Features, Bagging, and System Combination for the Chinese POS Tagging Task

Fei Xia
University of Washington
Seattle, WA 98195-4340, USA
fxia@u.washington.edu

Lap Cheung
University of Washington
Seattle, WA 98195-4340, USA
lapcheung@gmail.com

Abstract

In recent years more and more NLP packages become available to the public, and many of them are implementations of general machine learning methods. A natural question is how one can quickly build a good system using those packages. To address this issue, we built three part-of-speech taggers (i.e., trigram, TBL, and MaxEnt taggers) for Chinese using existing packages. Our experiments showed that adapting and extending a package is relative easy if the package is well-written and source code is available. We studied the contribution of each type of feature templates to the tagging accuracy and showed that adding some templates could help one tagger but hurt another one. Furthermore, we demonstrated that bagging (Breiman, 1996) provides a moderate gain for the TBL tagger, and combining TBL and MaxEnt taggers work better than using all three taggers.

1 Introduction

In recent years, there has been much progress in the NLP field, and more and more NLP packages become available to the public. Many of those packages are implementations of general machine learning methods, and their usefulness has been demonstrated by some particular tasks for certain languages. Given those resources, how can we quickly build a good system for a new language? For instance, how do packages differ and what kind of packages should we choose for a particular task? If a package requires additional input such as feature templates besides the training data, what kinds of templates help and what do not? Will the same templates have the same effect on different methods? Is it easy to extend a package to accept new types of templates? If multiple packages are available, will system combination provide an additional boost?

In order to answer those questions, we did a case study: the goal was to quickly develop a Chinese Part-of-speech (POS) tagger that performs well. Our approach has three steps. First, we built three baseline POS taggers using existing packages. Two of the taggers require feature templates as input, in addition to training data. In the second step, we created several types of templates for Chinese and studied the contribution of each type to the tagging accuracy. In the third step, we applied the bagging technique (Breiman, 1996) and showed that bagging provide a moderate gain for one of the taggers, and combining two taggers work better than combining all three. In the next three sections, we shall describe each step in detail.

2 Baseline taggers

POS tagging is an important task for many NLP applications. Some common approaches are Hidden Marcov Models (HMM), transformation-based learning (TBL) (Brill, 1995; Florian and Ngai, 2001), maximum entropy models (MaxEnt) (Ratnaparkhi, 1996), boosting (Abney et al., 1999), decision tree (Màrquez, 1999), just to name a few.

For our case study, we trained three taggers:

Proceedings of the Fifth SIGHAN Workshop on Chinese Language Processing, pages 25–32,
Sydney, July 2006.

a trigram tagger using Carmel (Knight and Al-Onaizan, 1999) as a decoder, a TBL tagger using the fnTBL package (Ngai and Florian, 2001), and a MaxEnt tagger built by Le Zhang.[1] We chose these packages mainly because they all provide source codes and are well-packaged with detailed tutorials.

Both MaxEnt and TBL taggers take feature templates as input. There are a series of interesting questions with respect to feature templates. For instance, are there some types of templates that are allowed by one tagger, but not by the other? If so, are those templates useful? When we use the same templates, do they have similar effects on both taggers? If we want to add a new type of templates, is it easier for one tagger than the other? Before answering those questions, let us take a closer look at the two taggers.

2.1 POS tagging as a classification problem

In a standard classification problem, there are a finite set of input attributes and a target (a.k.a. label or class). For supervised learning, a training corpus T is a set of classified examples; that is, T is a set of (x_i, y_i) pairs, where x_i is an example, and y_i is the correct label for x_i. At the training stage, a learner induces a classifier from T, and at the test stage the classifier labels new examples.

The POS tagging task is different from the standard classification problem in that its goal is to find the best tag sequence, not the best tag for each word. The MaxEnt tagger and the TBL tagger resolve this issue in different ways, as we shall discuss in the remaining of the section.

2.2 The MaxEnt tagger

The flowchart for the training stage of the MaxEnt tagger is shown in Figure 1. The tagger requires two inputs: an original training corpus (OT), which is a set of tagged sentences, and a set of feature templates (FT). They are fed to the C&FI module, which converts OT into the attribute-value representation and instantiate features based on FT. The converted corpus (CT) and the instantiated features (F) are sent to the MaxEnt learner, which iteratively adjusts feature weights.

