# N-fold Templated Piped Correction

**Dekai WU**[†1] dekai@cs.ust.hk

**Grace NGAI**[‡2] csgngai@polyu.edu.hk

**Marine CARPUAT**[†] marine@cs.ust.hk

[†] Human Language Technology Center
HKUST
Department of Computer Science
University of Science and Technology
Clear Water Bay, Hong Kong

[‡] Hong Kong Polytechnic University
Department of Computing
Kowloon
Hong Kong

## Abstract

We describe a broadly-applicable conservative error correcting model, **N-fold Templated Piped Correction** (NTPC), that consistently improves the accuracy of existing high-accuracy base models. Under circumstances where most obvious approaches actually reduce accuracy more than they improve it, NTPC nevertheless comes with little risk of accidentally degrading performance. NTPC is particularly well suited for natural language applications involving high-dimensional feature spaces, such as bracketing and disambiguation tasks, since its easily customizable template-driven learner allows efficient search over the kind of complex feature combinations that have typically eluded the base models. We show empirically that NTPC yields small but consistent accuracy gains on top of even high-performing models like boosting. We also give evidence that the various extreme design parameters in NTPC are indeed necessary for the intended operating range, even though they diverge from usual practice.

## 1 Introduction

In language processing tasks, situations frequently arise where (1) we have already trained a highly accurate model for classification and/or sequence recognition, (2) the model nevertheless cannot deal with some kinds of errors, because it does not consider complex conjunctions of many features (usually because the computational cost would be infeasible), and (3) we have some general idea as to what kinds of feature conjunctions might help. Such conditions are often found in tasks such as word sense disambiguation, phrase chunking, entity recognition, role labeling for understanding tasks, and the like.

We introduce a useful general technique called **N-fold Templated Piped Correction** or **NTPC** for robustly improving the accuracy of existing base models under such circumstances. Given that the base accuracy is already high, even small gains are desirable and difficult to achieve, particularly since it is difficult to correct the few remaining errors without also accidentally undoing correct classifications at the same time. NTPC has the virtues of consistently producing these small gains in accuracy, being straightforward to implement, and being applicable in a wide variety of situations.

To demonstrate the consistency and effectiveness of the method, we apply it to two tasks across four languages with very different characteristics, using an AdaBoost.MH base model already trained to high accuracy on named-entity datasets. This is typically representative of the kinds of language processing classification and sequence recognition tasks we are concerned with. Boosting was chosen for its superior reputation for error driven learning of ensemble models; along with maximum-entropy models and SVMs, it performs extremely well in language-independent named-entity recognition. Yet like all learning models, these base models can and do reach certain limits that other models are less susceptible to (despite which, boosting has typically been used for the final stage in NLP systems). This holds even after careful feature engineering to compensate is carried out. Error analysis for this task indicated that one of the major practical limitations of the model was that only a small number of conjunctions of features could be feasibly searched during boosting.

This type of phenomenon in fact occurs frequently, over a wide range of language processing tasks. Some systems attempt to compensate using *ad hoc* voting methods combining multiple base models, but this can lead to unpredictable results that often do not address the underlying causes of the remaining errors. The question is therefore how to systematically correct errors *after* a high-accuracy base model such as boosting has done its best.

[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.

NTPC combines a template-driven error correction learning mechanism with cross-validation style n-fold partitioning of datasets generated by the base model. The error correction mechanism possesses similarities in some theoretical respects to both decision lists and tranformation-based learning, but not with regard to their conventional uses. The n-fold partitioning setup is a form of stacking, but again not in a form typically used in NLP. We discuss the motivation and evidence for the necessity of the differences.

NTPC owes its leverage to (1) being able to explore a much wider range of conjunctive hypotheses than the base model, owing to its template-driven (and thus, template-constrained) hypothesis generator, (2) being able to observe right contexts not knowable to the base model, and (3) being extremely conservative via the combination of n-fold partitioning and zero error tolerance.

