Compiling Boostexter Rules into a Finite-state Transducer

Srinivas Bangalore
AT&T Labs–Research
180 Park Avenue
Florham Park, NJ 07932

Abstract

A number of NLP tasks have been effectively modeled as classification tasks using a variety of classification techniques. Most of these tasks have been pursued in isolation with the classifier assuming unambiguous input. In order for these techniques to be more broadly applicable, they need to be extended to apply on weighted packed representations of ambiguous input. One approach for achieving this is to represent the classification model as a weighted finite-state transducer (WFST). In this paper, we present a compilation procedure to convert the rules resulting from an AdaBoost classifier into an WFST. We validate the compilation technique by applying the resulting WFST on a call-routing application.

1 Introduction

Many problems in Natural Language Processing (NLP) can be modeled as classification tasks either at the word or at the sentence level. For example, part-of-speech tagging, named-entity identification supertagging[1], word sense disambiguation are tasks that have been modeled as classification problems at the word level. In addition, there are problems that classify the entire sentence or document into one of a set of categories. These problems are loosely characterized as semantic classification and have been used in many practical applications including call routing and text classification.

Most of these problems have been addressed in isolation assuming unambiguous (one-best) input. Typically, however, in NLP applications these modules are chained together with each module introducing some amount of error. In order to alleviate the errors introduced by a module, it is typical for a module to provide multiple weighted solutions (ideally as a packed representation) that serve as input to the next module. For example, a speech recognizer provides a lattice of possible recognition outputs that is to be annotated with part-of-speech and named-entities. Thus classification approaches need to be extended to be applicable on weighted packed representations of ambiguous input represented as a weighted lattice. The research direction we adopt here is to compile the model of a classifier into a weighted finite-state transducer (WFST) so that it can compose with the input lattice.

Finite state models have been extensively applied to many aspects of language processing including, speech recognition (Pereira and Riley, 1997), phonology (Kaplan and Kay, 1994), morphology (Koskenniemi, 1984), chunking (Abney, 1991; Bangalore and Joshi, 1999), parsing (Roche, 1999; Oflazer, 1999) and machine translation (Vilar et al., 1999; Bangalore and Riccardi, 2000). Finite-state models are attractive mechanisms for language processing since they (a) provide an efficient data structure for representing weighted ambiguous hypotheses (b) generally effective for decoding (c) associated with a calculus for composing models which allows for straightforward integration of constraints from various levels of speech and language processing.[2]

In this paper, we describe the compilation process for a particular classifier model into an WFST and validate the accuracy of the compilation process on a one-best input in a call-routing task. We view this as a first step toward using a classification model on a lattice input. The outline of the paper is as follows. In Section 2, we review the classification approach to resolving ambiguity in NLP tasks and in Section 3 we discuss the boosting approach to classification. In Section 4 we describe the compilation of the boosting model into an WFST and validate the result of this compilation using a call-routing task.

2 Resolving Ambiguity by Classification

In general, we can characterize all these tagging problems as search problems formulated as shown

[1] associating each word with a label that represents the syntactic information of the word given the context of the sentence.

[2] Furthermore, software implementing the finite-state calculus is available for research purposes.

in Equation (1). We notate Σ to be the input vocabulary, $\mathcal{Y}$ to be the vocabulary of n tags, an N word input sequence as W ($\in \Sigma^+$) and tag sequence as T ($\in \mathcal{Y}^+$). We are interested in T^*, the most likely tag sequence out of the possible tag sequences (T) that can be associated to W.

$$T^* = \underset{T}{argmax}\ P(T|W) \tag{1}$$

Following the techniques of Hidden Markov Models (HMM) applied to speech recognition, these tagging problems have been previously modeled indirectly through the transformation of the Bayes rule as in Equation 2. The problem is then approximated for sequence classification by a kth-order Markov model as shown in Equation (3).

$$T^* = \underset{T}{argmax}\ P(W|T)P(T) \tag{2}$$

$$\hat{T} = \underset{T}{argmax} \prod_{i=1}^{N} P(w_i \mid t_i)P(t_i \mid t_{i-1} \ldots t_{i-k-1}) \tag{3}$$

Although the HMM approach to tagging can easily be represented as a WFST, it has a drawback in that the use of large contexts and richer features results in sparseness leading to unreliable estimation of the parameters of the model.

