next section reviews intent detection and related work, then section 3 introduces interactive learning and explains how it is applied to building a single intent model. Section 4 presents an evaluation, then section 5 introduces and evaluates an end-to-end tool that enables developers to correct any stage of the intent/entity processing pipeline. Section 6 describes a live deployment in Microsoft Cortana of an intent model built using interactive learning. Section 7 briefly concludes.

2 Background and related work

Intent detector models are classifiers that map from a sequence of words to one of a set of pre-defined intents — for example, from "Am I free this afternoon" to READFROMCALENDAR, which is one of the pre-defined intents [16, 14]. A typical domain like calendaring has on the order of a dozen intents. In this paper, a *binary* classifier is trained for each intent, which allows new intents to be built and tested independently, facilitating extensibility across domains. Intent i is a model of the form $P_i(y|\mathbf{x})$, where $\mathbf{x}$ are the words in the utterance, and y is a binary variable where $y = 1$ indicates that the intent is present in the utterance and $y = 0$ indicates not. For a given utterance $\mathbf{x}$ and a set of intents $\mathscr{I}$, the most likely intent i^* can be selected as:

$$i^* = \arg\max_{i \in \mathscr{I}} P_i(y = 1|\mathbf{x}) \tag{1}$$

Out of domain utterances $i = \varnothing$ — i.e., those which match none of the intent detectors — can be explicitly modeled with a background model $P_\varnothing(y = 1|\mathbf{x})$.[1]

The model itself can be estimated in a variety of ways, such as boosting [9], support vector machines [5], or deep neural networks [8] among others — and the focus of much past work has been to maximize performance (accuracy, F-measure,

[1] This approach assumes that the scores are directly comparable. In this paper, the classifiers are not guaranteed to produce comparable scores, but since only a handful of classifiers are used and their calibration is similar enough, this mis-match will not be a practical problem. We'll return to this point in the conclusion.

etc.) given a fixed dataset. The approach described in this paper admits any model class which can be trained rapidly, but for simplicity we have used regularized log-linear models. Features will be words, n-grams of words, or other simple lexical features such as the length of the utterance.

The approach in this paper focuses on maximizing performance for a given *time budget*, where time can be spent on labeling or feature engineering. The primary time-budget approach taken in past work has been *active learning* [15, 13]. Active learning starts with a small seed set of labeled data instances, from which an initial classifier is trained. This classifier then scores a large set of unlabeled instances. A selection rule then draws instances based on their scores. For example, the rule might draw examples for which the classifier shows greatest uncertainty — e.g., a score of 0.5. As more instances are labeled, the classifier is re-trained, and the process is repeated. As compared to random sampling, active learning has been shown to have much better time efficiency — i.e., active learning requires fewer labels than random sampling to attain the same level of performance. By contrast, in this paper, we apply *interactive learning*.

Enabling non-experts to quickly build data-driven dialog systems is a long-standing goal in the research literature [4, 7, 3]. Unlike past efforts, this work draws on a massive set of utterances from a deployed commercial personal assistant, and builds upon interactive learning, described next.

3 Interactive learning

Interactive learning (IL) is a method for efficiently building classification models, where classifier definition, labeling, model building, and evaluation are all interleaved and done by a single developer [10]. Like active learning, IL is suitable when unlabeled data is abundant but labeling is expensive, as in our case. IL incorporates active learning but extends it substantially.

IL requires a large database of unlabeled data instances, such as webpages, emails, or (in our case) text of utterances to a personal assistant. The database must contain positive instances, although these instances may be very rare. Personal assistant logs often do contain utterances expressing intents which aren't currently implemented, because the functional scope of commercially-available personal assistants is continually growing and thus not well understood by all users. This allows intent detectors to be built in advance of functional implementation, at least for intents expressed at the first turn (logs are unlikely to contain subsequent utterances for functionalities that don't yet exist).

A developer begins with a general idea of the classifier they want to build. In our case these will be binary classifiers — in our case, detecting utterances which correspond to a particular intent. For IL we use a tool created in Microsoft Research called ICE, which stands for Interactive Classification and Extraction, and is shown in Figure 2.

