A Comparison of Classification Techniques for the Automatic Detection of Error Corrections in Human-Computer Dialogues

Katrin Kirchhoff
Department of Electrical Engineering
University of Washington
Seattle, USA
katrin@ee.washington.edu

Abstract

Previous work on automatically detecting users' error corrections in human-computer dialogues has mainly focused on finding optimal utterance features for classification. In this paper we compare and evaluate different classification schemes involving both single and multiple classifiers. We discuss preliminary experiments on error correction detection from partially observed utterances and present results obtained on travel-domain dialogues.

1 Introduction

In spite of recent advances in speech recognition and understanding technology, current automated dialogue systems are often unsatisfactory from a user's point of view. Natural language interaction is frequently impaired by failures of the system to understand the user's input (usually caused by speech recognition errors) and, moreover, by the lack of a principled strategy for gracefully recovering from such failures.

Misrecognitions by the system typically entail error correction attempts on the part of the user. It has been shown (Swerts et al., 2000) that such error correction utterances often exhibit significant acoustic variation, caused e.g. by hyperarticulation. For this reason, error correction utterances may be more difficult to recognize since they deviate more strongly from the recognizer's training data. This may lead to lengthy error correction subdialogues of the type exemplified in Figure 1.

The problem is that the same recognition strategy is typically used for both error corrections and regular utterances. Significant improvements could be made if a specialized module was used for the rapid detection of error corrections. Other modules could then be adapted based on the output of the error correction detector. Adaptation might include, for instance, the use of different acoustic models (e.g. for hyperarticulation), changes to the overall dialogue strategy, or different lexical choices during speech generation.

In this paper we describe the initial step towards such an adaptive error recovery procedure, viz. a module for the automatic detection of error corrections. Most of the previous studies which have addressed this problem have focussed on a specific dialogue system, and on exploiting internally computed system features (e.g. word confidence values, dialogue states) for error correction detection. Our focus, by contrast, is on developing system-independent classification methods which only make use of the acoustic signal itself, the "visible" system output (i.e. word recognition hypotheses and synthesized output strings), and generally available external resources such as pronunciation dictionaries or word taggers. The central goal of our work is to create a highly portable error detection module which can easily be applied to a new dialogue system without requiring an extensive development effort.

Furthermore, work on automatic error detection has concentrated on extracting and evaluating utterance features which aid classification. In this study we extend this work and address issues related to the classification procedure itself. In particular, we investigate classifier combination schemes such as cascading and boosting to improve performance. Finally, we look at the problem of online error correction detection, i.e. the possibility of making a classification decision while the utterance is still in progress.

The rest of this paper is organized as follows: in the following section we define the problem of error corrections in greater detail and give an

overview of previous and related work. In Section 3 we describe the corpus and the dialogue system which were used for the present study. Our error correction detection approach, as well as baseline results, are presented in Section 4. The baseline results are analyzed in Section 5; in Section 6 two classifier combination techniques are described and evaluated. Section 7 investigates the problem of online error correction detection; conclusions are presented in Section 8.

2 Error corrections in human computer dialogues

Figure 1 gives an example of an error correction subdialogue. We see that error corrections can be expressed in various ways: the user may explicitly contradict the system or may simply repeat or rephrase his original utterance. For the purpose of the present study we define an error correction as any conscious attempt by the user to change the system's current interpretation of one of his previous utterances.

Error corrections have been studied both descriptively and from the point of view of automatic classification. Descriptive studies (Oviatt et al., 1998; Hirasawa et al., 2000) of Wizard-of-Oz human-computer interactions have shown that error corrections differ from other utterances with respect to their acoustic-prosodic characteristics, in particular duration: error corrections typically contain more words, and both words and phones have a longer average duration. In an analysis of data collected in a real human-computer dialogue context, Swerts et al. (Swerts et al., 2000) additionally found that increases in F0 range and energy were correlated with error corrections.

An automatic error correction detector consisting of a decision tree trained on acoustic-prosodic features is described in (Levow, 1998). It achieved an accuracy of 75% on the task of detecting error corrections in command and control dialogues. Related studies have addressed the problems of automatic dialogue act classification (Wright, 1998; Stolcke et al., 2000) as well as the automatic prediction of misrecognised utterances (Hirschberg et al., 1999; Litman et al., 2000; Hirschberg et al., 2000). These studies all follow a similar approach in that they employ either decision trees or rule-based classifiers trained on a wide range of features characterizing the utterance. These include acoustic-prosodic features as described above, system-internal features such as word hypothesis confidence values, speaker-related information etc.