An alternate approach to arriving at T^* is to model Equation 1 directly. There are many examples in recent literature (Breiman et al., 1984; Freund and Schapire, 1996; Roth, 1998; Lafferty et al., 2001; McCallum et al., 2000) which take this approach and are well equipped to handle large number of features. The general framework for these approaches is to learn a model from pairs of associations of the form (x_i, y_i) where x_i is a feature representation of W and y_i ($\in$ $\mathcal{Y}$) is one of the members of the tag set. Although these approaches have been more effective than HMMs, there have not been many attempts to represent these models as a WFST, with the exception of the work on compiling decision trees (Sproat and Riley, 1996). In this paper, we consider the boosting (Freund and Schapire, 1996) approach (which outperforms decision trees) to Equation 1 and present a technique for compiling the classifier model into a WFST.

3 Boostexter

Boostexter is a machine learning tool which is based on the boosting family of algorithms first proposed in (Freund and Schapire, 1996). The basic idea of boosting is to build a highly accurate classifier by combining many "weak" or "simple" base learner, each one of which may only be moderately accurate. A weak learner or a rule h is a triple $(p, \vec{\alpha}, \vec{\beta})$, which tests a predicate (p) of the input (x) and assigns a weight α_i ($i = 1, ..., n$) for each member (y) of $\mathcal{Y}$ if p is true in x and assigns a weight (β_i) otherwise. It is assumed that a pool of such weak learners $H = \{h\}$ can be constructed easily.

From the pool of weak learners, the selection the weak learner to be combined is performed iteratively. At each iteration t, a weak learner h_t is selected that minimizes a prediction error loss function on the training corpus which takes into account the weight w_t assigned to each training example. Intuitively, the weights encode how important it is that h_t correctly classifies each training example. Generally, the examples that were most often misclassified by the preceding base classifiers will be given the most weight so as to force the base learner to focus on the "hardest" examples. As described in (Schapire and Singer, 1999), Boostexter uses *confidence rated* classifiers h_t that output a real number $h_t(x, y)$ whose sign (-1 or +1) is interpreted as a prediction, and whose magnitude $|h_t(x)|$ is a measure of "confidence". The iterative algorithm for combining weak learners stops after a prespecified number of iterations or when the training set accuracy saturates.

3.1 Weak Learners

In the case of text classification applications, the set of possible weak learners is instantiated from simple n-grams of the input text (W). Thus, if χ_n is a function to produce all n-grams up to n of its argument, then the set of predicates for the weak learners is $P = \chi_n(W)$. For word-level classification problems, which take into account the left and right context, we extend the set of weak learners created from the word features with those created from the left and right context features. Thus features of the left context (ϕ^i_L), features of the right context (ϕ^i_R) and the features of the word itself ($\phi^i_{w_i}$) constitute the features at position i. The predicates for the pool of weak learners are created from these set of features and are typically n-grams on the feature representations. Thus the set of predicates resulting from the word level features is $H_W = \cup_i \chi_n(\phi^i_{w_i})$, from left context features is $H_L = \cup_i \chi_n(\phi^i_L)$ and from right context features is $H_R = \cup_i \chi_n(\phi^i_R)$. The set of predicates for the weak learners for word level classification problems is: $H = H_W \cup H_L \cup H_R$.

3.2 Decoding

The result of training is a set of selected rules $\{h_1, h_2, \ldots, h_N\}$ ($\subseteq$ H). The output of the final classifier is $F(x, y) = \sum_{t=1}^{N} h_t(x, y)$, i.e. the sum of confidence of all classifiers h_t. The real-valued predictions of the final classifier F can be converted

into probabilities by a logistic function transform; that is

$$P(y|x) = \frac{e^{F(x,y)}}{\sum_{y' \in \mathcal{Y}} e^{F(x,y')}} \quad (4)$$

Thus the most likely tag sequence T^* is determined as in Equation 5, where $P(t_i|\phi^i_L, \phi^i_R, \phi^i_{w_i})$ is computed using Equation 4.

$$T^* = \underset{T}{argmax} \prod_{i=1}^{N} P(t_i|\phi^i_L, \phi^i_R, \phi^i_{w_i}) \quad (5)$$

To date, decoding using the boosted rule sets is restricted to cases where the test input is *unambiguous* such as strings or words (not word graphs). By compiling these rule sets into WFSTs, we intend to extend their applicability to packed representations of ambiguous input such as word graphs.

