Web-style ranking and SLU combination for dialog state tracking

Jason D. Williams
Microsoft Research, Redmond, WA, USA
jason.williams@microsoft.com

Abstract

In spoken dialog systems, statistical *state tracking* aims to improve robustness to speech recognition errors by tracking a posterior distribution over hidden dialog states. This paper introduces two novel methods for this task. First, we explain how state tracking is structurally similar to *web-style ranking*, enabling mature, powerful ranking algorithms to be applied. Second, we show how to use multiple spoken language understanding engines (SLUs) in state tracking — multiple SLUs can expand the set of dialog states being tracked, and give more information about each, thereby increasing both recall and precision of state tracking. We evaluate on the second Dialog State Tracking Challenge; together these two techniques yield highest accuracy in 2 of 3 tasks, including the most difficult and general task.

1 Introduction

Spoken dialog systems interact with users via natural language to help them achieve a goal. As the interaction progresses, the dialog manager maintains a representation of the state of the dialog in a process called *dialog state tracking* (Williams et al., 2013; Henderson et al., 2014). For example, in a restaurant search application, the dialog state might indicate that the user is looking for an inexpensive restaurant in the center of town. Dialog state tracking is difficult because errors in automatic speech recognition (ASR) and spoken language understanding (SLU) are common, and can cause the system to misunderstand the user's needs. At the same time, state tracking is crucial because the system relies on the estimated dialog state to choose actions – for example, which restaurants to present to the user.

Historically, commercial systems have used hand-crafted rules for state tracking, selecting the SLU result with the highest confidence score observed so far, and discarding alternatives. In contrast, statistical approaches compute a posterior *distribution* over many *hypotheses* for the dialog state, and in general these have been shown to be superior (Horvitz and Paek, 1999; Williams and Young, 2007; Young et al., 2009; Thomson and Young, 2010; Bohus and Rudnicky, 2006; Metallinou et al., 2013; Williams et al., 2013).

This paper makes two contributions to the task of statistical dialog state tracking. First, we show how to cast dialog state tracking as *web-style ranking*. Each dialog state can be viewed as a document, and each dialog turn can be viewed as a search instance. The benefit of this construction is that it enables a rich literature of powerful ranking algorithms to be applied. For example, the ranker we apply constructs a forest of decision trees, which — unlike existing work — automatically encodes conjunctions of low-level features. Conjunctions are attractive in dialog state tracking where relationships exist between low-level concepts like grounding and confidence score.

The second contribution is to incorporate the output of *multiple* spoken language understanding engines (SLUs) into dialog state tracking. Using more than one SLU can increase the number of dialog states being tracked, improving the chances of discovering the correct one. Moreover, additional SLUs supply more features, such as semantic confidence scores, improving accuracy.

This paper is organized as follows. First, section 2 states the problem formally and covers related work. Section 3 then lays out the data, features, and experimental design. Section 4 applies web-style ranking, and section 5 covers the usage of multiple SLUs. Section 6 extends the types of tracking tasks, section 7 compares performance to other entries in DSTC2, and section 8 briefly concludes.

2 Background

Statistical dialog state tracking can be formalized as follows. At each turn in the dialog, the state tracker maintains a set X of dialog state hypotheses $X = \{x_1, x_2, \ldots, x_N\}$. Each state hypothesis corresponds to a possible true state of the dialog. The posterior of a state x_i at a certain turn in the dialog is denoted $P(x_i)$.

Based on this posterior, the system takes an action a, the user provides an *utterance* in reply, and an automatic speech recognizer (ASR) converts the user's utterance into words. Since speech recognition is an error prone process, the speech recognizer outputs weighted alternatives, for example an N-best list or a word-confusion network. A *spoken language understanding* engine (SLU) then converts the ASR output into a meaning representation U for the user's utterance, where U can contain alternatives for the user's meaning, $U = \{u_1, \ldots, u_L\}$.

The state tracker then updates its internal state. This is done in three stages. First, a hand-written function G ingests the system's last action s, the meaning representation U, and the current set of states X, and yields a new set of possible states, $X' = G(s, U, X)$, where we denote the members of X' as $\{x'_1, x'_2, \ldots, x'_{N'}\}$. The number of elements in X' may be different than X, and typically the number of states increases as the dialog progresses, i.e. $N' > N$. In this work, G simply takes the Cartesian product of X and U. Second, for each new state hypothesis x'_i, a vector of J features is extracted, $\phi(x'_i) = [\phi_1(x'_i), \ldots, \phi_J(x'_i)]$. In the third stage, a scoring process takes all of the features for all of the new dialog states and scores them to produce the new distribution over dialog states, $P'(x'_i)$. This new distribution is used to choose another system action, and the whole process repeats.

Most early work cast dialog state tracking as a generative model in which hidden user goals generate observations in the form of SLU hypotheses (Horvitz and Paek, 1999; Williams and Young, 2007; Young et al., 2009; Thomson and Young, 2010). More recently, discriminatively trained direct models have been applied, and two studies on dialog data from two publicly deployed dialog systems suggest direct models yield better performance (Williams, 2012; Zilka et al., 2013). The methods introduced in this paper also use discriminative techniques.