Table 1: Attribute-value representation for the sentence *time flies like an arrow* in MaxEnt

x_i			y_i
w_0	t_{-1}	t_{-2}, t_{-1}	Target
time	-	-,-	N
flies	N	-,N	V
like	V	N,V	P
an	P	V,P	DT
arrow	DT	P,DT	N

The conversion from OT to CT is straightforward. For instance, let us assume that the FT contains only three templates: (1) the current word w_0, (2) the tag t_{-1} of the previous word, and (3) the tags "t_{-2}, t_{-1}" of the two previous words. If the OT contains only one sentence as in (#1), the corresponding CT is shown in Table 1: each word in the sentence corresponds to a row in the table; each column except the last one corresponds to a feature template, and the last column lists the tag for the word.

```
(#1) Time/N flies/V like/P an/DT arrow/N
```

At the test stage, in order to find the best tag sequence, the MaxEnt tagger uses beam search and labels words from left to right, as described in (Ratnaparkhi, 1996). Because the decoding is done from left to right, a feature template cannot contain the tag of the current word or tags in the right context such as t_1.[2]

2.3 The TBL tagger

Like MaxEnt, the TBL tagger takes two inputs: the original training corpus (OT) and feature templates (FT), as shown in Figure 2. TBL differs from MaxEnt and many other machine learning methods in that for each (x_i, y_i) in the training data, TBL also maintains a y'_i, which is the current label of x_i. Therefore, the converted corpus (CT) is a list of (x_i, y'_i, y_i) tuples.

The training stage has three steps: (1) Each x_i is labeled y'_i by an initial tagger (e.g., a unigram tagger), and the converted corpus (CT) is formed; (2) The current label y'_i is compared with the gold standard y_i, and the TBL learner

[1] The MaxEnt package can be downloaded from http://homepages.inf.ed.ac.uk/s0450736.

[2] In this paper w_i means the i-th word from the current word, not the i-th word in the sentence. t_i is defined similarly.

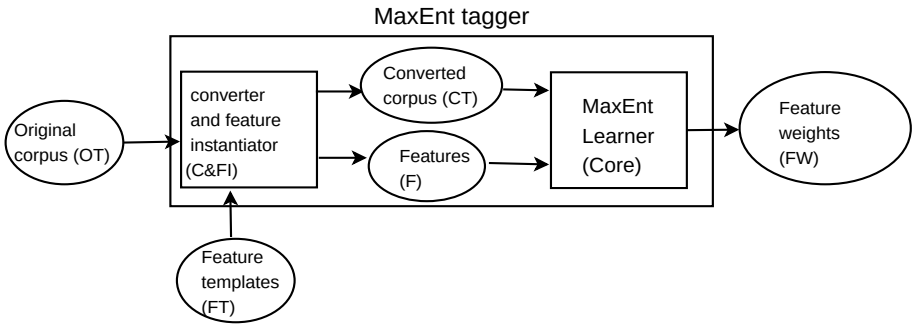

Figure 1: The training stage of the MaxEnt tagger

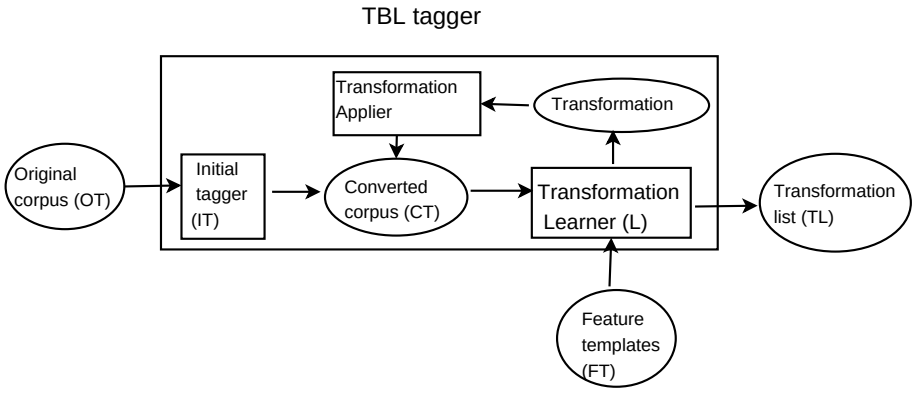

Figure 2: The training stage of the TBL tagger

selects a transformation that reduces the errors the most; (3) The CT is updated as the newly learned transformation is applied to it. Steps (2)-(3) are repeated until no more good transformations can be learned.