4 Compilation

We note that the weak learners selected at the end of the training process can be partitioned into one of three types based on the features that the learners test.

- h_W: test features of the word
- h_L: test features of the left context
- h_R: test features of the right context

We use the representation of context-dependent rewrite rules (Johnson, 1972; Kaplan and Kay, 1994) and their weighted version (Mohri and Sproat, 1996) to represent these weak learners. The (weighted) context-dependent rewrite rules have the general form

$$\phi \rightarrow \psi \mid \gamma \_ \delta \quad (6)$$

where ϕ, ψ, γ and δ are regular expressions on the alphabet of the rules. The interpretation of these rules are as follows: Rewrite ϕ by ψ when it is preceded by γ and followed by δ. Furthermore, ψ can be extended to a rational power series which are weighted regular expressions where the weights encode preferences over the paths in ψ (Mohri and Sproat, 1996).

Each weak learner can then be viewed as a set of weighted rewrite rules mapping the input word into each member y_i ($\in \mathcal{Y}$) with a weight α_i when the predicate of the weak learner is true and with weight β_i when the predicate of the weak learner is false. The translation between the three types of weak learners and the weighted context-dependency rules is shown in Table 1[3].

We note that these rules apply left to right on an input and do not repeatedly apply at the same point in an input since the output vocabulary $\mathcal{Y}$ would typically be disjoint from the input vocabulary Σ.

We use the technique described in (Mohri and Sproat, 1996) to compile each weighted context-dependency rules into an WFST. The compilation is accomplished by the introduction of context symbols which are used as markers to identify locations for rewrites of ϕ with ψ. After the rewrites, the markers are deleted. The compilation process is represented as a composition of five transducers.

The WFSTs resulting from the compilation of each selected weak learner (λ_i) are unioned to create the WFST to be used for decoding. The weights of paths with the same input and output labels are added during the union operation.

$$\Lambda = \cup_i \lambda_i \quad (7)$$

We note that the due to the difference in the nature of the learning algorithm, compiling decision trees results in a *composition* of WFSTs representing the rules on the path from the root to a leaf node (Sproat and Riley, 1996), while compiling boosted rules results in a *union* of WFSTs, which is expected to result in smaller transducers.

In order to apply the WFST for decoding, we simply compose the model with the input represented as an WFST (λ_x) and search for the best path (if we are interested in the single best classification result).

$$y^* = BestPath(\lambda_x \circ \Lambda) \quad (8)$$

We have compiled the rules resulting from boostexter trained on transcriptions of speech utterances from a call routing task with a vocabulary ($|\Sigma|$) of 2912 and 40 classes ($n = 40$). There were a total of 1800 rules comprising of 900 positive rules and their negative counterparts. The WFST resulting from compiling these rules has a 14372 states and 5.7 million arcs. The accuracy of the WFST on a random set of 7013 sentences was the *same* (85% accuracy) as the accuracy with the decoder that accompanies the boostexter program. This validates the compilation procedure.

5 Conclusions

Classification techniques have been used to effectively resolve ambiguity in many natural language

[3]For ease of exposition, we show the positive and negative sides of a rule each resulting in a context dependency rule. However, we can represent them in the form of a single context dependency rule which is ommitted here due to space constraints.

