Raising the Bar: Stacked Conservative Error Correction Beyond Boosting

Dekai WU[*1], **Grace NGAI**[†2], **Marine CARPUAT**[*]

[*] Human Language Technology Center
HKUST
Department of Computer Science
University of Science and Technology
Clear Water Bay, Hong Kong
{dekai, marine}@cs.ust.hk

[†] Hong Kong Polytechnic University
Department of Computing
Kowloon
Hong Kong
csgngai@polyu.edu.hk

Abstract

We introduce a conservative error correcting model, Stacked TBL, that is designed to improve the performance of even high-performing models like boosting, with little risk of accidentally degrading performance. Stacked TBL is particularly well suited for corpus-based natural language applications involving high-dimensional feature spaces, since it leverages the characteristics of the TBL paradigm that we appropriate. We consider here the task of automatically annonating named entities in text corpora. The task does pose a number of challenges for TBL, to which there are some simple yet effective solutions. We discuss the empirical behavior of Stacked TBL, and consider evidence that despite its simplicity, more complex and time-consuming variants are not generally required.

1. Setting the Bar: Introduction

In this paper we develop a general stacking-based method called *Stacked TBL* (STBL) that error-corrects the output of a boosting model that is already highly tuned. We deploy TBL in an unconventional fashion, and discuss motivation and evidence to support its use. Several modifications to the traditional TBL procedure are required, but the revised procedure remains relatively simple.

To demonstrate the applicability of STBL, we construct a base model trained using the AdaBoost boosting algorithm (Freund and Schapire, 1997). Boosting has acquired a superior reputation for error driven learning of ensemble models and, when used in corpus-based NLP systems, typically finds a place as the ultimate stage. Two of the best-performing three teams in the CoNLL-2002 Named Entity Recognition shared task evaluation used boosting as their base system (Carreras et al., 2002; Wu et al., 2002). However, we have found that, like all learning models, even boosting models can and do reach certain limits that other models are less susceptible to. This holds even after careful feature engineering to compensate is carried out. We are thus driven to investigate the problem of correcting errors *after* boosting has done its best. This establishes a high bar for any model stacked on the boosting base model, because it is difficult to correct the few remaining errors without also accidentally undoing correct classifications at the same time.

In the following sections, we first define the particular stacking approach we will use. Subsequently we describe our model in detail, and analyze issues that arise from repurposing TBL for this task. We discuss the principles behind our proposed solutions, and demonstrate the method's empirical behavior. Finally, we consider evidence that more complicated and time-consuming alternative variants of STBL are unnecessary in practice.

2. Piping versus Stacking

Stacking has become widely used since its introduction a decade ago (Wolpert, 1992). However, a few words of clarification on stacking are in order since it is an extremely general concept, that is often confusingly used to lump together various approaches that are in fact methodologically quite different.

The major division of stacking approaches is between (1) those employing multiple heterogenous base learning models, and (2) those employing a single base learning model. The former case is a general alternative to simple voting among heterogenous base models.

To avoid confusion we will use the term *piping* for the latter case, i.e., stacking with a single base learning model. In this paper we will restrict our attention to piping, specifically using a boosting base model.

Usage of piping has two common subcases. First, piping provides a kind of arcing framework (Breiman, 1998), and as such is an alternative to simple bagging models. As an arcing procedure, however, its use appears to have been eclipsed in recent years by boosting.

Second, for some tasks, piping is effective as a sequentially chained error correcting ensemble. The investigation in this paper falls in this category.

Piped classifiers differ from cascaded classifiers (e.g., Alpaydin (1998)) in several important respects, one of which is that in cascading, confidence scores are not assigned to the predictions of the earlier classifiers. A cascaded classifier only attempts to classify examples that earlier stages have voluntarily passed on; it does not identify where errors are likely to have occurred and therefore is not an error corrector *per se*.

Piping can, however, be viewed as a *combination* of cascading and confidence prediction. In this view, the error correcting stage is responsible for both predicting the con-

[1]The author would like to thank the Hong Kong Research Grants Council (RGC) for supporting this research in part through research grants RGC6083/99E, RGC6256/00E, and DAG03/04.EG09.

