Is This Conversation on Track?

Paul Carpenter, Chun Jin, Daniel Wilson,
Rong Zhang, Dan Bohus, Alex Rudnicky

School of Computer Science, Carnegie Mellon University
Pittsburgh, Pennsylvania USA
{carpep+, cjin+, dwilson+, rongz+, dbohus+, air+}@cs.cmu.edu

Abstract

Confidence annotation allows a spoken dialog system to accurately assess the likelihood of misunderstanding at the utterance level and to avoid breakdowns in interaction. We describe experiments that assess the utility of features from the decoder, parser and dialog levels of processing. We also investigate the effectiveness of various classifiers, including Bayesian Networks, Neural Networks, SVMs, Decision Trees, AdaBoost and Naive Bayes, to combine this information into an utterance-level confidence metric. We found that a combination of a subset of the features considered produced promising results with several of the classification algorithms considered, e.g., our Bayesian Network classifier produced a 45.7% relative reduction in confidence assessment error and a 29.6% reduction relative to a handcrafted rule.

1. Introduction. Related work

The CMU Communicator system is a telephone-based dialog system that handles multiple travel tasks, including flight arrangement and hotel and car reservation [1]. The system is implemented as a distributed architecture, consisting of a series of parallel modules, such as speech recognition, parsing, dialog management, natural language generation, and speech synthesis. As the control point of the entire system, the dialog manager is responsible for analyzing the inputs from various modules, understanding their meaning, keeping track of interaction with the user, and determining the next operation (i.e. response) necessary to complete the task.

Unfortunately, machine recognition of speech is imperfect at best. Even small changes in the environment, telephone line quality, and the user's pronunciation may seriously impair recognition performance. The parsing module can also cause trouble by providing an incorrect result or failing to eliminate ambiguity. In many cases the system not only misunderstands the user, but it takes this misunderstanding as fact and continues to act using invalid information. When this happens a simple parsing error can grow until the entire interaction is ruined. The system is unaware of the problem because it has no means to judge how well the conversation is proceeding.

To avoid such situations, the CMU Communicator currently employs a model based on information about recognizer confidence and parse goodness. It also incorporates simple heuristics that monitor other dialog characteristics that are symptomatic of breakdown. We would like a more accurate confidence annotation scheme that integrates information from the decoding, parsing and dialog level into a single framework. The model should assign a confidence score as a continuous variable describing the probability that a certain utterance was correctly perceived by the system.

The confidence metric problem has been investigated previously [2, 3, 4, 5]. Most of this work has focused on how to detect the decoding errors made by the speech recognizer, and thus the proposed schemes work mainly at the frame, phoneme, or word level. For instance, word-level confidence annotation assigns a reliability tag to each word token in the decoder hypothesis. Typically a two-class annotation scheme is used, which marks the word instance correct or incorrect. However, this type of model is not always sufficient for dialog systems.

More recently, attempts have been made to use features from the other levels of the dialog system in deriving confidence metrics. For example [6] reports a study using decoder, language model and parsing features with a neural network classifier. Others [7] have used confidence metrics in the upper levels of language understanding and dialog management in order to achieve more flexible dialog and clarification strategies. Our work considers additional high-level features derived from the dialog manager and systematically compares several different classifiers.

We considered several machine learning techniques (e.g., Bayesian networks, boosting, decision trees, neural networks, support vector machines and naive Bayes classification) and tried to establish which of these is best suited for the task at hand. In Section 2 we describe the data and features that were used in training the classifiers. Section 3 starts with a preliminary analysis of the training set, and then describes in detail the experiments we performed and the results obtained. Section 4 does a comparative analysis of the results obtained by the different classifiers, and Section 5 concludes the paper and describes several directions for future work.