One of the first approaches to direct models for dialog state tracking was to consider a small, fixed number of states and then apply a multinomial classifier (Bohus and Rudnicky, 2006). Since a multinomial classifier can make effective use of more features than a generative model, this approach improves precision, but can decrease recall by only considering a small number of states (e.g. 5 states). Another discriminative approach is to score each state using a binary model, then somehow combine the binary scores to form a distribution – see, for example (Henderson et al., 2013b) which used a binary neural network. This approach scales to many states, but unlike a multinomial classifier, each binary classifier isn't aware of its competitors, reducing accuracy. Also, when training a binary model in the conventional way, the training criteria is mis-matched, since the classifier is trained per hypothesis per timestep, but is evaluated only once per timestep.

Maximum entropy (*maxent*) models have been proposed which provide the strengths of both of these approaches (Metallinou et al., 2013). The probability of a dialog hypothesis x_i being correct ($y = i$) is computed as:

$$P(y = i | X, \lambda) = \frac{\exp(\sum_{j \in J} \lambda_j \phi_j(x_i))}{\sum_{x \in X} \exp(\sum_{j \in J} \lambda_j \phi_j(x))}. \quad (1)$$

Maximum entropy models yielded top performance in the first dialog state tracking challenge (Lee and Eskenazi, 2013). In this paper, we use maxent models as a baseline.

A key limitation with linear (and log-linear) models such as maximum entropy models is that they do not automatically build *conjunctions* of features. Conjunctions express conditional combinations of features such as whether the system attempted to confirm x *and* if "yes" was recognized *and* if the confidence score of "yes" is high. Conjunctions are important in dialog state tracking because they are often more discriminative than individual features. Moreover, in linear models for dialog state tracking, one weight is learned *per feature* (equation 1) (Metallinou et al., 2013). As a result, if a feature takes the same value for every dialog hypothesis at a given timestep, its contribution to every hypothesis will be the same, and it will therefore have no effect on the ranking. For example, features describing the current system action are identical for all state hypotheses. Concretely, if $\phi_j(x_i) = c$ for all i, then changing c causes no change in $P(y = i|X, \lambda)$ for all i.

Past work has shown that conjunctions improve dialog state tracking (Metallinou et al., 2013; Lee, 2013). However, past work has added conjunction *by hand*, and this doesn't scale: the number of possible conjunctions increases exponentially in the number of terms in the conjunction, and it's difficult to predict in advance which conjunctions will be useful. This paper introduces algorithms from web-style ranking as a mechanism for automatically building feature conjunctions.

In this paper we also use *score averaging*, a well-known machine learning technique for combining the output of several models, where each output class takes the average score assigned by all the models. Under certain assumptions — most importantly that errors are made independently — score averaging is guaranteed to exceed the performance of the best single model. Score averaging has been applied to dialog state tracking in previous work (Lee and Eskenazi, 2013). Here we use score averaging to maximize data use in cascaded models, and as a hedge against unlucky parameter settings.

3 Preliminaries

In this paper we use data and evaluation metrics from the second dialog state tracking challenge (DSTC2) (Henderson et al., 2014; Henderson et al., 2013a). Dialogs in DSTC2 are in the restaurant search domain. Users can search for restaurants in multiple ways, including via constraints, or by name. The system can offer restaurants that match, confirm user input, ask for additional constraints, etc.

There are three components to the hidden dialog state: user's **goal**, search **method**, and **requested slots**. The user's **goal** specifies the user's search constraints, and consists of 4 *slots*: area, pricerange, foodtype, and name. The number of values for the slots ranges from 4 to 113.[1] In DSTC2, trackers output scored lists for each slot, and also a scored list of joint hypotheses. For example, at a given timestep in a given dialog, three joint goal hypothesis might be (area=west,food=italian), (area=west), and (), where () means the user hasn't specified any constraints yet. Since tracking the joint user goal is the most general and most difficult task, we'll focus on this first, and return to the other tasks in section 6.

3.1 User goal features

For features, we broadly follow past work (Lee and Eskenazi, 2013; Lee, 2013; Metallinou et al., 2013). For a hypothesis x_i, *for each slot* the features encode 253 low-level quantities, such as: whether the slot value appears in this hypothesis; how many times the slot value has been observed; whether the slot value has been observed in this turn; functions of recognition metrics such as confidence score and position on N-best list; goal priors and confusion probabilities estimated on training data (Williams, 2012; Metallinou et al., 2013); results of confirmation attempts ("Italian food, is that right?"); output of the four rule-based baseline trackers; and the system act and its relation to the goal's slot value (e.g., whether the system act mentions this slot value).

Of these 253 features for each slot, 119 are the same for all values of that slot in a given turn, such as which system acts were observed in this turn. For these, we add 238 *conjunctions* with slot-specific features like confidence score, which makes these features useful to our maxent baseline. This results in a total of $253+238 = 491$ features per slot. The features for each of the 4 slots are concatenated together to yield $491 * 4 = 1964$ features per joint hypothesis.