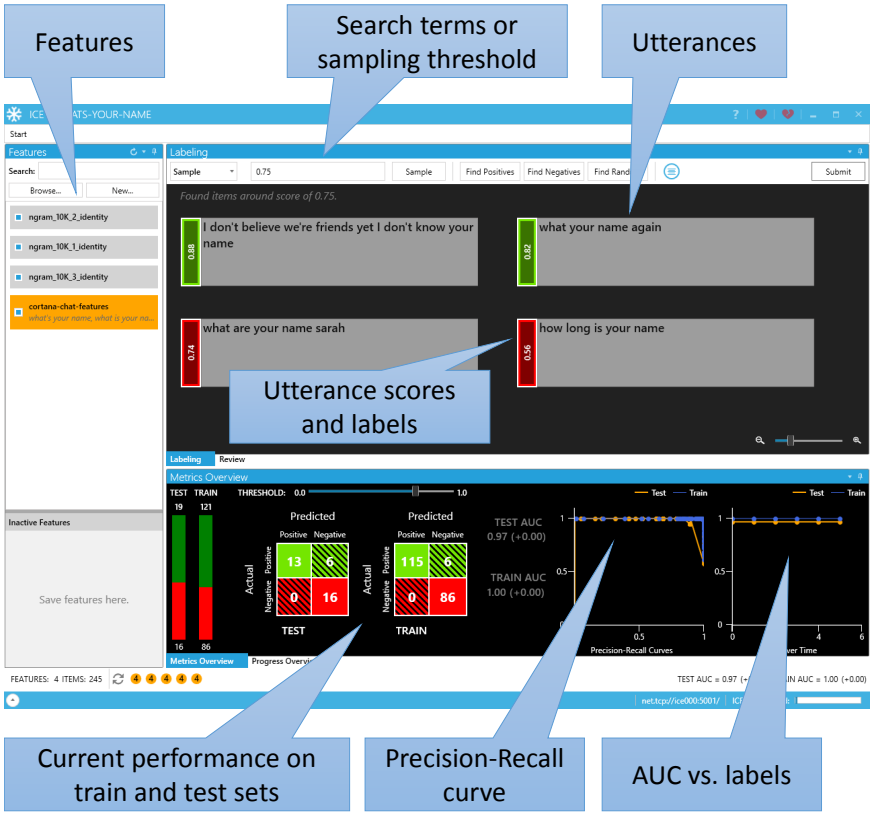

Fig. 2 ICE (Interactive Classification and Extraction): Tool at Microsoft Research for interactive learning [10]. In the top center, the developer can enter a search term or a score from which to sample. Utterances are shown in the central panel. Their scores under the current model are shown to the left of each utterance. The red (False) and green (True) bars indicate their suggested label by the model, which can be changed by clicking. The performance of the model is shown in the bottom panel, including confusion matrices, precision-recall curves, and area under the curve (AUC) as a function of how many labels have been entered. Features can be added and edited on the left.

The developer starts working by searching for data instances using textual search terms based on their domain knowledge. For example, they might issue searches like “calendar” or “my schedule”. The searches yield results which the developer then labels. Labels can be positive, negatives, or a special “don’t know” label.

After each label is entered, all the labels are randomly divided between a training and test set, and a model is built on the training set, excluding the “don’t know” labels. This model is then used in 3 ways. First, all of the instances labeled so far can be displayed graphically, showing the distribution of scores, giving an overview of performance. Second, when the developer searches for new instances, the model is used to propose labels. This accelerates labeling and also gives an indication of the

performance of the model on unseen data. Third, each time the model is rebuilt, it is applied to all of the unlabeled data, which allows the developer to draw samples of unlabeled instances at a particular score. Setting a value near 0.5 will draw instances most perplexing to the classifier, similar to active learning. Setting a high or low value searches for false-positive or false-negatives.

Past work has shown that domain experts (here, developers) can readily provide additional words that can be used as features [11], and that those words improve machine-learning performance [12]. Therefore, in ICE, the developer can populate a list of individual words or phrases which the developer believes will be important in the domain, like "my calendar", "am i free", "appointment", etc. Finally, the developer can provide *classes* that pertain to the domain, such as days of the week: Monday, Tuesday, etc. After a feature is edited (by adding or removing a phrase), enabled, or disabled, the model is re-built in a few seconds, and the train and test sets are re-scored. This allows the developer to experiment with different word-based features and immediately see the effects. The developer can also opt to use all observed words/n-grams as features.

As labeling progresses, the developer moves freely between evaluating and improving the model. Evaluation is done by looking at the distribution of scores on labeled instances and the scores assigned to new instances. Improvement is done by adding more labels or editing the features. In addition, in response to the data, the developer may decide to alter the definition of the classifier — i.e., the labeling guidelines they are following — and revise labels accordingly. For example, the developer may decide that "Show me my calendar" should be a different intent than "Am I free at 3 PM?" when previously they'd been grouped together.