S: let's review your flight plans january fourth from denver to tampa is that correct
U: no i'd like to leave denver and go to istanbul on june fourth
H: no i'd like to leave denver and go to houston full on june fourth

S: june fourth from denver to houston is that correct
U: i'd like to go to istanbul
H: i'd like to go to bristol

S: june fourth from denver to bristol is that correct
U: i'd like to go to istanbul
H: i'd like to go to houston seoul

S: june fourth from seoul to houston is that correct
U: no i'd like to go from denver to istanbul
H: no i'd like to go from denver to bristol

S: june fourth from denver to bristol is that correct
U: no i'd like to go to istanbul
H: no i'd like to go to senegal

S: june fourth from denver to bristol is that correct
U: no istanbul
H: no a doubletree

S: what information would you like to change
U: start over
H: start over

Figure 1: Example of an error correction subdialogue. S: system, H: system's hypothesis, U: user

3 Corpus and Dialogue System

The experiments reported in this study are part of ongoing work within the DARPA Communicator project, whose goal is the development

of advanced dialogue systems supporting multimodal and multi-party dialogues and mixed-initiative interaction. For data collection we used the Communicator-compliant dialogue system provided by Colorado University (Ward and Pellom, 1999), which consists of an audio server, a speech recognizer, a NLP module, a dialogue manager, a database interface, a text-to-speech synthesizer and a user preference module. The speech recognizer output is processed by the NLP parser, which fills in a set of semantic slots. The dialogue manager then integrates the parse output into the current context and decides on the next step, which may consist of request for clarification, conclusion of the dialogue (summarization and termination), retrieval and presentation of data from the database, or requesting required information from the user. The dialogue domain is travel information. The system is connected live to an Internet travel database, which users can query about flight, hotel or rental car information.

Our data set consists of dialogues collected both at Colorado University (CU) and at the University of Washington (UW). The total number of dialogues is 1054. The set of dialogues was randomized and split into a training set of 1000 dialogues and an independent test set of 54 dialogues. The training set was used for development experiments, using 10-fold cross-validation. The average number of utterances in the training/test set during cross-validation is 13388 vs. 1487, respectively. The number of utterances in the independent test set is 789. The percentage of error corrections in each cross-validation test set varies between 13% and 24%; the percentage in the independent test set is 18.5%. The data was transcribed orthographically and labelled for error corrections by a trained linguist. The word error rate of the speech recognizer on this task is 33.8%.

4 Features and Baseline Classifier

Our goal is to develop a classifier which assigns a positive or negative error correction label to each user utterance. Similar to previous work on utterance classification we use decision trees (in Quinlan's C4.5 implementation (Quinlan, 1993)) trained on a set of features extracted from the utterance. Our baseline classifier uses three different types of features (cf. Table 1), which represent dialogue context, lexical and acoustic-prosodic information.

Type	Features
contextual	potential error location
	type of previous system utterance
lexical	has cue phrase
	number of words
	number of phones
	lexical overlap
	phonological overlap
acoustic-prosodic	F0 mean
	F0 max
	F0 stdev
	RMS mean
	RMS max
	RMS stdev
	total duration
	% of pause frames
	avg. pause duration
	initial pause duration
	avg. word duration
	avg. phone duration
	% of voiced frames
	speaking rate

Table 1: Features used for error correction detection.

The contextual features indicate the type of the previous system utterance (question vs. statement) and whether the current user utterance is a potential location for an error correction. The latter feature is positive when the previous system utterance contains either an explicit request for confirmation (such as "Is that correct?") or a signal of misunderstanding. The system may signal failures to understand the user either directly (e.g. "I didn't understand you") or indirectly, by literally repeating the previous output.

Although different systems use different prompts for signaling misunderstanding and confirmation requests, these contextual features can be rapidly computed from the 'visible' output of the system using simple string comparison techniques. Alternatively, since only a very small number of prompts are typically used, they can be specified in advance by the sys-

tem developer. However, our contextual features do not require access to system-internal logs of dialogue states or similar information and can therefore readily be ported to new dialogue systems.