At the test stage, the test data are first labeled by the initial tagger, then the transformations learned at the training stage are applied to the test data one by one in the same order as they were learned.

We illustrate the main steps of TBL with an example. Suppose the TBL tagger uses the same set of feature templates and training data as in the MaxEnt tagger, and let (#2) be the result produced by the initial tagger. Table 2 shows the CT after the initial tagging. This table has one more column than Table 1, and the column, CurTag, stores the tag for the current word.[3] Also, if a feature template contains a t_i, the value of t_i comes from the CurTag column, not from the Target column.

[3]In this paper, CurTag is the same as t_0. We use CurTag in the running text, tables, and on the right-hand side of a transformation, and use t_0 when it appears in a feature template or on the left-hand side of a transformation.

Table 2: The training corpus after initial tagging in TBL

x_i			y'_i	y_i
w_0	t_{-1}	t_{-2}, t_{-1}	CurTag	Target
time	-	-,-	N	N
flies	N	-,N	**N**	V
like	**N**	N,**N**	**V**	P
an	**V**	**N,V**	DT	DT
arrow	DT	**V**,DT	N	N

Therefore, the errors in CurTag column are passed to the second and third columns, as marked in boldface.

```
(#2) Time/N flies/N like/V an/DT arrow/N
```

In the second step of the training, y_i and y'_i in the CT are compared and the best transformation is selected at each iteration. Let us assume the transformation selected at this iteration is $t_{-1}{=}N \Rightarrow \mathit{CurTag}{=}V$, which means that if the tag of the previous word is N and the CurTag for the current word is not V, then set CurTag to be V.

In the third step of training, the selected transformation is applied to the whole corpus, and the updated corpus is shown in Table 3. Notice that the CurTag for word *flies* is

Table 3: The training corpus after applying the transformation "t_{-1}=N ⇒ $CurTag$=V" (The updated parts are underlined).

x_i			y'_i	y_i
w_0	t_{-1}	t_{-2}, t_{-1}	CurTag	Target
time	-	-,-	N	N
flies	N	-,N	V	V
like	V	N,V	**V**	P
an	**V**	V,**V**	DT	DT
arrow	DT	**V**,DT	N	N

changed, and its corresponding values in the second and third columns are updated too.[4]

2.4 Differences between MaxEnt and TBL taggers

TBL and MaxEnt are very different learning methods: TBL is rule-based, whereas MaxEnt is statistical. Because MaxEnt is statistical, MaxEnt taggers can treat the tagging of each word as a classification problem, produce the top-N best tags for each word, and then use beam search to find the best tag sequence. Such an option is not available to the TBL method, because TBL does not provide scores or probabilities for its decisions.

In addition, the two methods have several major differences that are relevant to our present work:

- The CT in MaxEnt is a list of (x_i, y_i) pairs and is unchanged once created, whereas the CT in TBL is a list of (x_i, y'_i, y_i) tuples and is updated at each iteration.
- The feature set F in MaxEnt is unchanged once created; whereas the TBL learner may generate new features at each iteration because the CT keeps changing.
- It is easy to extend the MaxEnt tagger to accept new *types* of feature templates: we only need to change the C&FI module. Doing the same for TBL is harder as we need to change the transformation learner, which is more complicated.[5]
- The MaxEnt tagger labels a sentence from left to right, which implies that features used in MaxEnt cannot refer to tags in the right context. In contrast, TBL does not label words from left to right, and the CurTags for the words in the right context are always available. Therefore, features in TBL can refer to tags of any words in the sentence.
- Feature templates are used differently in the two taggers, as we shall explain in Section 3.