In the following sections, we first give the formal definition of the NTPC model. We then describe the experimental setup of the eight different tasks used to test NTPC, and discuss the relations to previous work. After presenting overall results, we identify and analyze some of the key theoretical characteristics that allow NTPC to perform better than other base models, and present the corresponding empirical confirmation to support these claims.

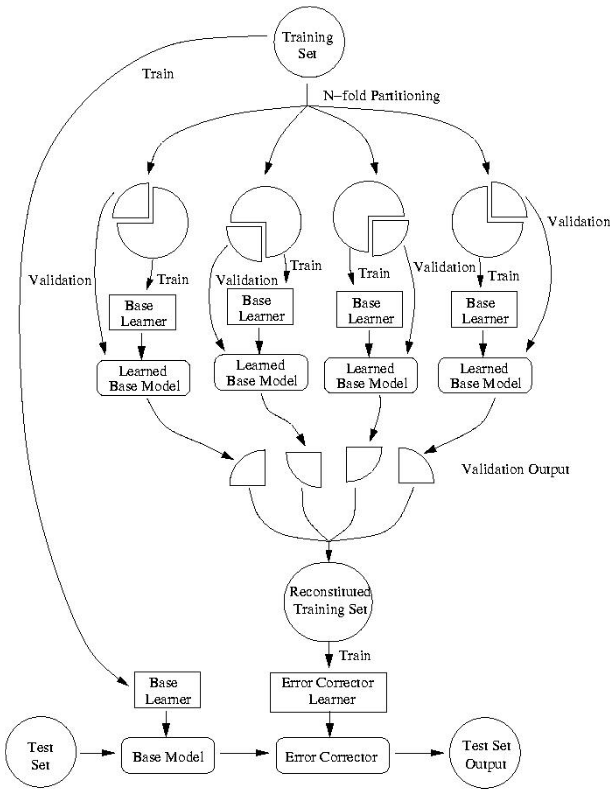

Figure 1: Piped architecture with n-fold partitioning.

## 2 Error Correction in Extreme Operating Ranges

Several conditions must be true for an error correction model to be effective at its task. This is even more the case when the base model is state-of-the-art and already high-performing. The error correction model must be:

- *Biased differently*: The corrector must be able to capture phenomena which were not learned by the base model – therefore it must have characteristics that vary significantly from the base model.

- *Extremely risk-averse*: The goal of an error corrector is to achieve performance *gains* on a base learner. Therefore, one of the basic requirements is that it should only correct existing errors, and not introduce any new ones by miscorrecting accurate predictions.

- *Extremely reliable*: The error corrector works in ranges where errors are far and few in between. As a result, it needs to be able to identify valid error patterns, as opposed to noise-induced abberations.

NTPC is designed to fulfill the above requirements. The following will give a description of the model and discuss the various design issues with respect to the foregoing design requirements.

The inputs to NTPC are (1) a set of rule templates which describe the types of rules that it is allowed to hypothesize, (2) a single base learning model (the example in this paper being an AdaBoost.MH model), and (3) an annotated training set.

Figure 1 shows the architecture of NTPC. The training set is partitioned $n$ times in order to train $n$ base models. Each base model is evaluated on its corresponding held-out validation set, and the labelled $n$ validation sets are then recombined to create the training set for *Error_Corrector_Leaner*. The Error Corrector learns a list of rules which are generated from a given set of templates:

$$\mathcal{R} = \{r | r \in \mathcal{H} \wedge \tau(r) > \tau_{\min} \wedge \epsilon(r) = 0\} \quad (1)$$

$$\tau(r) = \sum_{j=1}^{X} \sum_{r(x_j,\hat{y}_j) \neq \emptyset} \delta(r(x_j,\hat{y}_j), y_j) \quad (2)$$