3.2 Evaluation metrics

In DSTC2, there are 3 primary metrics for evaluation — accuracy of the top-scored hypothesis, the L2 probability quality, and an ROC measurement. The ROC measurement is only meaningful when compared across systems with similar accuracy; since our variants differ in accuracy, we omit ROC. However, note that *all* of the metrics, including ROC, for our final entries on the development set and test set are available for public download from the DSTC2 website.[2]

The DSTC2 corpus consists of three partitions: train, development, and test. Throughout sections 4-6, we report accuracy by training on the training set, and report accuracy on the development set and test set. The development set was available during development of the models, whereas the test set was not.

[1]Including a special "don't care" value.

[2]`camdial.org/~mh521/dstc/`

3.3 Baselines

We first compare to the four rule-based trackers provided by DSTC2. These were carefully designed by other research groups, and earlier versions of them scored very well in the first DSTC (Wang and Lemon, 2013). In each column in Tables 2 and 3, we report the best result from any rule-based tracker. We also compare to a maxent model as in Eq 1. Our implementation includes L1 and L2 regularization which was automatically tuned via cross-validation.

4 Web-style ranking

The ranking task is to order a set of N documents by relevance given a query. The input to a ranker is a query Q and set of documents $X = \{D_1, \ldots, D_N\}$, where each document is described in terms of features of that document and the query $\boldsymbol{\phi}(D_i, Q)$. The output is a score for each document, where the highest score indicates the most relevant document. The overall objective is to order the documents by *relevance*, given the query. Training data indicates the relevance of example query/document pairs. Training labels are provided by judges, and relevance is typically described in terms of several levels, such as "excellent", "good", "fair", and "not relevant".

The application of ranking to dialog state tracking is straightforward: instead of ranking features of documents and queries $\boldsymbol{\phi}(D_i, Q)$, we rank features of dialog states $\boldsymbol{\phi}(X_i)$. For labeling, the correct dialog state is "relevant" and all other states are "not relevant".

Like dialog state tracking, ranking tasks often have features which are constant over all documents – particularly features of the query. This is one reason why ranking algorithms have incorporated methods for automatically building conjunctions. The specific algorithm we use here is lambdaMART (Wu et al., 2010; Burges, 2010). LambdaMART is a mature, scalable ranking algorithm: it has underpinned the winning entry in a community ranking challenge task (Chapelle and Chang, 2011), and is the foundation of the ranker in the Bing search engine. LambdaMART constructs a forest of M decision trees, where each tree consists of binary branches on features, and the leaf nodes are real values. Each binary branch specifies a threshold to apply to a single feature. For a forest of M trees, the score of a dialog state x is

$$F(x) = \sum_{m=1}^{M} \alpha_m f_m(x) \tag{2}$$

where α_m is the weight of tree m and $f_m(x)$ is the value of the leaf node obtained by evaluating decision tree m by features $[\phi_1(x), \ldots, \phi_J(x)]$. The training objective is to maximize ranking quality, which here means one-best accuracy. The decision trees are learned by regularized gradient descent, where trees are added successively to improve ranking quality – in our case, to maximize how often the correct dialog state is ranked first. The number of trees to create and the number of leaves per tree are tuning parameters. Through cross-validation, we found that 500 decision trees each with 32 leaves were the best settings. We use the same set of 1964 features for lambdaMART as was used for the maxent baseline.

Results are shown in row 3 of table 2 under "Joint goal". Ranking outperforms both baselines on both the development and training set. This result illustrates that automatically-constructed conjunctions do indeed improve accuracy in dialog state tracking. An example of a single tree computed by lambdaMART is shown in Appendix A. The complexity of this tree suggests that human designers would find it difficult to specify a tractable set of good conjunction features.

5 Multiple SLU engines

As described in the introduction, dialog state tracking typically proceeds in three stages: enumeration of the set of dialog states to score, feature extraction, and scoring. Incorporating the output of multiple SLUs requires changing the first two steps. Continuing with notation from section 2, with a single SLU output U, the enumeration step is $X' = G(s, U, X)$ — recall that U is a *set* of SLU hypotheses from an SLU engine. With multiple SLU engines we have K SLU outputs $U_1, \ldots, U_K$, and the enumeration step is thus $X' = G(s, U_1, \ldots, U_K, X)$. In our implementation, we simply take the union of all concepts on all SLU N-best lists and enumerate states as in the single SLU case – i.e., the Cartesian product of dialog states X with concepts on the SLU output.

The feature extraction step is modified to output features derived from all of the SLU engines. Concretely, if a feature $\phi_j(x)$ includes information from an SLU engine (such as confidence score or position on the N-best list), it is duplicated K times – i.e., once for each SLU engine. Additional binary features are added to encode whether each SLU engine has output the slot value of this dialog state. This allows for the situation that a slot value is not output by all SLU engines, in which case its confidence score, N-best list position, etc. will not be present from some SLU engines. Using two SLU engines on our data increases the number of features per joint goal from 1964 to 3140.