To summarize, adding new feature template types to the MaxEnt tagger is easier than adding them to the TBL tagger because both CT and instantiated features are unchanged during the iteration in the MaxEnt tagger. However, the MaxEnt tagger cannot use features that refer to the tag of the current word and the tags in the right context, while the TBL tagger can. But are those features useful? We shall answer that question in Section 5.

3 Feature templates

Feature templates can be divided into two types: contextual templates that look at the context (e.g., neighboring words), and lexical templates that look at word spelling and the like.

MaxEnt and TBL taggers use feature templates in different ways. For instance, given a template "t_{-1}" and the training corpus in Table 1, the MaxEnt tagger will create the following feature for the word *flies*:

$$f_j(h,t) = \begin{cases} 1 & \text{if } t_{-1} = N \text{ and } t = V \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

Each feature f_j has a weight λ_j, and is used to calculate the probability of a history h and the tag t as defined below:

$$p(h,t) = \frac{e^{\sum_{j=1}^{k} \lambda_j f_j(h,t)}}{Z} \quad (2)$$

In contrast, for the same feature template t_{-1} and for the corpus in Table 2, the TBL tagger will create a transformation for the word *flies*:

$$t_{-1} = N \Rightarrow CurTag = V \quad (3)$$

[4] The fnTBL package stores the training corpus in a slightly different representation, but our statement about TBL still holds.

[5] In the fnTBL package, a template is a conjunction of so-called *atomic* templates, and each atomic template is defined as a C++ class. If a new template includes new types of atomic templates, we have to define additional C++ classes for the new atomic templates.

This transformation is applicable if the CurTag is **not** V and the previous tag is N; when the transformation applies, it will set the CurTag to be V.

If we treat a transformation as a rule, CurTag can appear in both sides of the rule. For instance for the template "t_0 t_{-1}" and the word *flies* in Table 2, the TBL tagger will create a transformation

$$t_0 = N \text{ and } t_{-1} = N \Rightarrow CurTag = V \quad (4)$$

Notice that the transformations in (3) and (4) are different, as (4) will not fire unless the CurTag is N. In Section 5, we shall show that whether or not CurTag is included in the templates could affect tagging accuracy significantly.

3.1 Contextual templates

To study the contribution of each type of templates, we define three types of contextual templates:[6]

C1: seven templates that are used in both the TBL tagger and the original MaxEnt tagger

C2: seventeen less commonly used templates

C3: thirteen templates that include tags from the right context

To test the effect of including CurTag, we also define a set C_i' for each C_i: templates in C_i' are the same as the ones in C_i except that they include CurTag. For instance, "t_{-1}" in C_1 becomes "t_0, t_{-1}" in C_1'.

3.2 Lexical templates

In addition to contextual templates, both taggers use lexical templates to handle unknown words. In our experiments, we use three types of lexical templates that are defined in the fnTBL package:

L1: Affix: An Affix template checks whether a word starts (or ends) with a character n-gram.

L2: WordInVoc: A WordInVoc template checks whether removing or adding a character n-gram will result in a word that has appeared in the training data.

L3: SubString: A SubString template checks whether a word contains a particular substring.

The original MaxEnt package uses features in C1 and L1. We modified the code so that it accepts features in C2, L2, and L3 as well.

4 Bagging

Bagging (Bootstrap Aggregating) is a method for generating multiple versions of a predictor and using them to get an aggregated predictor (Breiman, 1996).[7] The bagging algorithm is very simple: during the training stage, multiple bootstrap replicates[8] are generated from the training data and each replicate induces a predictor. During the decoding stage, each test example is processed by all the predictors and the final result is created in a voting approach.

Breiman (1996) shows that bagging achieves impressive results when applied to unstable learning algorithms (e.g., decision tree), but it could degrade the performance of stable algorithms (e.g., kNN). Henderson and Brill (2000) used bagging and boosting to improve a Treebank parser. For the POS tagging task, Màrquez et al. (1999) shows that bagging decision-tree taggers improves tagging accuracy on English text, and similar improvement is observed for Hungarian text when bagging TBL taggers (Kuba et al., 2005).