$$\epsilon(r) = \sum_{j=1}^{X} \sum_{r(x_j,\hat{y}_j) \neq \emptyset} 1 - \delta(r(x_j,\hat{y}_j), y_j) \quad (3)$$

where $\mathcal{X}$ is a sequence of $X$ training examples $x_i$, $\mathcal{Y}$ is a sequence of reference labels $y_i$ for each example respectively, $\hat{\mathcal{Y}}$ is a sequence of labels $\hat{y}_i$ as predicted by the base model for each example respectively, $\mathcal{H}$ is the hypothesis space of valid rules implied by the templates, and $\tau_{\min}$ is a confidence threshold. $\tau_{\min}$ is set to a relatively high value (say 15), which implements the requirement of high reliability. $\mathcal{R}$ is subsequently sorted by the $\tau_i$ value of each rule $r_i$ into an ordered list of rules $\mathcal{R}^* = (r_0^*, \ldots, r_{i-1}^*)$.

The evaluation phase is depicted in the lower portion of Figure 1. The test set is first labeled by the base model. The error corrector's rules $r_i^*$ are then applied in the order of $R^*$ to the evaluation set. The final classification of a sample is then the classification attained when all the rules have been applied. This differs from the similar-spirited decision list model (Rivest, 1987).

## 3 Experiments

To verify the hypotheses underlying the design of NTPC, we performed a series of experiments applying NTPC to eight different named entity recognition (NER) models, for various tasks and languages. The data used was from the shared tasks of the CoNLL 2002 and 2003 conferences (Tjong Kim Sang, 2002)(Tjong Kim Sang and Meulder, 2003), which evaluated NER in Spanish, Dutch, English and German. The data consisted of two subsets in which named entities had been manually annotated.

### 3.1 Previous Work

Boosting (Freund and Schapire, 1997), at present one of the most popular machine learning techniques, is based on the idea that a set of many simple but moderately weak classifiers can be combined to create a single highly accurate strong classifier. It has the advantage of being able to handle large numbers of sparse features, many of which may be irrelevant or highly interdependent. This would make it appear to be well suited for NLP tasks which often exhibit these characteristics.

In our experiments, we construct a high-performance base model based on AdaBoost.MH, (Schapire and Singer, 2000), the multi-class generalization of the original boosting algorithm, which implements boosting on top of decision stump classifiers (decision trees of depth one).

Boosting has been successfully applied to several NLP problems. In these NLP systems boosting is typically used as the ultimate stage in a learned system. For example, Schapire and Singer (2000) applied it to Text Categorization while Escudero *et al.*(2000) used it to obtain good results on Word Sense Disambiguation. More closely relevant to the experiments described here in, 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, precedents for improving performance *after* boosting are few. At the CoNLL-2002 shared task session, Tjong Kim Sang (unpublished) described an experiment using voting to combine the NER outputs from the shared task participants which, predictably, produced better results than the individual systems. A couple of the individual systems were boosting models, so in some sense this could be regarded as an example. We began preliminary investigation of methods based on error correction for the CoNLL-2003 shared task (Wu *et al.*, 2003).

Tsukamoto *et al.*(2002) used piped AdaBoost.MH models for NER. Their experimental results were somewhat disappointing, but this could perhaps be attributable to various reasons including the feature engineering or not using cross-validation sampling in the stacking.

The AdaBoost.MH base model's high accuracy sets a high bar for error correction. Aside from brute-force *en masse* voting of the sort at CoNLL-2002 described above, we do not know of any existing post-boosting models that improve rather than degrade accuracy. We aim to further improve performance, and propose using a piped error corrector.

## 4 Results

Table 1 presents the results of the boosting-only base model versus the NTPC-enhanced models on the eight different named-entity recognition models, using different tasks and languages. For each task/language combination, the top row shows the base model (AdaBoost) result, and the bottom row shows the result of the piped system.