[2]The author would like to thank the Hong Kong Polytechnic University for supporting this research in part through research grants A-PE37 and 4-Z03S.

fidence on each example that was output by the previous stage, as well as performing corrections on low-confidence examples. In fact, to be more precise, it is not actually necessary for the error correcting stage to predict an *absolute* confidence score for the base model's output. It merely needs to predict with high confidence when the error corrector's confidence score is *relatively* higher than the base model's.

Piping, broadly interpreted, is widespread in corpus-based NLP. An example of piping in an NER application very similar to the experiments described later in this paper is Florian (2002), who used a TBL base model piped to a forward-backward decoder model. However, narrowly interpreted piping, which employs correct and sound stacking training procedures, is rather less common, perhaps especially in NLP work.

3. Raising the Bar: Repurposing TBL for Error Correction

Certain kinds of models are better suited to error correction than others. This is particularly true when the performance of the base model is already high. The error corrector model must (1) have characteristics that vary sufficiently from the base model so that the corrector will make a significant difference, and (2) be excellent at "leaving well enough alone" so as not to miscorrect the already highly accurate predictions from the boosting model.

We consider transformation-based learning to be a reasonable candidate for error correction, and we call this model *Stacked TBL* (STBL). The odd thing is that although it is inherently an error-correcting paradigm, TBL has not received much attention as an error-corrector for models that already achieve high performance. Instead, TBL is traditionally used by itself, trained by correcting the output of relatively "dumb" base models. Even less attention has been given to using TBL as an error corrector for high-performance models.[1]

Nevertheless, TBL appeared to fit the bill since (1) its sequential processing characteristics are very different from the boosting model's, and (2) it can be modified to leave well enough alone, as we shall see in the subsequent section.

Transformation-based learning was first introduced by Brill (1995) for part-of-speech tagging, and it has since been applied to a wide range of corpus-based NLP tasks, including parsing (Brill, 1996), noun phrase chunking (Ramshaw and Marcus, 1999), phrase chunking (Florian et al., 2000), and dialog act tagging (Samuel et al., 1998). It is a flexible model which is easily extensible to various tasks, and it has the advantage of being able to achieve state-of-the-art performance with a small set of perspicious rules.

In some ways, TBL is similar to boosting in that it is an iterative process in which each iteration targets the residual error from previous iterations. A traditional TBL system is trained using the following algorithm:

1. Create an initial assignment of classifi cations using simple statistics.
2. Generate all productive rules according to a set of allowable templates.
3. For each rule:
   (a) Apply to a copy of the current state of the training corpus.
   (b) Score the result of the rule application with some objective function f.
4. Select the rule with the best score.
5. Stop if the score is smaller than some pre-set threshold Th.
6. Transform the current state of the training corpus by applying the rule to it.
7. Repeat from Step 2.

[1]Except, of course, when correcting TBL models themselves.

The output of the TBL is therefore an ordered list of rules, learned in a greedy fashion and scored according to the objective function, that progressively improve upon the performance of the learning algorithm on the training set. During the evaluation phase, the test set is initialized with the same initial class assignment. The rules are then applied, in the order they were learned, to the evaluation set. The final classification of a sample is then the classification attained when all the rules have been applied.

To repurpose TBL for our needs, we first need to simply replace Step 1 with the classification labels that are output by the AdaBoost.MH base model, instead of some trivial classifier based on simple statistics.

Unfortunately, this seemingly trivial change is not as harmless as it seems. It actually has the adverse effect of removing one of the strongest biases that ordinarily force TBL to generalize. This is because TBL typically relies on the fact that its input data is initially very poorly labeled, providing many examples to drive rule learning.

On the other hand, since our AdaBoost.MH base model already produces very high accuracy classification labels, there is very sparse data for TBL to work with. If used in this naive fashion, this would causes STBL to go astray easily.