The set of lexical features includes two measures of similarity between the current and the previous user utterance, viz. lexical overlap and phonological overlap. Both of these are based on the Levenshtein distance between the utterance strings, either measured at the word level or at the phone level. The phone-based distance measure is useful for encoding acoustic similarities between the utterances which are not reflected at the lexical level (e.g. homophones) and thus compensates to some extent for errorful word recognition output. The feature 'has cue phrase' indicates whether a cue phrase is present in the current utterance. Cue phrases are words or groups of words that are strongly correlated with error corrections. Lists of cue phrases are derived automatically based on a statistical analysis of the training data. First, user utterances are split into all possible subsequences up to a pre-specified length. Each word sequence s is then ranked by its probability given that the utterance is an error correction, $P(s|EC)$, minus its probability given other utterance types, $P(s|\neg EC)$. The n highest-ranking word sequences are then selected as cue phrases. Prior to classification, it is determined for each user utterance whether any of these cue phrases occurs in the utterance, using simple string matching. If this is the case, the attribute 'has cue phrase' is positive, else it is negative. Inspection of the automatically selected phrase lists showed that many negations and expressions of rejection were selected, such as "no", "nope", "I don't", "don't want to", etc. It was also observed that automatic cue phrase selection may be dominated by individual users with lengthy error correction subdialogues, or by the recording location (e.g. the word "Denver" occurred in our lists, due to the preponderance of dialogues collected at Colorado University). To avoid these effects, lists may be postprocessed by filtering out certain named entities, in particular proper names. The set of lexical features is complemented by the number of phones and words in the utterance.

The remaining features are acoustic-prosodic in nature and include minimum, maximum, mean and standard deviation of F0 and RMS, the total utterance duration, the percentage of pause frames and voiced frames (used for F0 computation), the duration of the initial pause, the average durations of words, phones, and pauses in the utterance, and speaking rate (Morgan and Fosler-Lussier, 1998).

We trained a baseline classifier on these features, which achieved an average error rate of 10.0% in the cross-validation experiments and an error rate of 10.1% on the independent test set. The expected error rates (obtained by always guessing the most frequent class), by contrast, were 18.1% on average on the development set and 18.5% on the test set. The results obtained by the baseline classifier are significantly better than the expected error rates.

5 Analysis

For the baseline experiments, no feature subset selection was performed prior to classification. Instead, the decision tree learning algorithm acts itself as a feature selector: attributes which are good predictors of error corrections will be identified during training and will be utilized at the higher levels in the tree, whereas other attributes may be discarded and do not occur in the final tree. An analysis of the trained decision trees therefore sheds light on the relative importance of the different input features. In order to better understand the performance of the baseline classifier we inspected the trained decision trees and simpler *if-then* rules derived from these. We found that the dialogue context features were the most important, followed by lexical features, whereas prosodic features were found at the lowest levels in the trees. Our analysis showed that a large number of test cases could be classified reliably using very simple rules. An example is given below, where the rule classifies an utterance as an error correction if it contains a cue phrase, it is a potential error correction location and the previous utterance is a question.

if (prev. utterance type = question $\wedge$
has cue phrase = true $\wedge$
potential error correction location = true) *then*
is error correction

Contextual and lexical features are strong predictors of error corrections, both individually and in combination. Utterances lacking these features are handled by lower-level subtrees which test for more complex combinations of acoustic-prosodic feature-values and lexical features.

6 Multiple Classifiers

Since many utterances can be classified using very simple rules it seemed promising to employ a more sophisticated classification scheme which employs multiple classifiers focusing on different regions in the input space. We investigated two such approaches, multistage cascading and boosting.

6.1 Cascading

Multistage classifier cascading (Alpaydin and Kaynak, 1998; Kaynak and Alpaydin, 2000) uses a sequence of classifiers in which a simple, low-complexity classifier is applied first to all test cases. Samples which cannot be classified reliably by the first classifier are handled by a second, more expensive classifier. The central idea is that the first classifier should be fast and easy to train and should reliably account for the majority of test cases. The second classifier is used only sparingly and can therefore exploit a more costly (e.g. memory-based) classification procedure. The threshold for sample rejection is determined by the confidence of the first-level classification decision, which may be, for instance, the posterior probability of the assigned class k given the input sample x, $P(k|x)$.