2. Data collection and feature extraction

2.1. Data collection and cleanup

We selected 2 successive months of CMU Communicator dialog data (logs and transcriptions) to work with. This information is logged automatically during telephone conversations hosted by the Communicator system. Each utterance in this dataset was hand-labeled as either OK or BAD. The OK label was assigned only to utterances free of all errors (e.g., parsing, recognition, etc.); otherwise the turn was labeled BAD.

Not all data were used, as a significant number of dialogs in this corpus were not well-formed. These were usually short conversations with no meaningful conclusion (e.g., hang-up calls, wrong numbers, etc.) We therefore established a criterion which required dialogs to have a minimum of 5 turns, others were discarded. Of the remaining data approximately 6% was further discarded because they contained a mix of OK and BAD labels. The cleaning process yielded a total of 4550 transcribed and labeled utterances.

2.2. Feature extraction

Choosing good features is paramount for the success of a classifier. From the multitude of features logged by the system we identified 12 that seemed most relevant for the task at hand. Generally speaking, these features can be grouped into 3 categories: *decoding*, *parsing* and *dialog*. To illustrate each feature clearly, we first present a sample extracted from the log file. The system prompt and user response are presented below, followed by the automatically generated hypothesis of the user's utterance and the parsing result.

`System:`	traveling to San Francisco International... and departing Pittsburgh on what day ?
`User:`	no I want to fly to Africa
`Hypothesis:`	NO I WANT A FLIGHT .?TO.? .?AFTER?. COME HOME
`Parsing:`	Respond [_no] (NO) Reserve_Flight [list] (I WANT A FLIGHT)

Sphinx, the Communicator speech recognition component, provides word-level confidence annotation for each word. This is denoted by the markers .? and ?. indicating that the tagged word is likely a misrecognition. The best hypothesis is then passed to the Phoenix parser [8] which produces a sequence of slots containing the concepts extracted from the utterance.

Decoding Features

1. **Word number** (word_num): The number of words in an utterance.
2. **Unconfident Percentage** (unconf): The percentage of the words tagged with the low confidence marker. The intuition is that a high unconfident percentage is often an indication of unreliable input. In the above example, "TO" and "AFTER" are tagged as unconfident words, so unconf = 2/9.

Parsing Features

1. **Uncovered Percentage** (uncov): The percentage of uncovered (or unparsed) words in a sentence. Similar to unconfident percentage, high uncovered percentage often means an unreliable input. In the above sample, the words "TO", "AFTER", "COME" and "HOME" are rejected by the parser, therefore uncov = 4/9.
2. **Fragment Transitions** (frag_num): The number of transitions between parsed fragments and unparsed fragments in a sentence. This feature describes the fragmentation degree of the parsing result. Frag_num is 1 for the sample sentence since there is one transition from the parsed fragment "NO I WANT A FLIGHT" to the unparsed fragment "TO AFTER COME HOME".
3. **Gap Number** (gap_num): The number of unparsed fragments in a sentence. In the sample sentence the second half part, "TO AFTER COME HOME", makes up a gap.
4. **Slot Number** (slot_num): The number of slots in the parsing result. There are two slots in the sample sentence above ([_no] and [list]).
5. **Slot Bigram** (bigram): The bigram score for the sequence of slots is computed from a bigram language model built for the parsing result considering the slots. Intuitively, an utterance with a high language model score is more likely to be a correctly decoded and parsed result.
6. **Garble** (garble): An input utterance is labeled as "[Garble]" by a post-parsing module if it has low coverage and high fragmentation (this is the current utterance level confidence metric used in the system).

Dialog Features

1. **Dialog State** (state): The current state of the dialog manager. The state for the sample sentence is query_depart_date.
2. **State Duration** (stay_here): The number of consecutive turns that the system remains in the same state. High values for this feature are also a good indicator of misunderstanding.
3. **Turn Number** (turn): The number of turns from the start of the conversation. Under normal conditions (no misunderstanding), there should be a correlation between the dialog state and turn number.
4. **Expected Slots** (expected_slot): This indicates whether or not the slots in the parsing result correspond to the current expectation of the dialog manager. For example, when the system is in the state query_depart_date, its expected slots are [date] or [time] .