For this reason it is an essential step in STBL to correctly generate n-fold cross validation partition sets via the base model (AdaBoost.MH in this case), as more traditional strict stacking models do. This can be a time-consuming step and, unfortunately, in NLP piping models, is more often than not omitted. Traditional stacking models do this primarily to avoid wasting data. This concern applies to STBL as well. However, there is an even more important factor for STBL in NLP applications because, in the type of high-dimensional feature spaces and data locality in training sets common to the NLP problems, ordinary partitioning creates incorrect strong biases that lead immediately to overfitting.

4. You-Break-It-You-Bought-It Pruning

On top of the sparse data problem just described, STBL faces a second problem. Unlike the case of cascaded classifiers, we have no information on the certainty of the base model's predictions, which would help the error corrector to be more selective in making corrections. Thus to avoid over-eagerly miscorrecting the base model's predictions, the error correction model must have extremely high confidence in any changes it makes. In other words, the error corrector must be extremely sure that it will not break anything that it picks up.

We therefore introduce a conservative method that we call *you-break-it-you-bought-it pruning,* which post-prunes the rules of a learned STBL model to err on the side of caution when making transforms to the current state.

The idea was inspired by the observation that the traditional $(\text{good} - \text{bad})$ objective function for rule scoring that TBL uses selects rules based on the recall of their outputs on the gold standard classifications, and does not take into account their precision. As a result, any given rule is allowed to make large numbers of mistakes, as long as it compensates with enough corrections. This does not mesh well with our requirement, which is that the postclassifier should avoid making errors at all costs.

To correct for this behavior, STBL includes a post-pruning algorithm to discard potentially errorful rules. After a transformation-based learner is trained in the usual manner, we evaluate *each* learned rule *independently* on the training set and record the number of mistakes and corrections resulting from the transformations it proposes. The rules are then pruned according to the following two principles:

- *Extremely risk-averse:* We require the postclassifier to be extremely cautious about not making any mistakes along with the corrections. Assuming that the training and testing corpora are drawn from the same distribution, if a rule makes a mistake on the training corpus, it is more likely to make one on the test corpus. Therefore, any rule that makes a mistake on the training set (bad > 0) is discarded.

- *Extremely high expectations:* We also require the postclassifier to have high confidence that any correction it makes is a valid correction, and not the result of noise aberration in the data. Therefore, any rule that does not make at least a certain threshold number of corrections (good $<$ T) is discarded.

It can be argued that as transformation-based learning rules are evaluated on the entire data set, post-pruning the rules in this manner ignores any possible interaction between them; in addition, any "rule chains" that were learned by TBL may be broken as rules are deleted from the list. However, this is not as big a problem as it appears to be at first blush. As TBL is error-driven, it focuses on samples with erroneous guess classifications on which rules are learned. When TBL is repurposed as an error corrector for another learning algorithm which has already achieved a good performance, errors are far and few in between, and hence the opportunities for rule interaction are decreased.

5. Experiments

To illustrate our hypothesis, we performed a series of experiments on named entity recognition using a set of English data from the Reuters corpus. The data consisted of two subsets in which named entities had been manually annotated.

The boosting and transformation-based systems used in our experiments were both constructed using publicly available software. For the boosting, we used Boostexter (Schapire and Singer, 2000), which implements boosting on top of decision stumps (decision trees of depth one). The transformation-based model was built with the fnTBL toolkit (Ngai and Florian, 2001), which implements several optimizations in rule learning to drastically speed up the time needed for training.

6. Results

Table 1 presents the results of the boosting-only and piped models on named entity recognition. The top row shows the performance of the boosting system; it can be seen that boosting sets the bar very high for TBL to improve upon. The rest of the rows show the results of stacking TBL on top of boosting:

- STBL (Completion): The TBL system was trained to completion (i.e. Th in Step 5 = 0, Error rate on training set = 0%).

- STBL (Stopped): The TBL system stopped training after the score of the rules had dropped below a threshold of 3, which is a commonly-set threshold in TBL training.

- STBL (Pruned): The TBL rules were post-pruned with the method from Section 4.

