

# LATTICE KERNELS FOR SPOKEN-DIALOG CLASSIFICATION

Corinna Cortes      Patrick Haffner      Mehryar Mohri

AT&T Labs - Research  
180 Park Avenue  
Florham Park, NJ 07932, USA  
{corinna,haffner,mohri}@research.att.com

## ABSTRACT

Classification is a key task in spoken-dialog systems. The response of a spoken-dialog system is often guided by the category assigned to the speaker's utterance. Unfortunately, classifiers based on the one-best transcription of the speech utterances are not satisfactory because of the high word error rate of conversational speech recognition systems. Since the correct transcription may not be the highest ranking one, but often will be represented in the word lattices output by the recognizer, the classification accuracy can be much higher if the full lattice is exploited both during training and classification. In this paper we present the first principled approach for classification based on full lattices. For this purpose, we use the Support Vector Machine framework with kernels for lattices. The lattice kernels we define belong to the general class of rational kernels. We give efficient algorithms for computing kernels for arbitrary lattices and report experiments using the algorithm in a difficult call-classification task with 38 categories. Our experiments with a trigram lattice kernel show a 15% reduction in error rate at a 30% rejection level.

## 1. MOTIVATION

One of the key tasks of spoken-dialog systems is classification. This consists of assigning, out of a finite set, a specific category to each speech utterance based on the transcription of that utterance by a speech recognizer. The choice of possible categories depends on the dialog context considered. A category may correspond to the type of billing problem in the context of a dialog related to billing, or to the type of problem raised by the speaker in the context of a hot-line service.

Categories are used to direct the dialog manager in formulating a response to the speaker's utterance. Classification is based on features such as relevant key words or key sequences used by a machine learning algorithm.

Unfortunately, the word error rate of conversational speech recognition systems is still too high in many tasks to rely only on the one-best output of the recognizer. However, the word lattices output by speech recognition systems may contain the correct transcription in most cases. Thus, it is preferable to use instead the word lattices for classification.

The application of classification algorithms to word lattices raises several issues. Even small lattices may contain billions of paths, thus the algorithms cannot be generalized by simply applying them to each path of the lattice. Additionally, the paths are weighted and these weights must be used to guide appropriately the classification task.

Recently, a general family of kernels based on weighted transducers or rational relations, *rational kernels*, was introduced [3]. These kernels apply to lattices or arbitrary weighted automata. Kernel methods [15] are widely used in statistical learning techniques such as Support Vector Machines (SVMs) [2, 4, 16] due to their computational efficiency in high-dimensional feature spaces. Combining rational kernels for lattices with SVMs constitutes the first principled technique for efficient classification algorithms based on the full lattice. The use of lattice kernels can be adapted

SEMIRING	SET	$\oplus$	$\otimes$	$\bar{0}$	$\bar{1}$
Boolean	$\{0, 1\}$	$\vee$	$\wedge$	0	1
Probability	$\mathbb{R}_+$	+	$\times$	0	1
Log	$\mathbb{R} \cup \{-\infty, +\infty\}$	$\oplus_{\log}$	+	$+\infty$	0
Tropical	$\mathbb{R} \cup \{-\infty, +\infty\}$	min	+	$+\infty$	0

**Table 1.** Semiring examples.  $\oplus_{\log}$  is defined by:  $x \oplus_{\log} y = -\log(e^{-x} + e^{-y})$ .

to other problems than classification, such as principle component analysis [15].

In this paper, we define a simple rational kernel and show its applications to spoken-dialog classification using SVMs. We briefly describe the generic method based on general weighted automata algorithm [3] for computing this kernel. We then give a more efficient algorithm for the special case of the lattice kernel considered here. Finally, we apply these kernels to a difficult call-classification task with 38 classes, AT&T *How May I Help You* (*HMIHY<sup>TM</sup>*). Our experiments show that lattice kernels reduce the SVM error rate by more than 15% at the 30% rejection level. This rejection level is the standard operation point for the task. Note that this task is different from the easier 19-class task described in the paper [6].

## 2. PRELIMINARIES

In this section, we present the definition and notation necessary to introduce lattice kernels.

A *weighted automaton* is a finite automaton in which each transition carries some weight element of a *semiring* in addition to the usual alphabet symbol.