Type of Weak Learner	Weak Learner	Weighted Context Dependency Rule
h_W:	if WORD==w then $y_i : \alpha_i$ else $y_i : \beta_i$	$w \rightarrow \alpha_1 y_1 + \alpha_2 y_2 \ldots + \alpha_n y_n \mid \_\ \_\ \_$ $(\Sigma - w) \rightarrow \beta_1 y_1 + \beta_2 y_2 \ldots + \beta_n y_n \mid \_\ \_\ \_$
h_L:	if LeftContext==w then $y_i : \alpha_i$ else $y_i : \beta_i$	$\Sigma \rightarrow \alpha_1 y_1 + \alpha_2 y_2 \ldots + \alpha_n y_n \mid w\ \_\ \_$ $\Sigma \rightarrow \beta_1 y_1 + \beta_2 y_2 \ldots + \beta_n y_n \mid (\Sigma - w)\ \_\ \_$
h_R:	if RightContext==w then $y_i : \alpha_i$ else $y_i : \beta_i$	$\Sigma \rightarrow \alpha_1 y_1 + \alpha_2 y_2 \ldots + \alpha_n y_n \mid \_\ \_\ w$ $\Sigma \rightarrow \beta_1 y_1 + \beta_2 y_2 \ldots + \beta_n y_n \mid \_\ \_\ (\Sigma - w)$

Table 1: Translation of the three types of weak learners into weighted context-dependency rules.

processing tasks. However, most of these tasks have been solved in isolation and hence assume an unambiguous input. In this paper, we extend the utility of the classification based techniques so as to be applicable on packed representations such as word graphs. We do this by compiling the rules resulting from an AdaBoost classifier into a finite-state transducer. The resulting finite-state transducer can then be used as one part of a finite-state decoding chain.

References

S. Abney. 1991. Parsing by chunks. In Robert Berwick, Steven Abney, and Carol Tenny, editors, *Principle-based parsing*. Kluwer Academic Publishers.

S. Bangalore and A. K. Joshi. 1999. Supertagging: An approach to almost parsing. *Computational Linguistics*, 25(2).

S. Bangalore and G. Riccardi. 2000. Stochastic finite-state models for spoken language machine translation. In *Proceedings of the Workshop on Embedded Machine Translation Systems*.

L. Breiman, J.H. Friedman, R.A. Olshen, and C.J. Stone. 1984. *Classification and Regression Trees*. Wadsworth & Brooks, Pacific Grove, CA.

Y. Freund and R. E. Schapire. 1996. Experiments with a new boosting alogrithm. In *Machine Learning: Proceedings of the Thirteenth International Conference*, pages 148–156.

C.D. Johnson. 1972. *Formal Aspects of Phonological Description*. Mouton, The Hague.

R. M. Kaplan and M. Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics*, 20(3):331–378.

K. K. Koskenniemi. 1984. *Two-level morphology: a general computation model for word-form recognition and production*. Ph.D. thesis, University of Helsinki.

J. Lafferty, A. McCallum, and F. Pereira. 2001. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *In Proceedings of ICML*, San Francisco, CA.

A. McCallum, D. Freitag, and F. Pereira. 2000. Maximum entropy markov models for information extraction and segmentation. In *In Proceedings of ICML*, Stanford, CA.

M. Mohri and R. Sproat. 1996. An efficient compiler for weighted rewrite rules. In *Proceedings of ACL*, pages 231–238.

K. Oflazer. 1999. Dependency parsing with an extended finite state approach. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, Maryland, USA, June.

F.C.N. Pereira and M.D. Riley. 1997. Speech recognition by composition of weighted finite automata. In E. Roche and Schabes Y., editors, *Finite State Devices for Natural Language Processing*, pages 431–456. MIT Press, Cambridge, Massachusetts.

E. Roche. 1999. Finite state transducers: parsing free and frozen sentences. In András Kornai, editor, *Extended Finite State Models of Language*. Cambridge University Press.

D. Roth. 1998. Learning to resolve natural language ambiguities: A unified approach. In *Proceedings of AAAI*.

R.E. Schapire and Y. Singer. 1999. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, December.

R. Sproat and M. Riley. 1996. Compilation of weighted finite-state transducers from decision trees. In *Proceedings of ACL*, pages 215–222.

J. Vilar, V.M. Jiménez, J. Amengual, A. Castellanos, D. Llorens, and E. Vidal. 1999. Text and speech translation by means of subsequential transducers. In András Kornai, editor, *Extened Finite State Models of Language*. Cambridge University Press.