3. Experiments

Deriving an utterance level confidence metric is essentially a classification task: given a set of features which characterize an utterance and its context, predict whether this utterance (as perceived by the system) is free of errors or not.

We explored several machine learning techniques and classifiers. The focus was on analyzing the capacity of these classifiers to correctly predict the binary target value of OK/BAD for each utterance. Most of the classifiers are nevertheless able to provide a continuous score, which is more fit for a true confidence metric (see Section 5 for future work).

As we wanted to be able to compare the performance of the classifiers, all the experiments described below were performed under the same conditions, on the same dataset, using 10-fold cross-validation. The dataset consisted of 4550 instances, each characterized by the 12 features described previously. 67.16% of these instances were labeled OK, thus giving a baseline error rate of 32.84% (when always considering the utterance as correct.)

The performance of each classifier is characterized by the mean and variance of the error rates in the cross-validation process. Another important factor is the *correct detection rate (CDR)*, i.e., the proportion of BAD utterances that are correctly identified. This number can be computed in terms of the false positives rate of each classifier, or in terms of the fallout, as the formula below illustrates (NBAD represents the total number of BAD utterances):

$$CDR = 1 - Fallout = 1 - \frac{FP}{NBAD} \qquad (1)$$

Note that there is a tradeoff between the correct detection rate and the number of false negatives (false alarms). A high correct detection rate can be achieved at the cost of introducing more false negatives. Therefore, to build the complete picture of the performance of the classifiers and their usefulness for confidence annotation we also report the false positive and false negative rates. Correct use of this information further requires the specification of a model that accurately captures the relative cost (to dialog efficiency) of false positives and false negatives. This cost will vary depending on the specific design of a dialog system. For example, the relative cost of a false negative in a system that requires an explicit backtrack or undo is higher than for a system that provides an over-write feature. A discussion of cost modeling is beyond the scope of the current paper.

3.1. Evaluation of individual features

We began by evaluating how well each individual feature is able to predict the target labels. The results are shown in Table 1, sorted with the best predictors on top.

Table 1: *Single Feature Prediction.*

Feature	**Mean Err.Rate**	**Var.**	**F/P Rate**	**F/N Rate**
Baseline	0.3284	-	-	-
uncov	0.1993	0.0012	0.1760	0.0233
expected_slot	0.2097	0.0006	0.1224	0.0873
gap_num	0.2301	0.0014	0.1451	0.0851
bigram	0.2314	0.0017	0.1580	0.0734
garble	*0.2532*	*0.0021*	*0.2530*	*0.0002*
slot_num	0.2569	0.0020	0.2552	0.0018
unconf	0.2734	0.0014	0.2618	0.0116
state	0.3103	0.0011	0.2582	0.0521
word_num	0.3233	0.0020	0.3207	0.0026
frag_num	0.3273	0.0015	0.2778	0.0495
stay_here	0.3284	0.0022	0.3202	0.0081
turn	0.3314	0.0021	0.3240	0.0075

As the table above illustrates, the features that best predict the target labels are **uncov**, **expected_slot** and **gap_num**, while the worst are **word_num**, **frag_num**, **stay_here** and **turn**. **garble** is a handcrafted rule incorporating **unconf**, **uncov** and **frag_num** information and is clearly inferior to some of the single features, including one of its components, in overall error rate. It does however provide by far the best false negative rate. Since this is the current confidence metric used in the Communicator system, we will consider the classification performance of **garble** (25.32%) as the baseline for the subsequent experiments.

3.2. Bayesian network classifier

Bayesian networks provide a probabilistic approach to inference. Bayesian reasoning assumes that our variables are describable by probability distributions, and that optimal decisions can be made by reasoning about these probabilities combined with observed data. This technique fits well with our problem because it provides a quantitative approach to judging the evidence supporting several hypotheses.