5.1 SLU Engines

We built two new SLU engines, broadly following (Henderson et al., 2012). Both consist of many binary classifiers. In the first engine **SLU1**, a binary classifier is estimated *for each slot/value pair*, and predicts the presence of that slot/value pair in the utterance. Similarly, a binary classifier is estimated for each user dialog act. Input features are word n-grams from the ASR N-best list. We only considered n-grams which were observed at least c times in the training data; infrequent n-grams were mapped to a special UNK feature. For binary classification we used decision trees, which marginally outperformed logistic regression, SVMs, and deep neural networks. Through cross-validation we set $n = 2$ and $c = 2$ – i.e., uni-grams and bi-grams which appear at least twice in the training data.

At runtime, the top SLU output on the N-best list is formed by taking the most likely combination of all the binary classifiers; the second SLU output is formed by taking the second most likely combination of all the binary classifiers; and so on, where only valid SLU combinations are considered. For example, the "bye" dialog act takes no arguments, so if "bye" and "food=italian" were the most likely combination, this combination would be skipped. Scores are formed by taking the product of all the binary classifiers, with some smoothing.

The second SLU engine **SLU2** is identical except that it also includes features from the word confusion network. Specifically, each word (unigram) appearing in the word confusion network is a feature. Bi-gram confusion network features did not improve performance.

If we train a new SLU engine and a ranker on the same data, this will introduce unwanted bias. Therefore, we divided the training data in half, and use the first half for training the SLU, and the second for training the ranker. Table 1 shows several evaluation metrics for each SLU engine, including the SLU included in the corpus, which we denote **SLU0**. SLU precision, recall, and F-measure are computed on the top hypotheses. Item cross-entropy (ICE) (Thomson et al., 2008) measures the quality of the scores for all the items on the SLU N-best list. Table 1 also shows joint goal accuracy by using SLU0, SLU1, or SLU2, for either a rule-based baseline or the ranking model. Overall, our SLU engines performed better on isolated SLU metrics, but did not yield better state tracking performance when used *instead* of the SLU results in the corpus.

5.2 Results with multiple SLU engines

Table 2, rows 4 and 7 show that an improvement in performance does results from using 2 SLU engines. In rows 4 and 7, the additional SLU engine is trained on the first half of the data, and the ranker is trained on the second half – we call this arrangement **Fold A**. To maximize use of the data, it's possible to train a second SLU/ranker pair by inverting the training data – i.e., train a second SLU on the second half, and a second ranker (using the second SLU) on the first half. We call this arrangement **Fold B**. These two configurations can be combined by running *both* trackers on test data, then averaging their scores. We call this arrangement **Fold AB**. If a hypothesis is output by only one configuration, it is assumed the other configuration output a zero score.

Table 2, rows 5 and 8 show that the fold AB configuration yields an additional performance gain.

5.3 Model averaging

A small further improvement is possible by averaging across multiple models (rankers) with different parameter settings. Since all of the models will be estimated on the same data, this is unlikely to make a large improvement, but it can hedge against an unlucky parameter setting, since the performance after averaging is usually close to the maximum.

To test this, we trained a second pair of ranking models, with a different number of leaves per tree (8 instead of 32). We then applied this second model, and averaged the scores between the two variants. Results are in Table 2, rows 6 and 9. Averaging scores across two parameter settings generally results in performance equal to or better than the maximum of the two models.

	Dev set						Test set					
SLU	SLU Metrics				Goal track. acc.		SLU Metrics				Goal track. acc.	
source	Prec.	Recall	F-meas.	ICE	Rules	Ranking	Prec.	Recall	F-meas.	ICE	Rules	Ranking
SLU0	**0.883**	0.666	0.759	2.185	**0.623**	**0.666**	**0.900**	0.691	0.782	1.955	**0.719**	**0.739**
SLU1	0.818	0.729	0.771	2.189	0.598	0.637	0.846	0.762	0.802	1.943	0.667	0.709
SLU2	0.844	**0.742**	**0.789**	**2.098**	0.605	0.658	0.870	**0.777**	**0.821**	**1.845**	0.685	0.734

Table 1: Performance of three SLU engines. SLU0 is the DSTC2 corpus; SLU1 is our engine with uni-grams and bi-grams of ASR results in the corpus; and SLU2 is SLU1 with the addition of unigram features from the word confusion network. Precision, Recall, F-measure, and ICE evaluate the quality of the SLU output, not state tracking. "ICE" is item-wise cross entropy — smaller numbers are better (Thomson et al., 2008). "Rules" indicates dialog state tracking accuracy for user joint goals by running the rule-based baseline tracker on the indicated SLU (alone); "Ranking" indicates joint goal accuracy of running a ranker trained on the indicated SLU (alone). For training, goal tracking results use the "Fold A" configuration (c.f. Section 5.2).

5.4 Joint goal tracking summary

The overall process used to train the joint goal tracker is summarized in Appendix B. For joint goal tracking, web-style ranking and multiple SLUs both yield improvements in accuracy on the development and test sets, with the improvement associated with multiple SLUs being larger. We also observe that ranking produces relatively poor L2 results. This can be attributed to its training objective, which explicitly maximizes 1-best accuracy without regard to the distribution of the scores. This is in contrast to maxent models which explicitly minimize the L2 loss. We examined the distribution of scores, and qualitatively the ranker is usually placing less mass on its top guess than maxent, and spreading more mass out among other (usually wrong) entries. We return to this in the future work section.