The evaluation uses the standard Precision/Recall/F-Measure metrics:

$$\begin{aligned}
\text{Precision} &= \frac{\text{num of correctly proposed NEs}}{\text{num of proposed NEs}} \\
\text{Recall} &= \frac{\text{num of correct proposed NEs}}{\text{num of gold standard NEs}} \\
\text{F-measure}_\beta &= \frac{(\beta^2 + 1) \times \text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}}
\end{aligned}$$

The results in the table show that boosting already sets the bar very high for NTPC to improve upon. Nevertheless, NTPC manages to achieve a performance improvement on *every* task/language combination. This holds true across all languages—from English, on which the baseline accuracy is high to begin with; to German, on which the boosting model performs the worst.

We now identitfy and discuss some of the key characteristics of NTPC that contribute to its effectiveness.

### 4.1 Templated Hypothesis Generation

One of the inputs to NTPC is a set of pre-defined templates. These templates are formed from conjunctions of basic features such as part-of-speech, lexical identity and gazetteer membership, and may be as simple as "lexical identity of the current word AND the part-of-speech of the previous word", or as complex as "capitalization information of the previous, current and next words AND the lexicon and gazetteer membership statuses of the current word AND the current (i.e. most recent) class label of the current word." The rule hypotheses are generated according to these templates, as such, the templates are usually motivated by linguistically informed expectations.

Table 1: NTPC consistently yields further F-measure gains on all eight different high-accuracy NER base models, across every combination of task and language.

| **Model** | **Task** | **Language** | **Experiment** | **Precision** | **Recall** | **F-Measure$_1$** |
|---|---|---|---|---|---|---|
| M1 | NE Bracketing | Dutch | M1 | 87.27 | 91.48 | 89.33 |
| | | | M1 + NTPC | **87.44** | **92.04** | **89.68** |
| M2 | NE Bracketing | English | M2 | 95.01 | 93.98 | 94.49 |
| | | | M2 + NTPC | **95.23** | **94.05** | **94.64** |
| M3 | NE Bracketing | German | M3 | **83.44** | 65.86 | 73.62 |
| | | | M3 + NTPC | 83.43 | **65.91** | **73.64** |
| M4 | NE Bracketing | Spanish | M4 | 89.46 | 87.57 | 88.50 |
| | | | M4 + NTPC | **89.77** | **88.07** | **88.91** |
| M5 | NE Classification + Bracketing | Dutch | M5 | 70.26 | 73.64 | 71.91 |
| | | | M5 + NTPC | **70.27** | **73.97** | **72.07** |
| M6 | NE Classification + Bracketing | English | M6 | 88.64 | 87.68 | 88.16 |
| | | | M6 + NTPC | **88.93** | **87.83** | **88.37** |
| M7 | NE Classification + Bracketing | German | M7 | **75.20** | 59.35 | 66.34 |
| | | | M7 + NTPC | 75.19 | **59.41** | **66.37** |
| M8 | NE Classification + Bracketing | Spanish | M8 | 74.11 | 72.54 | 73.32 |
| | | | M8 + NTPC | **74.43** | **73.02** | **73.72** |

The advantage of using hypotheses which are template-driven is that it allows the user to "tune" the system by writing templates which specifically target either the task at hand, or error patterns which are frequently committed by the base learner. They can also be used to prevent the error corrector from wasting time and memory by excluding rule templates which would be useless or overly trivial. In addition, these rule hypotheses are also often more complex and sophisticated than what the base models (in our case, decision stumps from AdaBoost.MH) can handle.

**Empirical Confirmation** To judge the contribution of templated hypothesis generation, we examine the top rules learned for each language. The following shows several example rules which are representative of those learned by NTPC.