To our knowledge, this work is the first to apply interactive learning to the task of building intent detectors for dialog systems. As compared to the traditional approach which requires about half a dozen staff, IL requires just one person, and that person need not be an expert in machine learning — therefore our hypothesis is that IL will result in substantial time savings over the traditional approach. Active learning addresses the labeling step in the traditional approach — and has been shown to reduce effort at that step — but still requires the same number of roles as the traditional approach.

4 Building intent detectors with interactive learning

As a first test of applying interactive learning to intent detection, we loaded 25M raw utterances from Microsoft Cortana into our IL tool. For typed utterances, the log contained the text entered; for spoken utterances, the log contained the output of the (possibly erroneous) speech recognizer. Utterances likely to contain personal or identifying information were excluded.

We then applied ICE to build three intent detectors in the movies domain:

- MOVIESDIRECTEDBY: The user is requesting to find all movies directed by a named person, for example "What movies did Stanley Kubrick direct?"

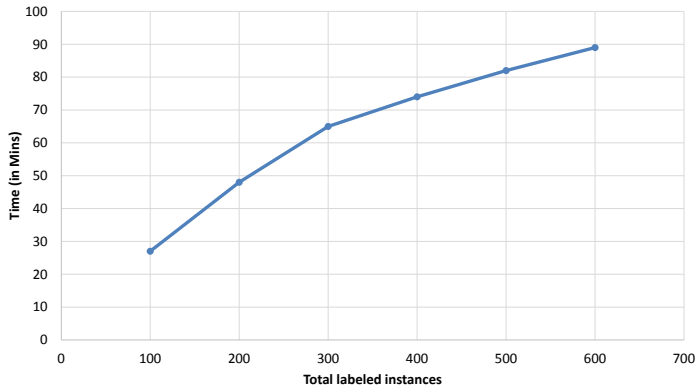

Fig. 3 Cumulative effort (time) for building the MOVIERUNTIME intent with interactive learning.

- WHODIRECTED: The user is requesting the name of a director for a named movie, for example "Who directed The Matrix?"
- MOVIERUNTIME: The user is requesting the duration of a named movie, for example "How long is Gone with the Wind?"

The first two intents were built to gain familiarity with the IL tool and as a result the effort duration was not carefully logged. The last intent was built under controlled, carefully logged conditions. The developer added bag-of-n-gram features, specific n-grams like "director", "movie", and "who directed", and also a class containing all movies names found in Freebase.[2]

The effort expended (in minutes) is shown in Figure 3. For MOVIERUNTIME, labeling 600 utterances required 90 minutes. Note that the marginal time per utterance declines sharply: the first 100 utterances required 28 minutes, whereas the last 100 utterances required 9 minutes. This illustrates the benefits of interactive learning: early in labeling, the developer is manually searching for utterances to label, the model is unable to suggest labels, and more feature engineering is required; later in labeling, the model can be used to select utterances to label and can often propose accurate labels, and the features are stable so little feature engineering is required.

We then evaluated the performance of all three intent detectors on held-out data. Precision is straightforward to evaluate: the models were run on randomly ordered unseen utterances; the first 150 utterances scored above a threshold were manually labeled. Results are shown in Table 1. The precision ranged from 81% to 93% for the three intents developed.[3]

Unlike precision, recall is infeasible to measure, since each intent appears very infrequently in the data, so accurately estimating recall would require labeling

[2] www.freebase.com

[3] The held-out test set excluded utterances which appeared in the training set, whereas in actual deployment, utterances in the training set may re-appear. Therefore, these are conservative estimates which could underestimate performance.

Table 1 Effort and precision for three binary intent classifiers

Intent	Effort	Number of labels	Test set size	Precision
MOVIESDIRECTEDBY	180 minutes*	1121	150	93%
WHODIRECTED	180 minutes*	1173	150	89%
MOVIERUNTIME	90 minutes	600	150	81%

* Estimate

100Ks or millions of utterances. As a basic check, 150 unseen utterances were chosen at random, were scored using the MOVIERUNTIME intent detector, and manually labeled. None were found to contain the MOVIERUNTIME intent, and the model scored all below the threshold.