6 Fixed-size state components

DSTC2 consists of three tracking tasks: in addition to the user's goal, the user's search method and which slots they requested to hear were also tracked. These other two tasks were comparatively simpler because their domains are of a small, fixed size. Thus classical machine learning methods can be applied – i.e., ranking is not directly applicable to tracking the method and required slots. However, applying multiple SLU engines is still applicable.

The search **method** specifies how the user wants to search. There are 5 values: *by-constraints* such as area=west,food=italian, *by-name* such as "royal spice", *by-alternatives* as in "do you have any others like that?", *finished* when the user is done as in "thanks goodbye", and *none* when the method can't be determined. At each turn, exactly one of the 5 methods is active, so we view the method component as a standard multinomial classification task. For features, we use the score for each method output by each of the 4 rule-based baselines, and whether each of the methods is available according to the SLU results observed so far. We also take conjunctions for each method with: whether each system dialog act is present in the current turn, or has ever been used, and what slots they mentioned; and whether each slot has appeared in the SLU results from this turn, or any turn. In total there are 640 features for the method classifier (when using one SLU engine).

The **requested slots** are the pieces of information the user wants to hear in that turn. The user can request to hear a restaurant's area, food-type, name, price-range, address, phone number, postcode, and/or signature dish. The user can ask to hear any combination of slots in a turn – e.g., "tell me their address and phone number". Therefore we view each requested slot as a binary classification task, and estimate 8 binary classifiers, one for each requestable slot. Each requested slot takes as features: whether the slot could logically be requested at this turn in the dialog; whether the SLU output contained a "request" act and which slot was requested; the score output by each of the 4 rule-based baselines; whether each system dialog act is present in the current turn, or has ever been used, and what slots they mentioned; and whether each slot has appeared in the SLU results from this turn, or any turn. For each requestable slot's binary classifier, this results in 187 features (with one SLU engine).

For each of these tasks, we applied a maxent

					Joint goal				Search method				Requested slot			
				Model	Dev. set		Test set		Dev. set		Test set		Dev. set		Test set	
Row	SLU	Fold	Model	comb.	Acc.	L2	Acc.	L2	Acc.	L2	Acc.	L2	Acc.	L2	Acc.	L2
1	0	—	rules	N	0.623	0.601	0.719	**0.464**	0.860	0.217	0.897	0.158	0.903	0.155	0.884	0.196
2	0	all	maxent	N	0.649	**0.532**	0.692	0.480	0.890	0.177	0.909	0.143	0.952	0.078	0.967	0.054
3	0	all	*	N	0.666	0.739	0.739	0.721	—	—	—	—	—	—	—	—
4	0+1	A	*	N	0.686	0.770	0.757	0.766	0.912	0.144	0.936	0.104	0.960	0.062	0.976	0.039
5	0+1	AB	*	N	0.697	0.749	0.769	0.748	0.913	0.135	0.938	0.097	0.962	0.060	0.978	0.037
6	0+1	AB	**	Y	0.699	0.766	0.770	0.766	**0.916**	0.135	0.943	0.091	0.964	0.059	0.978	0.036
7	0+2	A	*	N	0.697	0.731	0.765	0.727	0.910	0.146	0.939	0.099	0.966	0.058	0.979	0.037
8	0+2	AB	*	N	**0.711**	0.725	0.778	0.721	0.913	0.133	0.943	0.092	**0.967**	0.058	**0.980**	**0.033**
9	0+2	AB	**	Y	0.710	0.742	**0.781**	0.739	0.915	**0.132**	**0.948**	**0.085**	**0.967**	**0.057**	**0.980**	**0.033**

Table 2: Summary of accuracy and L2 for the three tracking tasks, trained on the "train" set. In rows marked (*), joint goal accuracy used ranking, and the other two tasks used maxent. In rows marked (**), several model classes/parameter settings were used and combined with score averaging.

model; results for this and the best rule-based baseline are in the rows 1 and 2 of Table 2. We tried applying decision trees, but this did not improve performance (not shown) as it did for goal tracking. Note that in the goal tracking task, one weight is learned for each *feature* for any class (goal), whereas in standard multiclass and binary classification, one weight is learned for each *feature,class* pair.[3] Perhaps decision trees were not effective in increasing accuracy for method and requested slots because, compared to joint goal tracking, some conjunctions are implicitly included in linear models.

We then added a second SLU engine in the same manner as for goal tracking. This increased the number of features for the method task from 640 to 840, and from 187 to 217 for each binary requested slot classifier. Results are shown in Table 2; rows 4 and 7 show results with one fold, and rows 5 and 8 show results with both folds. Finally, we considered alternate model forms for each classifier, and then combined them with score averaging. For the method task, we used a second maximum entropy model with different regularization weights, and a multi-class decision tree. For the requested slot binary classifiers, we added a neural network classifier. As above, score averaging across different model classes can yield small gains (rows 6 and 9).