**Definition 1 ([7])** A system  $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$  is a semiring if:  $(\mathbb{K}, \oplus, \bar{0})$  is a commutative monoid with identity element  $\bar{0}$ ;  $(\mathbb{K}, \otimes, \bar{1})$  is a monoid with identity element  $\bar{1}$ ;  $\otimes$  distributes over  $\oplus$ ;  $\bar{0}$  is an annihilator for  $\otimes$ : for all  $a \in \mathbb{K}$ ,  $a \otimes \bar{0} = \bar{0} \otimes a = \bar{0}$ .

Thus, a semiring is a ring that may lack negation. Table 1 lists some familiar examples of semirings. In addition to the Boolean semiring and the probability semiring used to combine probabilities, two semirings often used in applications are the *log semiring* which is isomorphic to the probability semiring via a log morphism, and the *tropical semiring* which is derived from the log semiring using the Viterbi approximation. To shorten the notation, in the following we will denote the semiring system  $(\mathbb{K}, \oplus, \otimes, \bar{0}, \bar{1})$  simply by the semiring set  $\mathbb{K}$ .

**Definition 2** A weighted finite-state transducer  $T$  over a semiring  $\mathbb{K}$  is an 8-tuple  $T = (\Sigma, \Delta, Q, I, F, E, \lambda, \rho)$  where:  $\Sigma$  is the finite input alphabet of the transducer;  $\Delta$  is the finite output alphabet;  $Q$  is a finite set of states;  $I \subseteq Q$  the set of initial states;  $F \subseteq Q$  the set of final states;  $E \subseteq Q \times (\Sigma \cup \{\epsilon\}) \times (\Delta \cup \{\epsilon\}) \times \mathbb{K} \times Q$  a finite set of transitions;  $\lambda : I \rightarrow \mathbb{K}$  the initial weight function; and  $\rho : F \rightarrow \mathbb{K}$  the final weight function mapping  $F$  to  $\mathbb{K}$ .

Weighted automata can be formally defined in a similar way by simply omitting the input or output labels.

Given a transition  $e \in E$ , we denote by  $i[e]$  its input label,  $p[e]$  its origin or previous state and  $n[e]$  its destination state or next state,  $w[e]$  its weight,  $o[e]$  its output label (transducer case). Given a state  $q \in Q$ , we denote by  $E[q]$  the set of transitions leaving  $q$ .

A path  $\pi = e_1 \cdots e_k$  is an element of  $E^*$  with consecutive transitions:  $n[e_{i-1}] = p[e_i]$ ,  $i = 2, \dots, k$ . We extend  $n$  and  $p$  to paths by setting:  $n[\pi] = n[e_k]$  and  $p[\pi] = p[e_1]$ . We denote by  $P(q, q')$  the set of paths from  $q$  to  $q'$  and by  $P(q, x, y, q')$  the set of paths from  $q$  to  $q'$  with input label  $x \in \Sigma^*$  and output label  $y$  (transducer case). These definitions can be extended to subsets  $R, R' \subseteq Q$ , by:  $P(R, x, y, R') = \bigcup_{q \in R, q' \in R'} P(q, x, y, q')$ .

The labeling functions  $i$  (and similarly  $o$ ) and the weight function  $w$  can also be extended to paths by defining the label of a path as the concatenation of the labels of its constituent transitions, and the weight of a path as the  $\otimes$ -product of the weights of its constituent transitions:  $i[\pi] = i[e_1] \cdots i[e_k]$ ,  $w[\pi] = w[e_1] \otimes \cdots \otimes w[e_k]$ . We also extend  $w$  to any finite set of paths  $\Pi$  by setting:  $w[\Pi] = \bigoplus_{\pi \in \Pi} w[\pi]$ .

The output weight associated by a transducer  $T$  to any pair of input-output string  $(x, y)$  is:

$$\llbracket T \rrbracket(x, y) = \bigoplus_{\pi \in P(I, x, y, F)} \lambda(p[\pi]) \otimes w[\pi] \otimes \rho(n[\pi]) \quad (1)$$

with  $\llbracket T \rrbracket(x, y) = \bar{0}$  when  $P(I, x, y, F) = \emptyset$ .