The false-positives for the MOVIERUNTIME intent were examined by hand. The main cause of errors was that the n-gram "how long is" refers to many types of queries, *and* many words in English are titles of movies, making some instances difficult to classify. For example, in the utterance "How long is Burger King open", the words "Burger" and "King" are both titles of movies, but "Burger King" is not related to movies in this context. This was one consideration in developing a second tool to explore and correct end-to-end operation, described next.

5 Improvement through end-to-end testing

The results in the previous section showed that it is possible to use interactive learning to quickly build a single intent detector with good precision, so we next set about building an end-to-end tool for testing and system improvement. Labeling and correcting end-to-end interactions allows developers to view and debug interactions as they will be experienced by users — i.e., developers can decide on intermediate labels for stages in the pipeline based on the response to the user. The design of the tool is to visualize the end-to-end processing done for an utterance. If the developer sees any errors in the pipeline, they can correct the output of any processing stage, and the remainder of the pipeline is immediately re-run using those corrections as the revised input. Once the whole pipeline has been checked and the correct answer is output at the end of the pipeline, the developer can save the labels, which stores labels for every component of the pipeline. The end-to-end tool also allows the developer to type in an utterance, which handles the case where variations of the target intent don't (yet) appear in the logs. This enables the developer to bootstrap intent models when sufficient example utterances do not yet exist. A screenshot of our tool is shown in Figure 4. In the upper left, the developer types in an utterance. The utterance is then scored by all of the available intent detectors — in our case, three — then performs entity identification, and calls a function which produces a response.

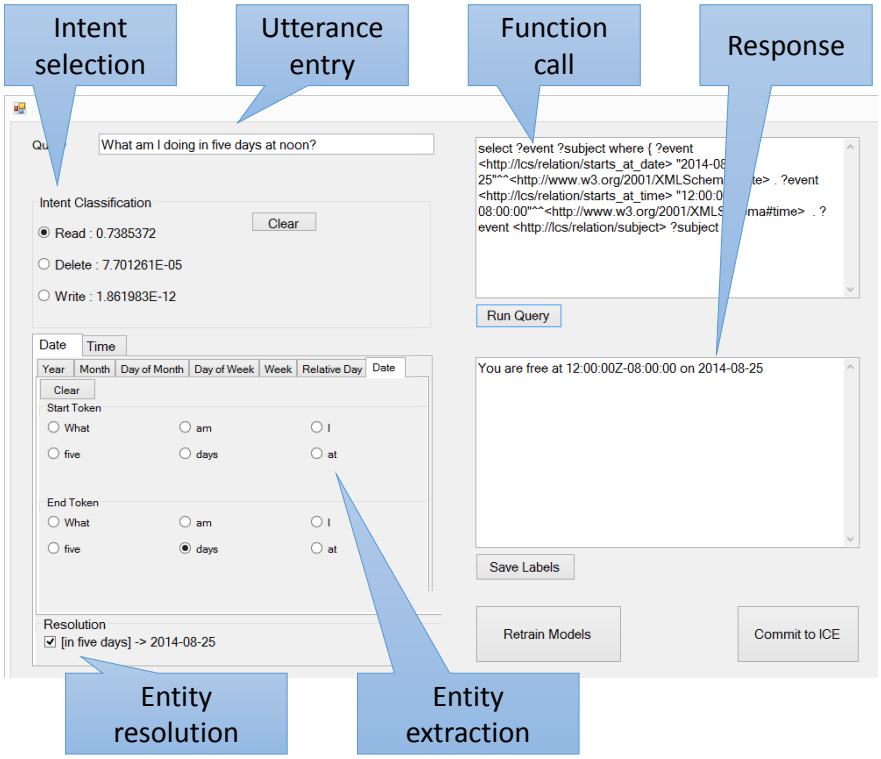

Fig. 4 End-to-end labeling tool. The developer enters an utterance in the upper left. The intent detectors and their scores are shown below that. The bottom left shows entity extraction and resolution. Finally, the resulting function call is shown in the upper right, and the system response is shown at middle right. Errors detected at any stage of the pipeline can be corrected and are propagated forward; once the whole pipeline produces the correct answer, the developer clicks *Save Labels*, which also clears the display so the developer can enter another utterance. The developer can also *Retrain models* and *Commit to ICE* which stores the new models and labels in the cloud.