Overall, as with goal tracking, adding a second SLU engine resulted in a substantial increase in accuracy. Unlike goal tracking which used a ranker, the standard classification models used here are explicitly optimized for L2 performance and as a result achieved very good L2 performance.

[3]Plus a constant term per class.

7 Blind evaluation results

When preparing final entries for the DSTC2 blind evaluation, we not longer needed a separate development set, so our final models are trained on the combined training and development sets. In the DSTC2 results, we are team2. Our entry 0 and 1 use the process described above, including score averaging across multiple models. Entry0 used SLU0+1, and entry1 used SLU0+2. Entry3 used a maxent model on SLU0+2, but without model averaging since its parameters are set with cross-validation.[4]

Results are summarized in Table 3. For accuracy for the joint goal and method tasks, our entries had highest accuracy. After the evaluation, we learned that we were the only team to use features from the word confusion network (WCN). Comparing our entry0, which does not use WCN features, to the other teams shows that, given the same input data, our entries were still best for the joint goal and method tasks.

The blind evaluation results give a final opportunity to compare the maxent model with the ranking model: entry1 and entry3 both use SLU0+2, and score an identical set of dialog states using identical features. Joint goal accuracy is better for the ranking model. However, as noted above, L2 performance for the ranking model was substantially worse than for the maxent model.

After the blind evaluation, we realized that we had inadvertently omitted a key feature from the "requested" binary classifiers — whether the "request" dialog act appeared in the SLU results.

[4]The other entries team2.entry2 and team2.entry4 are not described in this paper. In brief, entry2 was based on a recurrent neural network, and entry4 was a combination of entries 1, 2, and 3.

	Goal		Method		Requested		Requested*	
model	Acc.	L2	Acc.	L2	Acc.	L2	Acc.	L2
Best baseline	0.719	0.464	0.897	0.158	0.884	0.196	0.884	0.196
Best DSTC2 result from another team	0.768	**0.346**	0.940	0.095	**0.978**	**0.035**	0.978	0.035
SLU0+1, AB, model comb. (entry0)	0.775	0.758	0.944	0.092	0.954	0.073	0.977	0.037
SLU0+2, AB, model comb. (entry1)	**0.784**	0.735	**0.947**	**0.087**	0.957	0.068	**0.980**	**0.034**
SLU0+2, AB, maxent (entry3)	0.771	0.354	**0.947**	0.093	0.941	0.090	0.979	0.040

Table 3: Final DSTC2 evaluation results, training on the combined "train" and "development" sets. In the results, we are team2. "Model comb." indicates score averaging over several model instances. For the "requested" task, our entry in DSTC2 inadvertently omitted a key feature, which decreased performance significantly. "Requested*" columns indicate results with this feature included. They were computed after the blind evaluation and are not part of the official DSTC2 results.

Therefore table 3 shows results with and without this feature. With the inclusion of this feature, the requested classifiers also achieved best accuracy and L2 scores, although we note that this is not part of the official DSTC2 results. (The results in the preceding sections of this paper included this feature.)

8 Conclusion

This paper has introduced two new methods for dialog state tracking. First, we have shown how to apply web-style ranking for scoring dialog state hypotheses. Ranking is attractive because it can construct a forest of decision trees which compute feature conjunctions, and because it optimizes directly for 1-best accuracy. Second, we have introduced the usage of multiple SLU engines. Using additional SLU engines is attractive because it both adds more possible dialog states to score (increasing recall), and adds features which help to discriminate the best states (increasing precision).

In experiments, using multiple SLU engines improved performance on all three of the tasks in the second dialog state tracking challenge. Maximum entropy models scored best in the previous dialog state tracking challenge; here we showed that web-style ranking improved accuracy over maxent when using either a single or multiple SLU engines. Thus, the two methods introduced here are additive: they each yield gains separately, and further gains in combination.

Comparing to other systems in the DSTC2 evaluation, these two techniques yielded highest accuracy in DSTC2 for 2 of 3 tasks. If we include a feature accidentally omitted from the third task, our methods yield highest accuracy for all three tasks. This experience highlights the importance of the manual task of extracting a set of informative features. Also, ranking improved accuracy, but yielded poor probability quality. For ranking, the L2 performance of ranking was among the worst in DSTC2. By contrast, for the method task, where standard classification could be applied, our entry yielded best L2 performance. The relative importance of L2 vs. accuracy in dialog state tracking is an open question.

In future work, we plan to investigate how to improve the L2 performance of ranking. One approach is to train a maxent model on the output of the ranker. On the test set, this yields an improvement in L2 score from 0.735 to 0.587, and simply clamping ranker's best guess to 1.0 and all others to 0.0 improves L2 to 0.431. This is a start, but not competitive with the best result in DSTC2 of 0.346. Also, techniques which avoid the extraction of manual features altogether would be ideal, particularly in light of experiences here.

Even so, for the difficult and general task of user goal tracking, the techniques here yielded a relative error rate reduction of 23% over the best baseline, and exceeded the accuracy of any other tracker in the second dialog state tracking challenge.

Acknowledgements

Thanks to Dan Bohus for making his maxent software available, to Andrzej Pastusiak for helpful guidance with lambdaMART, and to Geoff Zweig for several helpful conversations.