The cascading algorithm given in (Alpaydin and Kaynak, 1998) uses a linear discriminant first-level classifier and a k-nearest neighbour (kNN) classifier at the second level:

Cascading

1. Train a linear discriminant classifier on a given training set.
2. Determine a confidence threshold θ, $0 < \theta < 1$.
3. For each sample in a separate validation set
   (a) Classify the sample
   (b) Check if the outcome is correct and the confidence value of the decision is larger than θ. If this is not the case, store the pattern in an exception table Z.
4. For each sample in the test set
   (a) Apply the first-level classifier
   (b) Check if the confidence of the classification falls below θ. If this is the case, apply kNN, using the stored examples in Z, to find the most likely class.

Naturally, this scheme can be generalized to more than two classifiers and to a broader range of classification algorithms. For our purpose we use a decision tree as the first-level classifier and kNN as the second classifier. The confidence values for the classifications made by the decision tree are computed by the Laplace ratio

$$C_k(x) = \frac{n_k + 1}{m + 2} \quad (1)$$

as suggested in (Quinlan, 1996). Here, $C_k(x)$ is the confidence with which sample x is assigned to class k, m is the total number of training samples assigned to the the leaf in the decision tree which is used to classify x, and n_k is the fraction of those training samples which belong to class k. The kNN classifier uses a Euclidean distance measure. Given k nearest neighbours, the final output class is determined by majority voting. If k is even and majority voting leads to a tie, the distance values themselves are taken into consideration and the class incurring the smallest average distance is chosen. In order to optimize the parameters θ and k we again used cross-validation, with 80% of the training data being used for training, 10% for validation and 10% for testing. The optimal values for θ and k were found to be 0.7 and 3, respectively. In order to compute the exception table for classification on the independent test set, 80% of the training data was used for training the first-level classifier, and 20% was used for validation. When applied to the independent test set, the error rate was reduced to 9.7%. The percentage of cases classified by the decision tree and kNN, respectively, was 94% vs. 6%.

6.2 Boosting

Another way of combining different classifiers is boosting (Schapire, 1990; Drucker et al., 1994). During boosting multiple classifiers are trained

sequentially; the training set for each classifier is determined by the errors of the the previous classifier in that previously misclassified samples are given more weight. Unlike cascading, however, boosting then combines all trained classifiers in parallel, using a weighted sum of their output scores. The particular algorithm we use is *AdaBoost* (Freund and Schapire, 1996):

AdaBoost

given: Training set with N samples $\{(x_i, d_i)\}_{i=1}^{N}$
Distribution $\mathcal{D}$ over training samples
Classifier

1. For all i, set $\mathcal{D}_1(i) = 1/N$
2. For $t = 1, 2, ...T$ iterations
   (a) Train classifier on distribution $\mathcal{D}_t(i)$
   (b) Apply the classifier to the training samples and obtain a class assignment $F_t : \mathbf{X} \rightarrow Y$
   (c) Compute the training error $\epsilon_t = \sum_{i:F_t(\mathbf{x}_i) \neq d_i} \mathcal{D}_t(i)$
   (d) Set $\beta_t = \frac{\epsilon_t}{1-\epsilon_t}$
   (e) Update $\mathcal{D}_t(i)$ such that

$$\mathcal{D}_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} \beta_n \text{ if} F_t(\mathbf{x}_i) = d_i \\ 1 \quad \text{otherwise} \end{cases}$$

   Z_t is a normalization constant such that $\mathcal{D}_{t+1}$ is a valid probability distribution.
3. Combine all trained classifiers by

$$F_T(\mathbf{x}) = argmax_{d \in \mathcal{D}} \sum_{t:F_t(\mathbf{x})=d} log\frac{1}{\beta_t}$$

The different weighting of training samples in each iteration can either be implemented as actual weights applied to individual samples, or a new training set can be generated by resampling according to $\mathcal{D}$. For our experiments we used boosting by resampling. In order to optimize the number of boosting iterations we chose the same 80%/20% split of the training data which was used for cascading. The optimal number of boosting iterations (after which performance improvements are virtually zero) was determined to be 40. Boosting applied to the independent test set yielded an error rate of 7.9%.