In this first experiment, we explored the rather limited question of whether it was possible to improve a single intent model using this end-to-end tool. We began with a working calendar dialog system with 2 intents: READFROMCALENDAR and WRITETOCALENDAR. We then asked a developer to create a new binary intent detector, DELETEFROMCALENDAR, in the ICE IL tool (not in the end-to-end tool). The devleoper labeled 470 instances in 60 minutes. On a held-out test set of 3M unlabeled utterances, using a decision threshold of 0.5, the model retrieved 114 utterances, of which 88% were correct — i.e., the precision was 88%. In the results below, this model is denoted "IL".

Next, the developer began using the end-to-end tool, with all 3 intents and 2 entity extractors. The developer examined 33 utterances in 30 minutes, and the labels collected were used to update all of the intent and entity models, including the

Table 2 Precision and utterances retrieved for DELETEFROMCALENDAR intent on a test set of 3M unlabeled utterances.

Method	Total effort	Number of labels	Precision	Utterances retrieved
IL	60 minutes	470	88%	114
IL+E2E	90 minutes	503	88%	134

DELETEFROMCALENDAR intent model. The updated model (denoted "IL+E2E") for DELETEFROMCALENDAR was run on the same held-out data used with the baseline (IL) model as above; we adjusted the threshold so that it achieved the same precision as the baseline (IL) model developed in isolation (88%). At this precision, the IL+E2E model retrieved 134 sentences — an increase of 18% compared to the baseline model, which was statistically significant at $p = 0.004$ using McNemar's test. These results are summarized in Table 2.

This analysis shows that the recall of the model increased for a fixed precision, which in turns means that the F-measure increased. As above, the intent is very rare in the data, so calculation of exact values for recall (and thus F-measure) are not possible.

While it is clear that the end-to-end tool yielded an increase in performance of the intent models, in future work we'd like to evaluate the time efficiency compared to the ICE IL tool in Sections 3 and 4. It is clear that the number of utterances per unit time in the end-to-end tool (33 utterances in 30 minutes) is lower than in the ICE IL tool (470 utterances in 60 minutes); however we note that each labeled utterance in the end-to-end tool produces labels for *every* intent model, entity extractor, and entity resolver. Therefore it is likely that this end-to-end evaluation also improved these models. We defer further evaluations to future work; rather, in the next section, we continue investigating the performance of a single intent detector in a live, public, large-scale personal assistant dialog system.

6 Interactive learning in production

In this section we report on preliminary results applying this technique to the live Cortana service. Here we apply IL to the social conversation domain — i.e., social utterances directed at the agent, such as "Cortana, where are you from?" or "Are you married?". The designer of this feature — who was not a machine learning expert — developed a binary detector for the COMPLIMENT intent, including utterances like "Cortana you're great" or "That was pretty clever".

The designer spent 135 minutes developing this classifier, labeling 2,254 utterances, approximately half positive and half negative. This classifier was then deployed into the production Cortana platform. A random sample of 144K utterances from the production platform was then collected. The compliments classifier fired on 1,160 of these. These 1,160 utterances were manually labeled. Precision was mea-

sured to be between 97.7% and 99.9%, with the variation due to how ambiguous utterances like "that's great" and "very nice" are counted, since these are compliments in some, but not all, contexts. As above, an exact recall measurement is not possible, but in a random sample of 512 utterances, 1 or 2 were compliments (one was ambiguous). This implies an occurrence rate in the data of 0.2% – 0.4%; the upper bound of the 95% confidence interval of the proportion $\frac{1}{512}$ is 1.1% and $\frac{2}{512}$ is 1.4%. The classifier fired on 0.8% of the utterances. These figures cannot be used to compute recall but do suggest that the recall is reasonable, since the fraction of utterances where the (high-precision) classifier fired is about equal to the occurrence rate in a small sample, and not far from the upper limit of the 95% confidence interval.

By contrast, labeling $2,254$ utterances selected uniformly at random (rather than selected using the IL tool) is highly unlikely to produce a good model. Assuming 0.5% of utterances are compliments, then a uniform random sample would result in $0.005 \cdot 2,254 = 11$ positive labels. Learning a good classifier from 11 positive labels and $2,243$ negative labels would be hopeless.

7 Conclusions

Quickly building intent detectors is a major bottleneck for expanding the functionalities of a personal assistant dialog system. In this paper, we have shown how interactive learning can be applied to this problem. Intent detectors have been built in three domains: movies, calendar, and social conversation. Each binary intent detector required 1-3 hours to build, and yielded good precision, in the range of 81% to 99%, without any obvious problems in recall.