References

Dan Bohus and Alex Rudnicky. 2006. A 'K hypotheses + other' belief updating model. In *Proc American Association for Artificial Intelligence (AAAI) Workshop on Statistical and Empirical Approaches for Spoken Dialogue Systems, Boston.*

Christopher J.C. Burges. 2010. From ranknet to lambdarank to lambdamart: An overview. Technical Report MSR-TR-2010-82, Microsoft Research.

Olivier Chapelle and Yi Chang. 2011. Yahoo! learning to rank challenge. *JMLR Workshop and Conference Proceedings*, 14:1–24.

Matthew Henderson, Milica Gasic, Blaise Thomson, Pirros Tsiakoulis, Kai Yu, and Steve Young. 2012. Discriminative spoken language understanding using word confusion networks. In *Proc IEEE Workshop on Spoken Language Technologies (SLT), Miami, Florida, USA*.

Matthew Henderson, Blaise Thomson, and Jason D. Williams. 2013a. Handbook for the dialog state tracking challenge 2 & 3. Technical report, Cambridge University.

Matthew Henderson, Blaise Thomson, and Steve Young. 2013b. Deep neural network approach for the dialog state tracking challenge. In *Proceedings of the SIGDIAL 2013 Conference*, pages 467–471, Metz, France, August. Association for Computational Linguistics.

Matthew Henderson, Blaise Thomson, and Jason Williams. 2014. The second dialog state tracking challenge. In *Proceedings of the SIGdial 2014 Conference*, Baltimore, U.S.A., June.

Eric Horvitz and Tim Paek. 1999. A computational architecture for conversation. In *Proc 7th International Conference on User Modeling (UM), Banff, Canada*, pages 201–210.

Sungjin Lee and Maxine Eskenazi. 2013. Recipe for building robust spoken dialog state trackers: Dialog state tracking challenge system description. In *Proceedings of the SIGDIAL 2013 Conference*, pages 414–422, Metz, France, August. Association for Computational Linguistics.

Sungjin Lee. 2013. Structured discriminative model for dialog state tracking. In *Proceedings of the SIGDIAL 2013 Conference*, pages 442–451, Metz, France, August. Association for Computational Linguistics.

Angeliki Metallinou, Dan Bohus, and Jason D. Williams. 2013. Discriminative state tracking for spoken dialog systems. In *Proc Association for Computational Linguistics, Sofia*.

Blaise Thomson and Steve Young. 2010. Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems. *Computer Speech and Language*, 24(4):562–588.

B Thomson, K Yu, M Gasic, S Keizer, F Mairesse, J Schatzmann, and S Young. 2008. Evaluating semantic-level confidence scores with multiple hypotheses. In *Proc Intl Conf on Spoken Language Processing (ICSLP), Brisbane, Australia*.

Zhuoran Wang and Oliver Lemon. 2013. A simple and generic belief tracking mechanism for the dialog state tracking challenge: On the believability of observed information. In *Proceedings of the SIGDIAL 2013 Conference*, pages 423–432, Metz, France, August. Association for Computational Linguistics.

Jason D Williams and Steve Young. 2007. Partially observable Markov decision processes for spoken dialog systems. *Computer Speech and Language*, 21(2):393–422.

Jason D. Williams, Antoine Raux, Deepak Ramachandran, and Alan Black. 2013. The dialog state tracking challenge. In *Proc SIGdial Workshop on Discourse and Dialogue, Metz, France*.

Jason D. Williams. 2012. Challenges and opportunities for state tracking in statistical spoken dialog systems: Results from two public deployments. *IEEE Journal of Selected Topics in Signal Processing, Special Issue on Advances in Spoken Dialogue Systems and Mobile Interface*, 6(8):959–970.

Qiang Wu, Christopher J. C. Burges, Krysta M. Svore, and Jianfeng Gao. 2010. Adapting boosting for information retrieval measures. *Journal of Information Retrieval*, 13(3):254–270.

Steve Young, Milica Gašić, Simon Keizer, François Mairesse, Jost Schatzmann, Blaise Thomson, and Kai Yu. 2009. The hidden information state model: a practical framework for POMDP-based spoken dialogue management. *Computer Speech and Language*, 24(2):150–174.

Lukas Zilka, David Marek, Matej Korvas, and Filip Jurcicek. 2013. Comparison of bayesian discriminative and generative models for dialogue state tracking. In *Proceedings of the SIGDIAL 2013 Conference*, pages 452–456, Metz, France, August. Association for Computational Linguistics.

Appendix A: Example decision tree

Figure 1 shows an example decision tree generated by lambdaMART. Note how the tree is able to combine features across different slots – for example, following the right-most path tests the scores of 3 different slots. Also, note how generally more positive evidence leads to higher scores.

Appendix B: Schematic of approach

Figure 2 shows a schematic diagram of our overall approach for training the state tracker (team2.entry0).