### 3. LATTICE KERNELS

#### 3.1. Kernel Methods

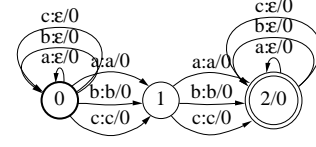
Classification algorithms such as SVMs have primarily been applied to input vectors  $x \in \chi$  of fixed length or structure. When the input vectors are not linearly separable, one forms non-linear features based on the input features using a mapping  $\Phi : \chi \rightarrow F$  such that the images  $\Phi(x)$ ,  $x \in \chi$ , of the input vectors, are linearly separable in the possibly higher-dimensional feature space  $F$ .  $F$  is assumed to be a Hilbert space. Training and classification by SVMs require the computation of the dot product in  $F$ . In many cases, that dot product may be very costly to compute directly due to the high dimensionality of the feature space. This problem can be overcome by using instead a *kernel*, that is a function  $K$  such that for all  $x, y$  in the classification set  $\chi$ :

$$K(x, y) = (\Phi(x), \Phi(y))$$

Kernels may often be more efficient to compute because they avoid the explicit computation of  $\Phi(x)$ . Functions  $K$  that ensure that the SVM optimization problem has a unique solution must satisfy certain conditions. It can be proved that if  $K$  is *symmetric*, *positive*, and *definite*, then these conditions are satisfied.

Kernels  $K$  that meet these conditions can be used to construct other families of kernels meeting these conditions [15]. *Polynomial kernels* of degree  $p$  are formed from the expression  $(K + a)^p$ , and *Gaussian kernels* can be formed as  $\exp(-d^2/\sigma^2)$  with  $d^2(x, y) = K(x, x) + K(y, y) - 2K(x, y)$ .

For our spoken-dialog classification problem we need to define kernels over word (or phone lattices). Lattice kernels that are symmetric, positive, and definite will allow us to apply the computationally efficient SVM classification technique directly to the lattices. Note that the number of linear features that can be constructed from a lattice may be very large. If one can construct a weighted transducer to efficiently compute lattice kernels, this can result in significant computational savings even in the linearly separable case.



**Fig. 1.** Weighted transducer  $T$  over the log semiring with  $\Sigma = \{a, b, c\}$ . The transition weights and the final weight at state 1 are all equal to 0.

#### 3.2. Lattice Kernel for Spoken-Dialog Classification

A general family of kernels for weighted automata, *rational kernels*, was introduced in [3]. The definition of these kernels is based on weighted finite-state transducers. In this section, we define a lattice kernel relevant to spoken-dialog classification tasks that belongs to the general class of rational kernels.

The basic idea behind the definition of this kernel is that two utterances are similar when they share many common  $n$ -gram sequences. The exact transcriptions of the utterances are not available but we can use the word (or phone) lattices output by the recognizer. A word lattice  $L$  can be viewed as a probability distribution  $P_L$  over all strings  $s \in \Sigma^*$ . Denote by  $|s|_x$  the number of occurrences of a sequence  $x$  in the string  $s$ . The expected count or number of occurrences of an  $n$ -gram sequence  $x$  in  $s$  for the probability distribution  $P_L$  is:

$$c(L, x) = \sum_s P_L(s) |s|_x$$

Two lattices output by a speech recognizer can be viewed as similar when the sum of the product of the expected counts they assign to their common  $n$ -gram sequences is sufficiently high. Thus, we define an  $n$ -gram kernel  $K$  for two lattices  $L_1$  and  $L_2$  by:

$$K(L_1, L_2) = \sum_{|x|=n} c(L_1, x) c(L_2, x)$$

It is already clear from this expression that our kernel  $K$  is symmetric, positive and definite since it is written as a dot product. The question is whether we can construct a weighted transducer that efficiently computes  $K$ .

Fortunately, there exists a simple weighted transducer over the log semiring that for any lattice  $L$  computes  $-\log(c(L, x))$  for all  $n$ -gram sequences  $x$ . Figure 1 shows that transducer in the case of bigram sequences ( $n = 2$ ) and for the alphabet  $\Sigma = \Delta = \{a, b, c\}$ . More generally, the rational relation corresponding to the transducer  $T$  with input and output alphabet  $\Sigma$  is:

$$(\Sigma \times \{\epsilon\})^* (\bigcup_{a \in \Sigma} (\{a\} \times \{a\})^n) (\Sigma \times \{\epsilon\})^*$$

and the final weight and the weights of all transitions is 0.