6.3 Comparison

While boosting leads to an appreciable reduction of the classification error rate (significant at the 0.1 level using a difference of proportions significance test), cascading only yields a very small improvement. This finding departs from earlier experimental results obtained on a smaller data set, where cascading performed better than boosting. The performance improvements to be gained from these different multiple classifier schemes therefore seem to depend primarily on on the availability of sufficient training data. Since boosting makes use of several fully trained classifiers, additional training data will have a beneficial effect. Cascading, on the other hand, does not increase the number of parameters to be trained but instead relies on a comparison with previously stored examples. The performance of the cascading algorithm depends on the size of the exception table and on the nature of the examples stored. When more training material is available for training the baseline classifier, fewer samples may be filtered out by the confidence threshold criterion and the resulting exception table may be less suitable for the second-level classifier. Further experiments will be performed in the future to further compare these two methods. It should be noted that cascading is to be preferred from the computational point of view: less computational effort is required for cascaded classifiers than for parallel classifier, both during the training and the testing phase. This may be of relevance in actual applications of dialogue systems.

7 Online Error Correction Detection

We finally turn to the question of how early in an utterance it is possible to decide reliably whether it is an error correction or not. This in an important question because it relates to the possibility of dynamically and rapidly adapting to the current situation while the error correction utterance is still in progress. From a pattern classification point of view, this scenario raises the problem of potential missing feature values. Whereas values for the continuous acoustic-prosodic features can be estimated from even a few frames, the question whether the utterance contains any cue phrase, for instance, can in principle not be answered reliably

until the end of the utterance. When encoding such partial test samples, these features may either receive a negative value or the value may be treated as missing. We simulated online error correction detection by re-computing all features in Table 1 after every new word hypothesis in the time-aligned word recognition output and applying the classifier to every new feature vector. For this experiment we used the baseline decision-tree classifier trained on features computed over complete utterances. Figure 2 plots the percentage of correctly classified utterances against the percentage of words processed. It is noticeable that correct classification can be in approximately 90% of the cases after only 50% of the words in the utterance have been processed. The reason for this can be seen in the fact that users typically place information which is correlated with error corrections near the beginning of the utterance (esp. negatives like "No", "Correction") - in other utterance types, important information is often placed at the end ("I'd like to go to Denver").

In a second experiment we replaced the binary values for the 'cue phrase' feature with missing values in cases where no error correction had been detected but the utterance was not yet complete. C4.5 handles missing values by arithmetically combining the classifications resulting from all possible instantiations of the missing feature, weighted by the probability of these instantiations as determined from the set of training samples with known values for that attribute. We found that missing feature values did not result in any significant changes to the classification accuracy compared to using fully specified (but possibly unreliable) feature values.

8 Summary and Conclusions

In this paper we have presented various classification schemes for the automatic detection of error corrections in human-computer dialogues. We first described a baseline decision tree classifier trained on a set of heterogeneous utterance features, including contextual, lexical and acoustic-prosodic characteristics. The baseline classifier achieved an error rate of 10.0% on a training/development set and an error rate of 10.1% on an independent test set. Both of these results constitute a statistically significant improvement over the expected error rates. We furthermore investigated two classifier combination schemes, cascading and boosting. Cascading uses a sequence of classifiers, where the first-level classifier handles the majority of test cases and the second-level classifier is applied to low-confidence outliers. Boosting, by contrast, applies several classifiers in parallel. We observed an appreciable reduction in error rate for boosting (from 10.1% to 7.9%) but only a small reduction for cascading (from 10.1% to 9.7%).

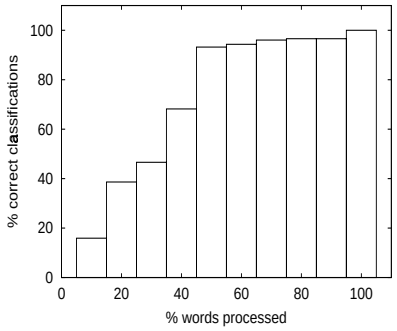

Figure 2: Online error correction detection: percentage of words processed vs. percentage of correct classifications

Finally, we reported preliminary results on online error correction detection, using both test samples with missing values and test samples with fully specified (but possibly unreliable) values. Generally, the results are promising for future work on within-dialogue adaptation: utterances can often be classified reliably after only 50% of the words in the utterance have been seen.