In our present work, we test the effect of bagging on our three baseline taggers for Chinese text.

5 Experiments

We ran our experiment on the Chinese Penn Treebank (CTB) version 5.0, which contains 500 thousand words of newspaper and magazine articles from three sources: Xinhua News Agency from Mainland China, HKSAR from

[6] We started with the templates used in the fnTBL package and then made some modifications. The complete contextual template list is given in the Appendix.

[7] A predictor can be any function that maps input to output, and in this case, a predictor is simply a POS tagger.

[8] Given a training set of n examples, a bootstrap replicate is created by randomly drawing with replacement n times.

Hong Kong, and Sinorama Magazine from Taiwan. Three previous work (Xue et al., 2002; Florian and Ngai, 2001; Ng and Low, 2004) were trained on various earlier versions of the treebank, which contain 100K, 160K, and 250K words respectively. (Tseng et al., 2005) is the only previous work that was trained on version 5.0; therefore we used the same data split as they did.[9] Table 4 shows the sources and sizes of the data sets. The average sentence length is 27 words, and the OOV rate (the percentage of unknown tokens) is 6.42% on the DevSet and 5.86% on the TestSet. The unigram tagging accuracy on the DevSet is 86.31%.

Table 4: Training, development, and test data

Data Set	Sections	words
Training		461406
Xinhua	026-270, 301-325, 400-454, 600-931	213910
HKSAR	none	0
Sinorama	1003-1039, 1043-1151	247496
DevSet		23839
Xinhua	001-025	7844
HKSAR	500-527	8202
Sinorama	590-593, 1001-1002	7793
TestSet		23522
Xinhua	271-300	8008
HKSAR	528-554	7153
Sinorama	594-596, 1040-1042	8361

5.1 TBL results

To test the effects of contextual templates on TBL, we first use an empty lexical template set, which means all the unknown words are assigned a default tag at both the training and the test stages. The results are in Table 5, where the three columns list the tagging accuracy for all words, known words, and unknown words, respectively. As shown in the first two rows, C1' works much better than C1, indicating that it is useful to include CurTag in the feature templates for this template set.

Then we add either $C2$ or $C2'$ to $C1'$. In both cases, the tagging accuracy improves, and $C2$ works slightly better than $C2'$. Next, we add C3 or C3', and the results are further improved.

To test the effect of lexical templates, we start with the best result from Table 5, which uses C1'+C2+C3'. The results are in Table 6. The lexical templates greatly improve the accuracy of unknown words, and the overall accuracy increases from 92.13% to 92.83%.

[9]Our training data plus the DevSet is the same as the Training III defined in their paper.

Table 5: Effects of contextual templates for TBL (on the DevSet)

Features	Overall	Known	Unk
C1	88.82	91.24	53.52
C1'	91.35	93.46	60.76
C1'+C2	91.70	93.83	60.69
C1'+C2'	91.60	93.80	59.71
C1'+C2+C3	92.05	94.22	60.52
C1'+C2+C3'	**92.13**	94.26	61.15

Table 6: Effects of lexical templates for TBL (on the DevSet)

Features	Overall	Known	Unk
C1'+C2+C3'	92.13	94.26	61.15
C1'+C2+C3'+L1	92.74	94.17	71.90
C1'+C2+C3'+L1+L2	92.46	93.89	71.66
C1'+C2+C3'+L1+L2+L3	**92.83**	94.29	71.56

5.2 MaxEnt results

The MaxEnt tagger cannot use C1', C2', C3' (which refer to the tag of the current word) and C3 (which refers to tags of words to the right). We ran the tagger with other template sets, and the results are in Table 7. When we compare this with TBL results, we observe that some templates can have opposite effects on the two taggers. For instance, while adding C2 helps the TBL tagger, it hurts the MaxEnt tagger.