For any transducer  $U$ , denote by  $\pi_2(U)$  the automaton obtained from  $U$  by omitting its input labels. Then, for any  $n$ -gram sequence  $x$ ,  $\llbracket \pi_2(L_1 \circ T) \rrbracket(x) = -\log(c(L_1, x))$ , and similarly  $\llbracket \pi_2(L_2 \circ T) \rrbracket(x) = -\log(c(L_2, x))$ . Thus, by definition of composition [11]:

$$\begin{aligned} -\log(K(L_1, L_2)) &= -\log \left( \sum_{|x|=n} c(L_1, x) c(L_2, x) \right) \\ &= -\bigoplus_{\substack{\log \\ |x|=n}} -\log(c(L_1, x)) - \log(c(L_2, x)) \\ &= -\bigoplus_{\substack{\log \\ |x|=n}} \llbracket \pi_2(L_1 \circ T) \rrbracket(x) + \llbracket \pi_2(L_2 \circ T) \rrbracket(x) \\ &= -\bigoplus_{\substack{\log \\ s}} \llbracket L_1 \circ (T \circ T^{-1}) \circ L_2 \rrbracket(s) \end{aligned}$$

This proves that  $K$  is a rational kernel based on the transducer  $T \circ T^{-1}$ . Thus, the kernel we define falls within the general framework of SVM optimization [3].

### 3.3. Kernel Computation Algorithms

The definition of the kernel  $K$  suggests a simple algorithm for its computation based on general weighted automata and graph algorithms: composition of weighted transducers to compute  $L_1 \circ (T \circ T^{-1}) \circ L_2$  [11, 13], and a general shortest-distance algorithm to compute the  $\oplus_{\log}$ -sum of the weights of all the paths of this machine [10]. The general utilities provided by the FSM library can be used to compute  $K(L_1, L_2)$  for lattices  $L_1$  and  $L_2$  [12].

Note that the size of the transducer  $T$  is  $O(n|\Sigma|)$ . In practice, a lazy implementation simulating the presence of the transitions of  $T$  labeled with all elements of  $\Sigma$  reduces the size of the machine used to  $O(n)$ . Since the complexity of composition is quadratic [11, 13] and since the general shortest distance algorithm just mentioned is linear for acyclic graphs such as the lattices output by speech recognizers, the worst case complexity of the algorithm is:  $O(n^2 |L_1| |L_2|)$ .

In what follows, we present a more efficient algorithm for computing  $K(L_1, L_2)$  whose complexity is linear in  $|L_1|$  and  $|L_2|$  for a fixed  $n$ :  $O(n(|L_1| + |L_2|))$ . We first show that  $L \circ T$  can be computed efficiently for any lattice  $L = (Q, I, F, \Sigma, \delta, \sigma, \lambda, \rho)$  using a general shortest distance algorithm over the log semiring [10]. For each state  $q \in Q$ , denote by  $d[q]$  the shortest distance from  $I$  to  $q$  and by  $f[q]$  the shortest distance from  $q$  to  $F$ :

$$d[q] = \bigoplus_{\log, \pi \in P(I, q)} (\lambda(p[\pi]) + w[\pi]) \quad f[q] = \bigoplus_{\log, \pi \in P(q, F)} (w[\pi] + \rho(n[\pi]))$$

The shortest distances  $d[q]$  and  $f[q]$  can be computed for all states  $q \in Q$  in linear time ( $O(|L|)$ ) since  $L$  is an acyclic weighted automaton [10].

**Proposition 1** *Let  $L$  be a weighted automaton over the log semiring. Then, for any  $n$ -gram  $x$ ,  $-\log(c(L, x))$  can be computed by taking the  $\oplus_{\log}$ -sum of the weights of all paths  $\pi$  of  $L$  labeled with  $x$ , and the shortest distances to the origin and from the destination state of  $\pi$ :*

$$-\log(c(L, x)) = \bigoplus_{\log, i[\pi]=x} d[p[\pi]] + w[\pi] + f[n[\pi]]$$

*Proof.* To emphasize the distributivity of  $+$  over  $\oplus_{\log}$ , we will use the notation  $\otimes$  instead of  $+$  in this proof. By definition of  $c(L, x)$ ,  $-\log(c(L, x))$  is the  $\oplus_{\log}$ -sum of the weights of all the paths  $\pi_1 \pi_2 \pi_3$  from  $I$  to  $F$  with a sub-path  $\pi_2$  labeled with  $x$ :

$$-\log(c(L, x)) = \bigoplus_{\log, \pi_1 \pi_2 \pi_3 \in P(I, F), i[\pi_2]=x} \lambda(p[\pi_1]) \otimes w[\pi_1] \otimes w[\pi_2] \otimes w[\pi_3] \otimes \rho(n[\pi_3])$$