In future work, we will tackle the problem of ensuring comparability of binary classifier scores. Because classifiers produced with interactive learning (or active learning) do not use a training set randomly sampled from the data, their scores are all calibrated differently and are thus not guaranteed to be directly comparable. With a handful of intents, this has not been a practical problem, but in the future there could be 1000s or more binary classifiers running in parallel, with classifiers being added, removed, or changed at any time. The problem of combining many binary classifiers is well-studied in the machine learning literature and numerous solutions exist [1, 2]; what remains to be done is evaluate their applicability to this problem.

In future work we will also consider *structured intents*, where intents can be composed of multiple relations like "Show movies directed by X and starring Y" [6]. We anticipate binary detectors can be used to detect each relation; the open question is how to compose the detected relations together into a structured intent.

Even so, this paper has illustrated the impact of interactive learning for intent detection. The conventional process for labeling data and building a model has been reduced from weeks to hours, while achieving very high precision. In the process, the number of staff required has been reduced from a half dozen or so to one, and that individual does not require machine learning expertise. We anticipate both of

these contributions will help efforts to grow the functionality of personal assistant dialog systems.

Acknowledgements

Thanks to Puneet Agrawal for assistance with the Cortana service, and to Meg Mitchell, Lihong Li, Sheeraz Ahmad, Andrey Kolobov, and Saleema Amershi for helpful discussions.

References

1. Allwein, E.L., Schapire, R.E., Singer, Y.: Reducing multiclass to binary: A unifying approach for margin classifiers. The Journal of Machine Learning Research **1**, 113–141 (2001)
2. Beygelzimer, A., Langford, J., Ravikumar, P.: Error-correcting tournaments. In: Algorithmic Learning Theory, pp. 247–262. Springer (2009)
3. Fukubayashi, Y., Komatani, K., Nakano, M., Funakoshi, K., Tsujino, H., Ogata, T., Okuno, H.G.: Rapid prototyping of robust language understanding modules for spoken dialogue systems. In: Proc IJCNLP (2008)
4. Glass, J.R., Weinstein, E.: Speechbuilder: facilitating spoken dialogue system development. In: Proc Eurospeech, Aalborg, Denmark (2001)
5. Haffner, P., Tur, G., Wright, J.H.: Optimizing SVMs for complex call classification. In: Proc Intl Conf on Acoustics, Speech, and Signal Processing (ICASSP), Hong Kong (2003)
6. Heck, L., Hakkani-Tur, D., Tur, G.: Leveraging knowledge graphs for web-scale unsupervised semantic parsing. In: Proc Interspeech, Lyon, France (2013)
7. Jung, S., Lee, C., Kim, S., Lee, G.G.: Dialogstudio: A workbench for data-driven spoken dialog system development and management. Speech Communication **50**, 697–715 (2008)
8. Sarikaya, R., Hinton, G., Ramabhadran, B.: Deep belief nets for natural language call-routing. In: Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on, pp. 5680–5683 (2011)
9. Schapire, R., Singer, Y.: Boostexter: A boosting-based system for text categorization. Machine Learning **39**(2-3), 135–168 (2000)
10. Simard, P., Chickering, D., Lakshmiratan, A., Charles, D., Bottou, L., Suarez, C.G.J., Grangier, D., Amershi, S., Verwey, J., Suh, J.: ICE: Enabling non-experts to build models interactively for large-scale lopsided problems (2014). URL http://arxiv.org/ftp/arxiv/papers/1409/1409.4814.pdf
11. Stumpf, S., Rajaram, V., et al, L.L.: Toward harnessing user feedback for machine learning. In: Proc IUI (2007)
12. Stumpf, S., Rajaram, V., et al, L.L.: Interacting meaningfully with machine learning systems: Three experiments. IJHCI **67**(8), 639–662 (2009)
13. Tur, G., Hakkani-Tur, D., Schapire, R.E.: Combining active and semi-supervised learning for spoken language understanding. Speech Communication **45**(2), 171 – 186 (2005)
14. Tur, G., Mori, R.D.: Spoken Language Understanding — Systems for Extracting Semantic Information from Speech. John Wiley and Sons (2011)
15. Tur, G., Schapire, R., Hakkani-Tur, D.: Active learning for spoken language understanding. In: Proc Intl Conf on Acoustics, Speech, and Signal Processing (ICASSP), Hong Kong, vol. 1, pp. I–276–I–279 vol.1 (2003)
16. Wang, Y.Y., Deng, L., Acero, A.: Spoken language understanding. Signal Processing Magazine, IEEE **22**(5), 16–31 (2005)