We used a very basic network structure in which each feature related directly to the classification, since we are interested in how the features affect the classification, not in how they affect each other. From Table 1 we can see that some features are more predictive than others. Accordingly, we placed the most predictive features together in the network, but observed results not much better than with individual features. This is likely because the features shared a large amount of mutual information. The trick was to discover which features worked best together (i.e., which features shared the least mutual information). We conducted further experiments using various subsets of features in our networks.

After some experimentation we discovered the combination of features that worked best: slot bigram, uncovered percentage, dialog state, garble, and expected slots. Training and testing on the network revealed an error rate of 17.82%, equivalent with a 29.62% relative reduction in error rate from the **garble** baseline (or 45.74% from the original baseline).

Table 2: *Performance of different classifiers*

Classifier	**Mean Err.Rate**	**Var.**	**F/P Rate**	**F/N Rate**
Garble (baseline)	0.2532	0.0021	0.2530	0.0002
AdaBoost	0.1659	0.0006	0.1143	0.0516
Decision Tree	0.1732	0.0008	0.1182	0.0549
Bayesian Net.	**0.1782**	**0.0008**	**0.0941**	**0.0842**
SVM	0.1840	0.0015	0.1501	0.0339
Neural Net.	0.1890	0.0008	0.1508	0.0382
Naive Bayes	0.2165	0.0014	0.1424	0.0741

3.3. AdaBoost classifier

Boosting is a voting technique which combines a set of weak learners (the performance of each individual learner needs to be slightly better than random) and iteratively boosts their performance by changing the distribution over the training examples to "focus" the learners on the hard instances. A more in-depth description of boosting can be found in [9].

A typical AdaBoost algorithm was employed. We considered the set of predictors based on each individual features as the weak learners. The relatively high error rate of each individual predictor (see Table 1) reduces the risk of overfitting and makes them good candidates for weak learners in AdaBoost.

The algorithm was run for 750 boosting stages. The mean error rate of the combined hypothesis was 16.59% (as illustrated in Table 2). This is equivalent with a 34.48% relative reduction of error from our **garble** baseline (or 49.48% from the original baseline). The variance was relatively low, indicating that no significant overfitting had occurred.

3.4. Decision tree classifier

Another widely used classification technique we decided to explore was decision trees. In this approach classification is performed by dividing the feature space into many small subspaces, and ultimately identifying each sub-space with a corresponding class. The partitioning process is implemented by iteratively choosing the next best feature based on information gain.

The average number of nodes in the resulting trees was around 300. The mean error rate and variance are relatively low, making decision trees one of the best classifiers in our experiments. We obtained a mean error rate of 17.31%, and thus a 31.64% reduction in error rate over the **garble** baseline.

3.5. Neural network classifier

Next, we turned to Artificial neural networks (or multi-layer perceptrons). This type of classifier is able to learn complex functions with continuous valued outputs, and is generally robust to noise in the training set. In our experiments, we used a typical three-layered feed-forward network architecture (with 50 nodes in the hidden layer). Training was done using the backpropagation algorithm.

Compared with the other classifiers, the performance of the neural network is slightly worse. The classification error rate of 18.90% puts it in fifth place. Moreover, the neural network exhibits a high false positives rate (15.01%) which makes it unsuitable for use in confidence annotation, as this translates into a correct detection rate significantly lower than that of the previous classifiers.

3.6. Support vector machine classifier

Support vector machines have received a great deal of attention in recent years. It has been shown that on some domains the performance of this approach is equivalent to those of traditional approaches as neural networks and decision trees. For many problems, it is difficult to find a classification boundary directly in the feature space. SVMs accomplish this by mapping the samples to a higher dimensional space using a kernel function, and then seeking a simple, linear separator in that space [10].

We examined several kernel functions such as dot, polynomial, radial, neural and anova in our experiments. Some kernel functions are more sensitive than others to the training samples. We report the results of the dot function, which had the most stable performance. Using this kernel function, the SVM achieved a 18.40% error rate, equivalent with a 27.33% reduction in error rate. Nevertheless, the false positives rate (and thus the correct detection rate) of the SVM classifier is similar to that of the neural network, making it an unlikely candidate for use in confidence annotation.