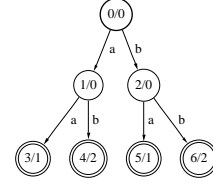
Using the distributivity of  $\otimes$  over  $\oplus_{\log}$  and the definition of the shortest-distances, this can be rewritten as:

$$-\log(c(L, x)) = \bigoplus_{\log, i[\pi_2]=x} d[p[\pi_2]] \otimes w[\pi_2] \otimes f[n[\pi_2]]$$

which proves the proposition.  $\square$

By proposition 1,  $-\log(c(L, x))$  can be computed in a straightforward manner by locating  $x$  in each path of  $L$  whose label contains  $x$ . This can be done with a simple depth-first search of  $L$  augmented in the following way: when a state previously visited is found, the search must continue from that state but only for paths of length less or equal to  $n - 1$ . The worst case complexity of that search is  $O(n|L|)$ .

For each subpath  $\pi$  found labeled with an  $n$ -gram sequence  $x$ , the weight  $d[p[\pi]] + w[\pi] + f[n[\pi]]$  is added to the current total weight of the sequence  $x$ . To store the weight associated to each sequence, we can maintain a hash table for the sequences  $x$  found



**Fig. 2.** Tree  $t$  accepting all bigram sequences over the alphabet  $\{a, b\}$  augmented with failure states. Within each circle, the first number denotes the state number, the second the failure state.

so far or use a tree  $t$  representing all  $n$ -gram sequences and store the weights at its leaves (see figure 2).

Since all operations other than the search are done in constant time, the total complexity of the computation of all  $c(L, x)$  for  $n$ -gram sequences  $x$  appearing in  $L$  using this algorithm is  $O(n|L|)$ . Only the necessary part of the tree  $t$ , that is the part corresponding to  $n$ -gram sequences found in  $L$ , needs to be constructed during the search.

The computation of  $K(L_1, L_2)$  for two lattices  $L_1$  and  $L_2$  can thus be done in the following way. We first compute  $-\log(c(L_1, x))$  for all  $n$ -gram sequences of  $L_1$  and construct just the part of the tree  $t$  needed. We then use the same tree to store the expected counts corresponding to  $L_2$  and keep only those leaves visited in that search for which  $c(L_1, x) \neq 0$  and sum  $-\log(c(L_1, x))$  and  $-\log(c(L_2, x))$  for those leaves, which gives exactly  $-\log(K(L_1, L_2))$ . This adds only constant time operations at each node of  $t$  visited in the second search. Thus, the total complexity of our algorithm for computing  $K(L_1, L_2)$  is  $O(n|L_1| + n|L_2|)$ .

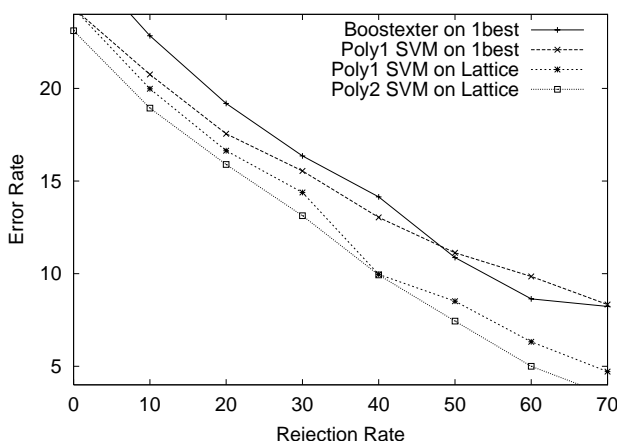
The use of the tree can be made more efficient by using the notion of *failure function* as in many efficient string-matching algorithms [1, 9]. Using failure functions, the computation of the kernel can be done in  $O(n + |L_1| + |L_2|)$  for lattices reduced to just one path. To each node  $q$  of that tree, we associate its *failure node*  $\phi[q]$  defined as follows. Let  $u$  be the unique string reaching node  $q$  from the root of the tree and let  $v$  be the longest suffix of  $u$  that is different from  $u$ . Then  $\phi[q]$  is the node of the tree obtained by reading  $v$  from the root. The failure nodes are used as follows. When reading an input string starting from the root node, the transition labeled with the next symbol is taken if such a transition exists, otherwise a transition labeled with that symbol from the failure node of the current node is taken. Failure functions can also be used to generalize the algorithm to the more general case of the computation of expected counts for elements of any finite set  $X \subseteq \Sigma^*$ .