- *German rule 2*: (if the current word is currently labeled as part of a PERSON name, but the word "pradesh" follows in one of the succeeding three words, make it part of a LOCATION name)

```
ne_0=I-PER
word:[1,3]=pradesh
=> ne=I-LOC
```

- *Spanish rule 3*: (if the current word and the previous word are both uppercased but don't start a new sentence, and the following word is lowercased, and the current word is not in the lexicon nor in the gazetteer and is currently not part of a named entity, make it part of an ORGANIZATION name)

```
wcaptype_0=noneed-firstupper
wcaptype_-1=noneed-firstupper
wcaptype_1=alllower
captypeLex_0=not-inLex
captypeGaz_0=not-inGaz
ne_0=O
=> ne=I-ORG
```

- *Dutch rule 1*: (if the current word is "de", labeled as part of a PERSON's name and is uppercased but is the first word in a sentence, it should not be part of a named-entity)

```
ne_0=I-PER
word_0=de
captype_0=need-firstupper
=> ne=O
```

The templates for these rules were all written with the base learner errors in mind, and thus contain highly complex conjunctions of features. It is a valid question to ask whether it is possible to add these conjunctive features to the base AdaBoost learner as additional decision stump features. This was indeed attempted, but AdaBoost was not able to handle the combinatorial explosion of feature-value pairs generated as a result.

## 4.2 Sensitivity to Right Context

One of NTPC's advantages over the base model is its ability to "look forward" to the right context. The problem for many NLP tagging and chunking models, where the unit of processing is a sentence, is that the text is processed from left-to-right, with the classifier deciding on the class label for each word before moving onto the next one. The result is that when features are extracted from the corpus for a particular word, the only class labels that can be extracted as features are those from the preceding (left-context) words. Since the words are labeled in order, the words that come later in the order (those in the right context) are not labeled yet, and as such, their labels cannot be used as features.

NTPC deals with this problem since the base model has already assigned a set of fairly accurate labels to all the words in the corpus. The error corrector has access to all these labels and can use them as word features as it deems necessary.

**Empirical Confirmation** Our experiments confirmed that NTPC's ability to include right context features into its rules helped it outperform the base model. On average, 2-3 of the top ten rules learned for each language were right-context sensitive. The following shows an example of such a rule:

- *Dutch rule 6*: (if the current word is currently labeled as part of a PERSON's name, and the next word does not contain any uppercase characters and is currently not part of any named entity, take the current word out of the PERSON's name)

```
ne_0=I-PER
ne_1=O
captype_0=alllower
=> ne=O
```

### 4.3 N-fold Piping

The n-fold partitioning and "reconstitution" of the training set for NTPC's error corrector is a crucial step for NTPC's error corrector. The highly accurate labels generated by the base model are not as trivial and harmless as they appear—in fact, their presence results in very sparse data for the error corrector to learn from, and makes it very hard for the error corrector to generalize. If the n-fold partitioning step were omitted from the NTPC system, it would cause the error corrector to go astray easily. This is unlike the case of, say, transformation-based tagging, in which the training set is partitioned just once, as the poor initial state of the data actually serves to provide a strong bias that forces the learner to generalize across many examples. For this reason, it is important to correctly generate the n-fold cross validation partition sets with the base model. However, this is a time-consuming step, which may explain why it seems to be omitted from NLP models.

**Empirical Confirmation** The n-fold piping is a complicated process and it is valid to ask whether this is actually necessary in practice. To test this, four experiments were performed, where the trained base model was used directly to relabel the training data. This data, together with the reference labels, was then provided to the error corrector as training data.

Figure 2 shows the results of the experiments. The stopping point for training (when $\tau(r) < \tau_{\min}$) is denoted by the short vertical bar on each NTPC performance curve. The high performance of the base learner *on its own training data* creates a very small $\tau(r)$ at the start of training process—and as a result, *zero* error correcting rules are learned. It is possible to ignore the $\tau_{\min}$ constraint and learn rules with very low $\tau(r)$ (the dashed lines show the performance of these rules). However, since these rules are mostly of dubious quality and also apply far and few in between, in most cases, they will not improve performance, and may even cause more errors to result—and it is not possible to reliably predict when a performance improvement will happen. In contrast, NTPC will *always* give a performance improvement.