The first item that leaps to the eye is the fact that the naive method of piping the boosting model to TBL in fact *degrades* the performance. This turns out to hold regardless of whether TBL is trained to completion or stopped using an early stopping criterion, and can be attributed to overfitting the training set as discussed earlier.

Post-pruning the TBL rules, on the other hand, has a positive effect—the performance of the combined system improves beyond the already high performance of the plain boosting model. Empirically this has proved to be a reliable, robust effect; we have witnessed the same pattern exhibited on numerous data sets under various training conditions.

System	**Precision**	**Recall**	**F-Measure**$_1$
Boosting alone	95.01%	93.98%	94.49
Boosting + STBL (Completion)	92.78%	93.78%	93.28
Boosting + STBL (Stopped)	93.94%	94.67%	94.31
Boosting + STBL (Pruned)	94.13 %	95.16%	94.64

Table 1: Results of Boosting-only and Piped Models on Named-Entity Recognition

System	**Precision**	**Recall**	**F-Measure**$_1$
m partitioned sets from boosting + STBL (Pruned)	94.13 %	95.16%	94.64
m partitioned sets from boosting + m STBL (Pruned) + Voting	95.21%	94.13%	94.67
m partitioned sets from boosting + m STBL (Stopped) + Union-of-rules (Pruned)	94.14%	95.18%	94.66

Table 2: Other modes of Piping: Voting on n-fold-trained TBL and Rule Consolidation

7. Discussion

Training the STBL model remains a remarkably simple yet effective procedure that consumes all the partition sets from the AdaBoost.MH base model at once.

It has been suggested that the simple way in which STBL combines the data from the partition sets is oversimplistic and could be improved upon. We do not believe this is generally the case in practice. To test this, we constructed several alternative models with more sophisticated mechanisms for combining the base model partition sets. In one alternative, multiple pruned STBL models are trained on the partition sets, and they vote in the final classifier. In another alternative, multiple STBL models are trained using early stopping criteria (compensating for the increased number of models to train and increasing the bias toward common rules).

Our experiments indicate that there is no need to increase the complexity and training time by using the alternative models. The results, an example of which is given in Table 2, consistently show no statistically significant improvement over the STBL model.

One interesting piece of information to investigate would be the effect our aggressive, high-precision you-break-it-you-bought-it pruning operation had on the rule set. As might be expected, a large number of rules did not pass the requirements and were deleted—out of a total of 351 rules, only 37 survived the pruning.

Figure 1 shows the performance achieved by applying each successive rule to the system for both the pruned and unpruned rule list. (Only the first 40 rules were included from the unpruned rule list for comparison purposes.)

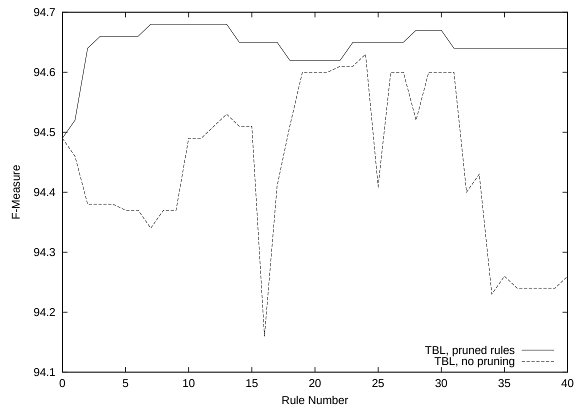

Figure 1: STBL with and without Rule Pruning.

It is immediately apparent from the figure that the unpruned rules lead to massive unpredictable fluctuation in performance. This is a good illustration of the fact that an STBL, naively trained without careful attention to the input data, will overfit and go astray very easily. In contrast, the performance of the pruned system exhibits far less fluctuation, behaves more predictably, and stays within a much narrower range. The result is a much more robust system which achieves a better performance.

A second item of note is that for the pruned system, much of the performance improvement is achieved within the first few rules. (Note that these are not necessarily the first rules that were learned—the fact that performance curves for the pruned and unpruned system diverge immediately at the beginning indicate that the rule sets are different.) We did not exploit this fact in our experiments (since doing so would have entailed looking at the test set performance), but it does give a direction for further rule pruning.