### 3.4. Linear Lattice Kernels and SVMs

Our lattice kernel  $K$  is linear with respect to each of its lattice parameters. As a consequence, when it is used as the kernel for SVM classification, support vectors can be combined into a single vector by taking a linear combination of the support vectors. Indeed, let  $L_i, i = 1, \dots, N$ , be the support vectors determined by the SVM classification and let  $\alpha_i, i = 1, \dots, N$ , be the corresponding multipliers. More precisely, we can divide  $N$  into two sets  $N^+$  and  $N^-$  corresponding to the positive and negative multipliers. Then, the SVM classifier assigns the following weight to a lattice  $L$ :

$$\begin{aligned} \sum_{i \in N} \alpha_i K(L_i, L) &= \sum_{\alpha_i > 0} |\alpha_i| K(L_i, L) - \sum_{\alpha_i < 0} |\alpha_i| K(L_i, L) \\ &= K(L^+, L) - K(L^-, L), \quad \text{where} \\ L^+ &= \bigoplus_{\log, \alpha_i > 0} (-\log |\alpha_i| + L_i) \text{ and } L^- = \bigoplus_{\log, \alpha_i < 0} (-\log |\alpha_i| + L_i) \end{aligned}$$

This reduces the number of kernel computations during classification from  $N$  to just two. The price to pay is the size of the lattices



**Fig. 3.** Comparison of SVMs trained with trigram kernel applied to the lattice versus Boostexter, an otherwise superior classifier of similar tasks to the HMIHY system.

$L^+$  and  $L^-$  which are typically larger than any of the support lattices  $L_i$ . But,  $L^+$  and  $L^-$  can be optimized off-line using determinization and minimization of weighted automata [8]. In practice, this can significantly speed up classification when the linear kernel  $K$  is used directly.

Linear lattice classifiers also offer an efficient alternative to a common approach taken in spoken language recognition, which consists of applying the classifier to each of the  $N$ -best recognized sentences and to define as the voted output a weighted average of each classifier output. When the kernel is linear, a weighted average of the  $N$ -best classifications is nothing but a poor and inefficient approximation to the lattice kernel classification.

#### 4. EXPERIMENTAL RESULTS

We used the AT&T FSM Library [12] for the implementation of our lattice kernel and incorporated our algorithm in a general learning library for large-margin classification, LLAMA, [6]. LLAMA offers an optimal multi-class recombination of binary SVMs where kernels are optimized separately for each class.

We used the lattice kernel defined in the previous section for call-classification in the spoken language understanding (SLU) component of the AT&T *How May I Help You (HMIHY<sup>TM</sup>)* natural dialog system. In this system, users ask questions about their bill, calling plans, etc. The utterances are the responses to the first greeting: “Hello, This is AT&T. How May I Help You?”. The word error rate for these utterances with the current recognition system is 28.7% and the oracle accuracy, defined as the accuracy of the path in the lattice closest to the transcription, is 91.7%.

Within the SLU component, the objective is to classify the input telephone call into one of 38 classes (call-types and named entities), such as *Billing Credit*, or *Calling Plans* [5]. Each utterance may be assigned to several classes and it is considered to be an error if the highest scoring class is not one of these labels. In our experiments, we used 7,449 utterances as our training data and 2,228 utterances as our test data. The feature space corresponding to our lattice kernel is that of all possible trigrams over a vocabulary of 5,405 words. Training required just a few minutes on a single processor of a 1GHz Intel Pentium processor linux cluster with 2GB of memory and 256 KB cache.