3.7. Naive Bayes classifier

Finally, we constructed a naive Bayes classifier. The 21.65% error rate placed this classifier on the last position. A t-test showed that its performance was significantly worse than that of the top three classifiers, but statistically indistinguishable from that of the SVM and Neural Network at the 0.05 level of confidence.

4. Results analysis

The results for the various classifiers are shown in Table 2. When judged by classification error, all the classifiers except the Naive Bayes perform similarly, achieving error rates around 18%. A t-test showed that there is no statistically significant difference between the mean error rates of these classifiers at the 0.05 level of confidence. Naive Bayes performs the worst. We suspect that this is due to the feature independence assumption made by this classifier. This assumption is clearly violated, as our experiments with the Bayesian network have indicated.

As mentioned in Section 3, in the context of building a confidence annotator another very important indicator is the correct detection rate (CDR). From this perspective, the Bayesian Network classifier has the best result. It gives a 9.41% false positives rate, which is equivalent with being able to correctly detect 71.35% of the BAD utterances. This correct detection rate is achieved in the context of a 8.42% false negative rate.

5. Conclusion

We described the development of an utterance-level confidence annotator scheme for the CMU Communicator spoken dialog system. Features from the decoder, parser, and dialog levels were used together with several classifiers and machine learning techniques to derive a predictor of the reliability of the input. In terms of classification error rate, with the notable exception of Naive Bayes, all the classifiers returned statistically indistinguishable results in the 16.5-19% range. The Bayesian Network classifier had the best correct detection rate (71.35%). All of these performed better than a handcrafted rule.

We regard the development of an accurate confidence annotator as an essential step towards a higher-level framework for confirmation and clarification in dialog systems. With a carefully designed scheme, the opportunity exists for selective error recovery techniques, including reminding, warning, asking the user to repeat, asking the user to confirm, or launching more sophisticated clarification sub-dialogs.

6. Acknowledgements

We would like to thank Tina Bennett for her help in labeling understanding accuracy, and Ananlada Chotimongkol for her help with constructing the slot bigram model. This research was sponsored in part by the Space and Naval Warfare Systems Center, San Diego, under Grant No. N66001-99-1-8905. The content of the information in this publication does not necessarily reflect the position or the policy of the US Government, and no official endorsement should be inferred.

7. References

[1] Rudnicky, A., et. al. “Creating natural dialogs in the Carnegie Mellon Communicator system”, Proceedings of Eurospeech, 1999.

[2] Bansal, D., and Ravishankar, M. K. “New Features for Confidence Annotation”, ICSLP-98.

[3] Chase, L. “Error-Responsive Feed back Mechanisms for Speech Recognizers”, Ph.D. Thesis, CMU. 1997.

[4] Cox S., and Rose R. “Confidence Measures for the Switchboard Database”, ICASSP-96.

[5] Kemp T., and Schaaf T. “Estimating Confidence Using Word Lattice”, EuroSpeech-97

[6] San-Segundo, R., Pellom, B., and Ward, W. “Confidence Measures for Dialogue Management in the CU Communicator System”, ICASSP-2000.

[7] Hazen, T., Burianek, T., Polifroni, J., Seneff, S. “Integrating Recognition Confidence Scoring with Language Understanding and Dialog Modeling”, ICASSP-2000.

[8] Ward W., and Issar S. “Recent Improvements in the CMU Spoken Language Understanding System”, Proceedings of the ARPA Human Language Technology Workshop, March 1994, 213-216.

[9] Schapire E. “A Brief Introduction to Boosting”, In Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, 1999

[10] Burges C., “A Tutorial on Support Vector Machines for Pattern Recognition”, Data Mining and Knowledge Discovery 2(2), 1998.