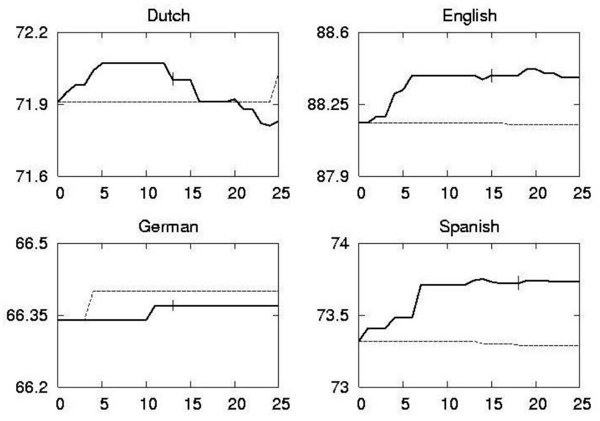

Figure 2: Performance improvement is not reliably obtained without n-fold partitioning. (x-axis = number of rules learned, y-axis = F-Measure; bold = NTPC, dashed = without n-fold partitioning)

### 4.4 Zero Error Tolerance

One of the most extreme decisions in the NTPC was the $\epsilon(r) = 0$ condition in *Error_Corrector_Learner*. In effect, this means that NTPC allows zero tolerance for noise and is overwhelmingly conservative about making any changes. The reason for this is, as an error *corrector*, NTPC has to be extremely careful not to introduce new errors. Since we have no information on the certainty of the base model's predictions, we assume that the training and testing corpora are drawn from the same distribution. This would mean that if a rule makes a mistake on the training corpus, it would be similarly likely to make one on the test corpus. Thus, to avoid over-eagerly miscorrecting the base model's predictions, the error corrector was designed to err on the side of caution and not make any corrections unless it has extremely high confidence that whatever it does will not cause any additional harm.

There are some structural similarities between NTPC and methods such as decision list learning (Rivest, 1987) and transformation-based learning (Brill, 1995), and some of the design decisions of NTPC may seem extreme when compared to them. However, these methods were not designed to be run on top of high-performing base models. A traditional rule list which is working on a very

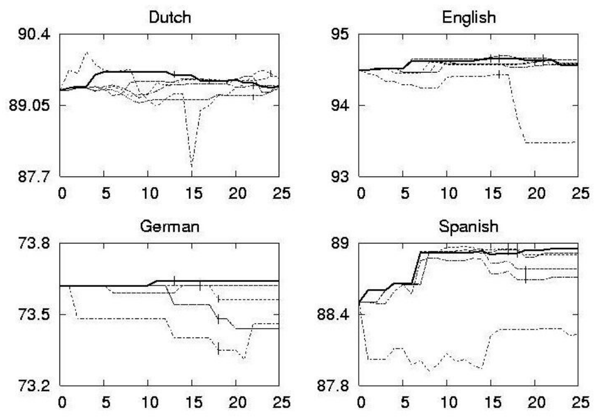

Figure 3: NTPC's zero tolerance condition yields less fluctuation and generally higher accuracy than the relaxed tolerance variations, in bracketing experiments. (x-axis = number of rules learned, y-axis = F-Measure; bold = NTPC, dashed = relaxed tolerance)