The current study has mainly focused on comparing different classification schemes and is part of a larger effort devoted to the development of highly portable error correction detectors. The results obtained here could of course be improved further by adding system-specific features such as word hypothesis confidence values. Another issue worth pursuing in the future is user-specific error correction modeling. Different users often differ markedly in their error correction activities whereas individual users typically adhere to a fairly consistent pattern. Rather than applying an across-the-board detection strategy, different error correction detectors could be applied to different users. In many

cases it will not be possible to collect a sufficient amount of data from a single user to build a truly speaker-dependent error correction detector. However, error correction detection may be conditioned on the prior classification of the user, taking into account fairly general lexical and contextual dialogue patterns. Technically experienced users, for instance, tend to interact more naturally with the system and produce lengthy utterances. Novices, by contrast, often provide the system with only the bare minimum of information and seem to consciously try to limit their utterances to single words ("yes/no") or short phrases.

In the future we intend to integrate automatic error correction detection with the acoustic modeling component in the speech recognizer and with the speech generation component. Based on the error correction classification decision, the user utterance may be rescored using specialized acoustic models for hyperarticulation. Moreover, systems prompts will be changed dynamically, in order to enable the system to respond more quickly to the user's error correction efforts.

Acknowledgements

This work was funded by DARPA, Contract N660019928924. We thank Eric Fosler-Lussier for sharing his speaking rate estimation software.

References

E. Alpaydin and C. Kaynak. 1998. Cascading classifiers. *Kybernetika*, 34(4):369–774.

H. Drucker, C. Cortes, L.D. Jackel, and Y. LeCun. 1994. Boosting and other ensemble methods. *Neural Computation*, pages 1289–1301.

Y. Freund and R.E. Schapire. 1996. Experiments with a new boosting algorithm. In *Proceedings of Thirteenth International Conference on Machine Learning*, pages 148–156, Bari, Italy.

J. Hirasawa, N. Miyazaki, M. Nakano, and K. Aikawa. 2000. New feature parameters for detecting misunderstandings in spoken dialogue systems. In *Proceedings of ICSLP*, Beijing, China.

J. Hirschberg, D. Litman, and M. Swerts. 1999. Prosodic cues to recognition errors. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*, Keystone, Colorado.

J. Hirschberg, D. Litman, and M. Swerts. 2000. Generalizing prosodic prediction of speech recognition errors. In *Proceedings of ICSLP*, Beijing, China.

C. Kaynak and E. Alpaydin. 2000. Multistage cascading of multiple classifiers: one man's noise is another man's data. In *Proceedings of ICML*, pages 455–462, Stanford, California.

G.A. Levow. 1998. Characterizing and recognizing spoken corrections in human-computer dialogue. In *Proceedings of ACL*.

D. Litman, J. Hirschberg, and M. Swerts. 2000. Predicting automatic speech recognition performance using prosodic cues. In *Proceedings of NAACL-ANLP*, Seattle, Washington.

N. Morgan and E. Fosler-Lussier. 1998. Combining multiple estimates of speaking rate. *ICASSP 98*, pages 729–732.

S. Oviatt, G.A. Levow, E. Moreton, and M. MacEachern. 1998. Modeling global and focal hyperarticulation during human-computer error resolution. *JASA*, 104(5):3080–3098.

J.R. Quinlan. 1993. *C4.5: Programs for Machine Learning*. Morgan Kaufman, San Mateo, CA.

J.R. Quinlan. 1996. Boosting, bagging and C4.5. In *Proceedings of AAAI*.

R.E. Schapire. 1990. The strength of weak learnability. *Machine Learning*, 5:197–227.

Stolcke et al. 2000. Dialogue act modeling for automatic tagging and recognition of conversational speech. *Computational Linguistics*, 26(3):339–373.

M. Swerts, D. Litman, and J. Hirschberg. 2000. Corrections in spoken dialogue systems. In *Proceedings of ICSLP*, Beijing, China.

W. Ward and B. Pellom. 1999. The CU Communicator system. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*, Keystone, Colorado.

H. Wright. 1998. Automatic utterance type detection using suprasegmental features. In *Proceedings of ICSLP*, Sydney, Australia.