Table 7: Tagging accuracy of the MaxEnt tagger (on the DevSet)

Features	Overall	Known	Unk
C1	92.75	94.46	67.99
C1+C2	92.56	94.28	67.54
C1+L1	**93.65**	94.83	76.53
C1+L1+L2	93.62	94.79	76.60
C1+L1+L2+L3	93.42	94.76	74.05

5.3 Bagging results

Table 8 shows the bagging results. The first row is the baseline result without bagging. The rest are the results when the number of bags ranges from 1 to 100.

A few observations are in order: First, among three taggers, bagging helps TBL the most (from 92.83% to 93.25%) with a relative error reduction of 5.85%, but the gain is very

Table 8: Bagging with different numbers of bags (on the DevSet)

	Trigram	TBL	MaxEnt	Trigram+TBL+MaxEnt	TBL+MaxEnt
no bagging	90.41	92.83	**93.65**	93.25	N/A
1 bag	89.71	91.71	92.73	92.87	N/A
3 bags	90.06	92.70	93.15	93.32	93.55
10 bags	90.30	93.08	93.45	93.46	93.85
25 bags	90.43	93.13	93.58	93.47	93.91
50 bags	90.50	93.20	93.60	93.48	93.93
75 bags	90.48	93.23	93.61	93.49	93.93
100 bags	90.51	93.25	93.60	93.50	**93.95**

moderate.[10] Second, the last two columns indicate that leaving out the trigram tagger from voting yields better results. Overall, bagging and system combination together improves the tagging accuracy from 93.65% (the best single system) to 93.95% (TBL+MaxEnt with 100 bags), achieving a relative error rate deduction of 4.72%.

5.4 Results on the test data

Table 9 shows the results on the test data. It follows the same pattern as Table 8, and the overall improvement is from 93.14% to 93.58%.

Table 9: Tagging accuracy on the TestSet

	Trigram	TBL	MaxEnt	TBL + MaxEnt
no bagging	90.32	92.41	**93.14**	N/A
1 bags	89.63	91.19	92.31	N/A
3 bags	89.95	92.13	92.73	93.17
10 bags	90.16	92.49	93.00	93.42
25 bags	90.32	92.65	93.08	93.52
100 bags	90.33	92.69	93.10	**93.58**

For comparison, among all the previous work on the Chinese POS tagging, (Tseng et al., 2005) was the only one that was trained and tested on the Chinese Penn Treebank version 5.0. However, in their experiments, the whole corpus was cleaned up before training, which yielded a 0.46% absolute performance gain when their tagger was trained on a subset of the whole training data. Because of the clean-up step and the fact that they used a different MaxEnt package, their MaxEnt baseline result on the test set was 93.51% whereas ours was 93.14%. By adding seven new types of templates, their tagging accuracy improves from 93.51% to 93.74%, whereas our tagging accuracy improves from 93.14% to 93.58% when MaxEnt and TBL output were combined. The two sets of experiments show that both adding features and system combination could improve tagging accuracy.

[10] In comparison, Kuba et al. (2005) reports a relative error reduction of 18% (tagging accuracy changes from 98.26% to approximately 98.58%) on Hungarian text. Because the two experiments used texts from different languages and the baseline results (92.83% vs. 98.26%) are quite different, comparing the error reduction rates is not very meaningful. Nevertheless, we plan to look into this difference in the future.

6 Conclusion

In this paper, we investigated the possibility of quickly adapting existing tools for a new language. We learned a few important lessons.

First, one should choose good NLP packages as the starting point. For our experiments, it is crucial that the packages include source code, which allows us to modify or extend the packages. For instance, we modified the MaxEnt tool so that it can accept feature templates in (C2), (L2), and (L3).

Second, both MaxEnt and TBL taggers use feature templates, but the same templates (e.g., C2, L2, and L3) could have opposite effects on the two taggers. In addition, TBL taggers can use feature templates that refer to the tags of the current word and the words to the right. Including CurTag in C1 greatly improves the tagging accuracy (from 88.82% to 91.35%), but adding it in C2 hurts the accuracy slightly (from 91.70% to 91.60%). Adding templates that refer to tags of the words to the right helps a little bit (from 91.70% to 92.13%).