8. Conclusion and Future Directions

We have introduced a conservative model, Stacked TBL, that can reliably improve the performance of even high-performing models like boosting on high-dimensional tasks such as named entity recognition. TBL has characteristics that are well suited to this challenge. We have discussed the issues that arise from repurposing TBL for this task, and described simple yet effective techniques to overcome them. We also provided evidence that more complex and time-consuming variants are not generally necessary, and showed typical performance curves illustrating why Stacked TBL performs well as a conservative error corrector. The method has been applied within the context of a larger NER system (Wu et al., 2003).

We have begun to investigate a related but even more radical error correction model called NTPC (N-fold Templated Piped Correction), based on similar but more comprehensive empirical analyses across a larger range of tasks (Wu et al., 2004). To date, all results further confirm that templated error correcting models represent a remarkably underutilized paradigm that, in practice, consistently helps to combat the weaknesses of many commonly used corpus-based NLP models.

9. References

Ethem Alpaydin. 1998. Techniques for combining multiple learners. In Ethem Alpaydin, editor, *Proceedings of Engineering of Intelligent Systems*, volume 2, pages 6–12. ICSC Press.

Leo Breiman. 1998. Arcing classifi ers. *The Annals of Statistics*, 26(3):801–849.

Eric Brill. 1995. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*, 21(4):543–565.

Eric Brill. 1996. Learning to parse with transformations. In Harry Bunt and Masaru Tomita, editors, *Recent Advances in Parsing Technology*. Kluwer.

Xavier Carreras, Lluís Màrques, and Lluís Padró. 2002. Named entity extraction using AdaBoost. In *Proceedings of CoNLL-2002*, pages 167–170. Taipei.

Radu Florian, John C. Henderson, and Grace Ngai. 2000. Coaxing confi dence from an old friend: Probabilistic classifi cations from transformation rule lists. In *Proceedings of EMNLP-SIGDAT 2000*, Hong Kong, October. Association of Computational Linguistics.

Radu Florian. 2002. Named entity recognition as a house of cards: Classifi er stacking. In *Proceedings of CoNLL-2002*, pages 175–178. Taipei.

Yoram Freund and Robert E. Schapire. 1997. A decision-theoretic generalization of on-line learning and an application to boosting. In *Journal of Computer and System Sciences, 55(1)*, pages 119–139.

Grace Ngai and Radu Florian. 2001. Transformation-based learning in the fast lane. In *Proceedings of NAACL '01*, pages 40–47, Pittsburgh, PA. ACL.

Lance Ramshaw and Mitchell Marcus. 1999. Text chunking using transformation-based learning. In Susan Armstrong, Kenneth W. Church, Pierre Isabelle, Sandra Manzi, Evelyne Tzoukermann, and David Yarowsky, editors, *Natural Language Processing Using Very Large Corpora*. Kluwer.

Ken Samuel, Sandra Carberry, and Vijay K. Shanker. 1998. Dialogue act tagging with transformation-based learning. In *Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics*, volume 2. Association of Computational Linguistics.

Robert E. Schapire and Yoram Singer. 2000. Boostexter: A boosting-based system for text categorization. *Machine Learning*, 2(3):135–168.

David Wolpert. 1992. Stacked generalization. *Neural Networks*, 5:241–259.

Dekai Wu, Grace Ngai, Marine Carpuat, Jeppe Larsen, and Yongsheng Yang. 2002. Boosting for named entity recognition. In *Proceedings of CoNLL-2002*, pages 195–198. Taipei.

Dekai Wu, Grace Ngai, and Marine Carpuat. 2003. A stacked, voted, stacked model for named entity recognition. In *Proceedings of CoNLL-2003*, pages 200–203. Edmonton, Canada.

Dekai Wu, Grace Ngai, and Marine Carpuat. 2004. N-fold Templated Piped Correction. In *Proceedings of IJCNLP-2004, First International Joint Conference on Natural Language Processing*. Hainan, China.