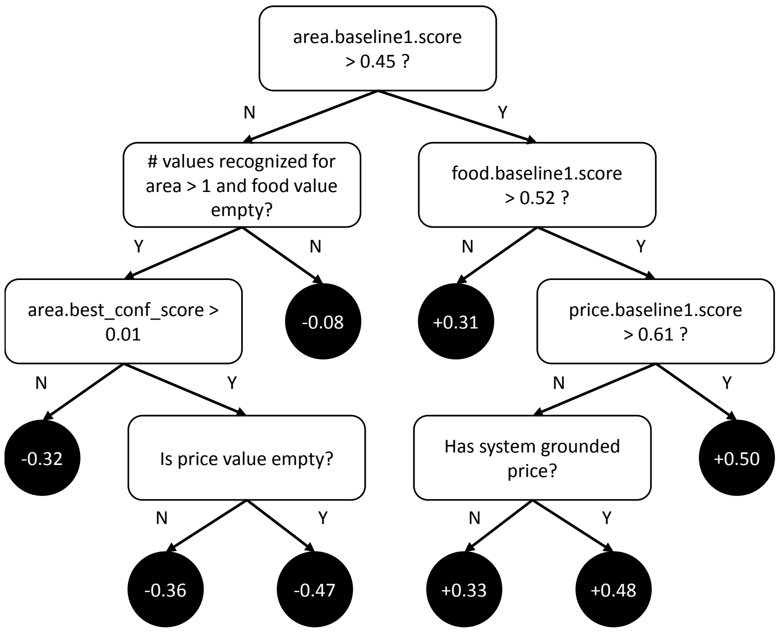
area.baseline1.score > 0.45 ?
N
Y
# values recognized for area > 1 and food value empty?
food.baseline1.score > 0.52 ?
Y
N
N
Y
area.best_conf_score > 0.01
-0.08
+0.31
price.baseline1.score > 0.61 ?
N
Y
N
Y
-0.32
Is price value empty?
Has system grounded price?
+0.50
N
Y
N
Y
-0.36
-0.47
+0.33
+0.48

Figure 1: Appendix A: Example decision tree with 8 leaves generated by lambdaMART. Each non-terminal node contains a binary test; each terminal node contains a real value that linearly contributes to the score of the dialog state being evaluated. "baseline1" refers to the output of one of the rule-based baseline trackers, used in this classifier as an input feature.

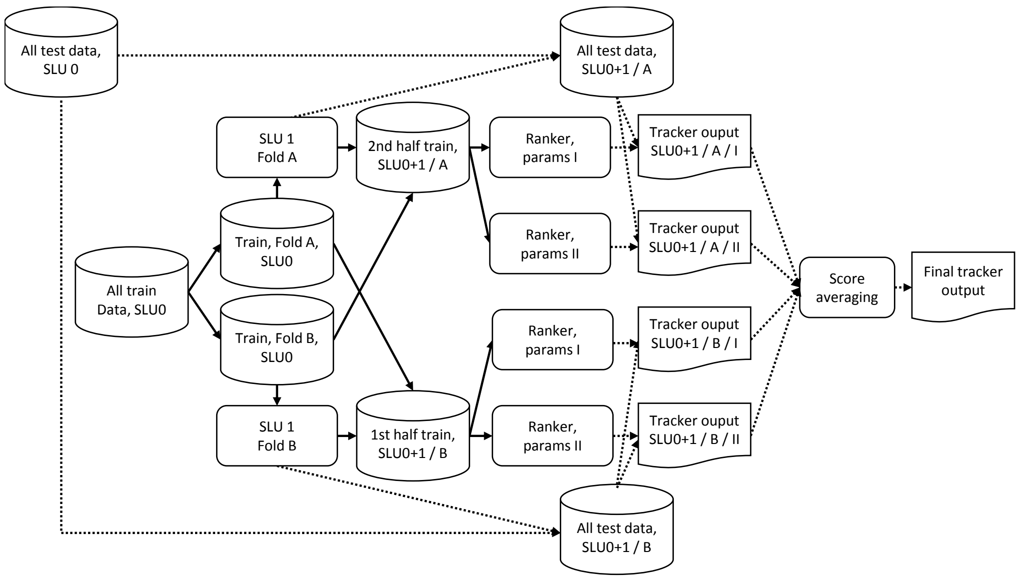
All test data, SLU 0
All test data, SLU0+1 / A
SLU 1 Fold A
2nd half train, SLU0+1 / A
Ranker, params I
Tracker ouput SLU0+1 / A / I
Train, Fold A, SLU0
Ranker, params II
Tracker ouput SLU0+1 / A / II
All train Data, SLU0
Score averaging
Final tracker output
Train, Fold B, SLU0
Ranker, params I
Tracker ouput SLU0+1 / B / I
SLU 1 Fold B
1st half train, SLU0+1 / B
Ranker, params II
Tracker ouput SLU0+1 / B / II
All test data, SLU0+1 / B

Figure 2: Appendix B: Schematic diagram of our overall approach for training the state tracker, using SLU1 (team2.entry0). Cylinders represent data, rectangles are models, and scripts are tracker output. Solid arrows are steps done at training time, and dotted arrows are steps done at test time. Approach for SLU2 (team2.entry1) is identical except that additional features are used in training the SLU models.