Third, among the three taggers bagging helps TBL the most, and it has little effect on the trigram and the MaxEnt taggers. Combining TBL and MaxEnt provides a moderate gain, and it is better to leave out the Trigram tagger from the system combination.

For future work, we plan to experiment with other learning algorithms such as SVM, and other methods of system combination.

A Contextual templates used in the TBL and MaxEnt taggers

We follow the template format used in the fnTBL package. All the numbers in the templates are relative positions with respect to the current word. For instance, pos_-1 is the tag of the previous word, and word_1 is the next word. [i,j] is a range of the context. For instance, word:[-3,-1] means one of the previous three words. Notice that C3 is used only in TBL.

```
C1: seven basic templates that do not use
    tags in the right context.

word_0
word_-1
word_1
word_-2
word_2
pos_-1
pos_-2 pos_-1

C2: seventeen more templates that do not
    use tags in the right context.

word_0 word_1 word_2
word_-1 word_0 word_1
word_0 word_-1
word_0 word_1
word_0 word_2
word_0 word_-2
word:[1,2]
word:[-2,-1]
word:[1,3]
word:[-3,-1]
word_0 pos_-2
word_0 pos_-1
pos_-2
pos:[-3,-1]
pos:[-2,-1]
pos_-1 word_-1 word_0
pos_-1 word_0 word_1

C3: thirteen templates that use the tags in
    the right context.

word_0 pos_1
word_0 pos_2
pos_-1 pos_1
pos_1 pos_2
pos_1
pos_2
pos_1 word_0 word_1
pos_1 word_0 word_-1
pos:[1,3]
pos:[1,2]
pos_1 pos_2 word_1
pos_1 word_0 word_1
pos_1 word_0 word_-1
```

References

Steven Abney, Robert E. Schapire, and Yoram Singer. 1999. Boosting Applied to Tagging and PP Attachment. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP-1999)*, pages 38–45.

Leo Breiman. 1996. Bagging predictors. *Machine Learning*, 24(2):123–140.

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.

Radu Florian and Grace Ngai. 2001. Multidimensional transformation-based learning. In *Proceedings of the 5th Conference on Computational Natural Language Learning (CoNLL-2001)*.

John Henderson and Eric Brill. 2000. Bagging and boosting a treebank parser. In *Proceedings of the 6th Applied Natural Language Processing Conference (ANLP-2000)*.

Kevin Knight and Yaser Al-Onaizan. 1999. A primer on finite-state software for natural language processing. downloadable from http://www.isi.edu/licensed-sw/carmel/carmel-tutorial2.pdf.

András Kuba, László Felföldi, and András Kocsor. 2005. Pos tagger combinations on hungarian text. In *2nd International Joint Conference on Natural Language Processing (IJCNLP-2005)*.

Lluís Màrquez, Horacio Rodríguez, Josep Carmona, and Josep Montolio. 1999. Improving pos tagging using machine-learning techniques. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP-1999)*, pages 53–62.

Lluís Màrquez. 1999. *Part-of-speech Tagging: A Machine Learning Approach based on Decision Trees*. Ph.D. thesis, Universitat Politècnica de Catalunya.

Hweee Tou Ng and Jin Kiat Low. 2004. Chinese Part-of-speech Tagging: One-at-a-Time or All-at-Once? Word-based or Character-based? In *Proc. of the 9th Conf. on Empirical Methods in Natural Language Processing (EMNLP-2004)*.

Grace Ngai and Radu Florian. 2001. Transformation-based learning in the fast lane. In *Proceedings of North American ACL (NAACL-2001)*, pages 40–47, June.

Adwait Ratnaparkhi. 1996. A Maximum Entropy Model for Part-of-speech Tagging. In *Proc. of Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP-1996)*.

Huihsin Tseng, Daniel Jurafsky, and Christopher Manning. 2005. Morphological features help pos tagging of unknown words across language varieties. In *Proc. of the 4th Workshop on Chinese Language Processing (SIGHAN-2005)*.

Nianwen Xue, Fu dong Chiou, and Martha Palmer. 2002. Building a Large-scale Annotated Chinese Corpus. In *Proc. of the 19th International Conference on Computational Linguistics (COLING-2002)*.