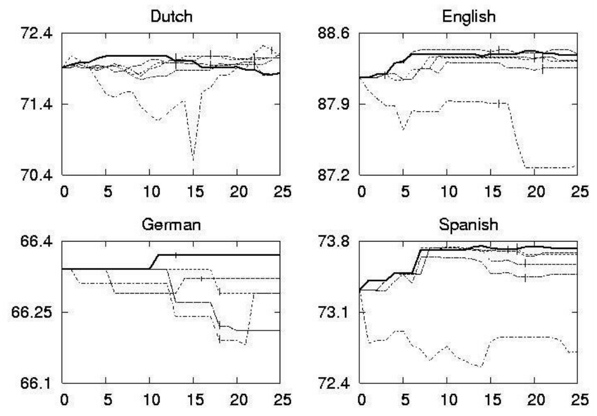

Figure 4: NTPC's zero tolerance condition yields less fluctuation and generally higher accuracy than the relaxed tolerance variations, in bracketing + classification experiments. (x-axis = number of rules learned, y-axis = F-Measure; bold = NTPC, dashed = relaxed tolerance)

poorly labelled data set may be able to justify trading off some corrections with some mistakes, provided that the overall change in accuracy is positive. NTPC, on the other hand, is designed for situations in which the base accuracy of the initial data is already very high to begin with. With errors so few and far at hand, the sparse data problem is exacerbated. Furthermore, an error correction algorithm should, at the very least, not create more errors than it started out with, which is a valid argument on the side of being conservative. Overall, NTPC's approach and design decisions are well-justified when the details of the task at hand are considered.

**Empirical Confirmation** The final issue behind NTPC's design is the $\epsilon(r) = 0$ condition in *Error_Corrector_Learner*. Considering that algorithms such as decision lists and transformation-based larning allow for some degree of error in their decisions, this seems like an overly extreme decision. Figures 3 and 4 show results of experiments which compare NTPC against four other systems that allow relaxed $\epsilon(r) \leq \epsilon_{\max}$ conditions for various $\epsilon_{\max} \in \{1, 2, 3, 4, \infty\}$. The system that only considers net performance improvement—i.e. $\epsilon_{\max} = \infty$, as transformation-based learning would have done—gets the worst performance in every case. Overall, the most accurate results are achieved by keeping $\epsilon(r) = 0$—which also achieves the most consistent results over time (number of rules learned). This bears out our hypothesis for keeping a zero error tolerance design.

## 5 Conclusion

We have introduced a general conservative error-correcting model, **N-fold Templated Piped Correction** (NTPC) that, unlike other existing models, can reliably deliver small but consistent gains on the accuracy of even high-performing base models on high-dimensional tasks, with little risk of accidental degradation. We have given theoretical rationales and empirical evidence to show that the various design parameters underlying NTPC are essential, including (1) easily customizable template-driven hypothesis generation, (2) sensitivity to right context, (3) n-fold piping, and (4) zero error tolerance. The resulting method is robust and should be well suited for a broad range of sequential and classification NLP tasks such as bracketing and disambiguation.

## References

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

Xavier Carreras, Lluís Màrques, and Lluís Padró. Named entity extraction using adaboost. In Dan Roth and Antal van den Bosch, editors, *Proceedings of CoNLL-2002*, pages 167–170. Taipei, 2002.

Gerard Escudero, Lluis Marquez, and German Rigau. Boosting applied to word sense disambiguation. In *European Conference on Machine Learning*, pages 129–141, 2000.

Yoram Freund and Robert E. Schapire. 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, 1997.

Ronald L. Rivest. Learning decision lists. *Machine Learning*, 2(3):229–246, 1987.

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

Erik Tjong Kim Sang and Fien Meulder. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In Walter Daelemans and Miles Osborne, editors, *Proceedings of CoNLL-2003*. Edmonton, Canada, 2003.

Erik Tjong Kim Sang. Introduction to the conll-2002 shared task: Language-independent named entity recognition. In Dan Roth and Antal van den Bosch, editors, *Proceedings of CoNLL-2002*, pages 155–158. Taipei, 2002.

Koji Tsukamoto, Yutaka Mitsuishi, and Manabu Sassano. Learning with multiple stacking for named entity recognition. In Dan Roth and Antal van den Bosch, editors, *Proceedings of CoNLL-2002*, pages 191–194. Taipei, 2002.

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

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