Utterances that score lower than a given confidence level are rejected. We present our results in the form of a plot of the classification error as a function of the rejection rate. Figure 3 shows the result of the use of the trigram kernel enhanced with a 2-degree polynomial. In comparison to Boostexter, that otherwise has demonstrated superior performance on similar tasks to the HMIHY system [14], the lattice kernel brings a significant re-

duction of the one-error rate from 16.6% to 13.1% at a rejection level of 30%, a typical operation point for the task. The plot also demonstrates how the 1-best solution is clearly out-performed by the full lattice kernel solution (Poly1 SVM on 1-best versus Poly1 SVM on Lattice), and how a 2-degree polynomial applied to the lattice kernel further reduces the classification error (Poly1 SVM on Lattice versus Poly2 SVM on Lattice).

Note that combining an  $n$ -gram kernel with a polynomial of degree  $p$ , corresponds to SVM-classification using features of unordered gappy  $n \times p$ -grams. It is a topic for further research to determine the relationship between performance obtained from SVMs using true gappy  $n$ -grams and polynomial enhancements of lower order  $n$ -grams.

#### 5. CONCLUSION

We presented a principled approach to classification based on full lattices output by a recognizer. We introduced a general kernel for speech recognition lattices and efficient algorithms for its computation. Experimental results in a large and difficult spoken-dialog classification task using these kernels demonstrated their practicality and effectiveness.

The lattice kernel presented belongs to the general class of rational kernels [3]. More complex rational lattice kernels can be defined and computed using general weighted automata algorithms. Such kernels are likely to further improve the classification accuracy of spoken-dialog systems.

#### 6. REFERENCES

- [1] A. V. Aho and M. J. Corasick. Efficient string matching: An aid to bibliographic search. *Communication of the Association for Computing Machinery*, 18 (6):333–340, 1975.
- [2] B. E. Boser, I. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop of Computational Learning Theory*, volume 5, pages 144–152, Pittsburgh, 1992. ACM.
- [3] C. Cortes, P. Haffner, and M. Mohri. Rational Kernels. In *Advances in Neural Information Processing Systems (NIPS 2002)*, volume to appear, Vancouver, Canada, December 2002.
- [4] C. Cortes and V. N. Vapnik. Support-Vector Networks. *Machine Learning*, 20(3):273–297, 1995.
- [5] A. Gorin, G. Riccardi, and J. Wright. Automated natural spoken dialog. *IEEE Computer Magazine*, 35(4):51–56, April 2002.
- [6] P. Haffner, G. Tur, and J. Wright. Optimizing SVMs for complex Call Classification. In *Submitted to ICASSP’03*, 2002.
- [7] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*. Number 5 in EATCS Monographs on Theoretical Computer Science. Springer-Verlag, Berlin, Germany, 1986.
- [8] M. Mohri. Finite-state transducers in language and speech processing. *Computational Linguistics*, 23:2, 1997.
- [9] M. Mohri. String-Matching with Automata. *Nordic Journal of Computing*, 4(2):217–231, 1997.
- [10] M. Mohri. Semiring Frameworks and Algorithms for Shortest-Distance Problems. *Journal of Automata, Languages and Combinatorics*, To appear, 2002.
- [11] M. Mohri, F. C. N. Pereira, and M. Riley. Weighted automata in text and speech processing. In *ECAI-96 Workshop, Budapest, Hungary*. ECAI, 1996.
- [12] M. Mohri, F. C. N. Pereira, and M. Riley. General-purpose Finite-State Machine Software Tools. <http://www.research.att.com/sw/tools/fsm>, AT&T Labs – Research, 1997.
- [13] F. C. N. Pereira and M. Riley. Speech recognition by composition of weighted finite automata. In E. Roche and Y. Schabes, editors, *Finite-State Language Processing*, pages 431–453. MIT Press, Cambridge, Massachusetts, 1997.
- [14] R. Schapire, M. Roichery, M. Rahim, and N. Gupta. Incorporating prior knowledge in boosting. In *Proc. of ICML*, 2002.
- [15] B. Scholkopf and A. Smola. *Learning with Kernels*. MIT Press: Cambridge, MA, 2002.
- [16] V. N. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, New-York, 1998.