An Improved Tag Dictionary for Faster Part-of-Speech Tagging

Robert C. Moore
Google Inc.
bobmoore@google.com

Abstract

Ratnaparkhi (1996) introduced a method of inferring a tag dictionary from annotated data to speed up part-of-speech tagging by limiting the set of possible tags for each word. While Ratnaparkhi's tag dictionary makes tagging faster but less accurate, an alternative tag dictionary that we recently proposed (Moore, 2014) makes tagging as fast as with Ratnaparkhi's tag dictionary, but with no decrease in accuracy. In this paper, we show that a very simple semi-supervised variant of Ratnaparkhi's method results in a much tighter tag dictionary than either Ratnaparkhi's or our previous method, with accuracy as high as with our previous tag dictionary but much faster tagging—more than 100,000 tokens per second in Perl.

1 Overview

In this paper, we present a new method of constructing tag dictionaries for part-of-speech (POS) tagging. A tag dictionary is simply a list of words[1] along with a set of possible tags for each word listed, plus one additional set of possible tags for all words not listed. Tag dictionaries are commonly used to speed up POS-tag inference by restricting the tags considered for a particular word to those specified by the dictionary.

Early work on POS tagging generally relied heavily on manually constructed tag dictionaries, sometimes agumented with tag statistics derived from an annotated corpus (Leech et al., 1983; Church, 1988; Cutting et al., 1992). Merialdo (1994) relied only on a tag dictionary extracted from annotated data, but he used the annotated tags from his test data as well as his training data to construct his tag dictionary, so his evaluation was not really fair.[2] Ratnaparkhi (1996) seems to have been the first to use a tag dictionary automatically extracted only from training data.

Ratnaparkhi's method of constructing a tag dictionary substantially speeds up tagging compared to considering every possible tag for every word, but it noticeably degrades accuracy when used with a current state-of-the-art tagging model. We recently presented (Moore, 2014) a new method of constructing a tag dictionary that produces a tagging speed-up comparable to Ratnaparkhi's, but with no decrease in tagging accuracy. In this paper, we show that a very simple semi-supervised variant of Ratnaparkhi's method results in a much tighter tag dictionary than either Ratnaparkhi's or our previous method, with accuracy as high as we previously obtained, while allowing much faster tagging—more than 100,000 tokens per second even in a Perl implementation.

1.1 Tag Dictionaries and Tagging Speed

A typical modern POS tagger applies a statistical model to compute a score for a sequence of tags $t_1, \ldots, t_n$ given a sequence of words $w_1, \ldots, w_n$. The tag sequence assigned the highest score by the model for a given word sequence is selected as the tagging for the word sequence. If $\mathcal{T}$ is the set of possible tags, and there are no restrictions on the form of the model, then the time to find the highest scoring tag sequence is potentially $O(n|\mathcal{T}|^n)$ or worse, which would be intractable.

To make tagging practical, models are normally defined to be factorable in a way that reduces the time complexity to $O(n|\mathcal{T}|^k)$, for some small integer k. For models in which all tagging decisions are independent, or for higher-order mod-

[1] According to the conventions of the field, POS tags are assigned to all tokens in a tokenized text, including punctuation marks and other non-word tokens. In this paper, all of these will be covered by the term *word*.

[2] Merialdo (1994, p. 161) acknowledged this: "In some sense this is an optimal dictionary for this data, since a word will not have all its possible tags (in the language), but only the tags it actually had within the text."

Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, pages 1303–1308,
Lisbon, Portugal, 17-21 September 2015.

els pruned by fixed-width beam search, $k = 1$, so the time to find the highest scoring tag sequence is $O(n|\mathcal{T}|)$. But this linear dependence on the size of the tag set means that reducing the average number of tags considered per token should further speed up tagging, whatever the underlying model or tagger may be.

1.2 Ratnaparkhi's Method

For each word observed in an annotated training set, Ratnaparkhi's tag dictionary includes all tags observed with that word in the training set, with all possible tags allowed for all other words. Ratnaparkhi reported that using this tag dictionary improved per-tag accuracy from 96.31% to 96.43% on his Penn Treebank (Marcus et al., 1993) Wall Street Journal (WSJ) development set, compared to considering all tags for all words.

With a more accurate model, however, we found (Moore, 2014) that while Ratnaparkhi's tag dictionary decreased the average number of tags per token from 45 to 3.7 on the current standard WSJ development set, it also decreased per-tag accuracy from 97.31% to 97.19%. This loss of accuracy can be explained by the fact that 0.5% of the development set tokens are known words with a tag not seen in the training set, for which our model achieved 44.5% accuracy with all word/tag pairs permitted. With Ratnaparkhi's dictionary, accuracy for these tokens is necessarily 0%.

1.3 Our Previous Method

We previously presented (Moore, 2014) a tag dictionary constructed by using the annotated training set to compute a smoothed probability estimate for any possible tag given any possible word, and for each word in the training set, including in the dictionary the tags having an estimated probability greater than a fixed threshold T. In this approach, the probability $p(t|w)$ of tag t given word w is computed by interpolating a discounted relative frequency estimate of $p(t|w)$ with an estimate of $p(t)$ based on "diversity counts", taking the count of a tag t to be the number of distinct words ever observed with that tag. The distribution $p(t)$ is also used to estimate tag probabilities for unknown words, so the set of possible tags for any word not explicitly listed is $\{t|p(t) > T\}$.

If we think of w followed by t as a word bigram, this is exactly like a bigram language model estimated by the interpolated Kneser-Ney (KN) method described by Chen and Goodman (1999). The way tag diversity counts are used has the desirable property that closed-class tags receive a very low estimated probability of being assigned to a rare or unknown word, even though they occur very often with a small number of frequent words. A single value for discounting the count of all observed word/tag pairs is set to maximize the estimated probability of the reference tagging of the development set. When T was chosen to be the highest threshold that preserves our model's 97.31% per tag WSJ development set accuracy, we obtained an average of 3.5 tags per token.

1.4 Our New Approach

We now present a new method that reduces the average number of tags per token to about 1.5, with no loss of tagging accuracy. We apply a simple variant of Ratnaparkhi's method, with a training set more than 4,000 times larger than the Penn Treebank WSJ training set. Since no such hand-annotated corpus exists, we create the training set automatically by running a version of our tagger on the LDC English Gigaword corpus. We thus describe our approach as a semi-supervised variant of Ratnaparkhi's method. Our method can be viewed as an instance of the well-known technique of self-training (e.g., McClosky et al., 2006), but ours is the first use of self-training we know of for learning inference-time search-space pruning.

We introduce two additional modifications of Ratnaparki's approach. First, with such a large training corpus, we find it unnecessary to keep in the dictionary every tag observed with every word in the automatically-annotated data. So, we estimate a probability distribution over tags for each word in the dictionary according to unsmoothed relative tag frequencies, and include for each word in the dictionary only tags whose probability given the word is greater than a fixed threshold.

Second, since our tokenized version of the English Gigaword corpus contains more than 6 million unique words, we reduce the vocabulary of the dictionary to the approximately 1 million words having 10 or more occurrences in the corpus. We treat all other tokens as instances of unknown words, and we use their combined unsmoothed relative tag frequencies to estimate a tag probability distribution for unknown words. We use the same threshold on this distribution as we do for words explicitly listed in the dictionary, to obtain a set of possible tags for unknown words.

2 Experimental Details

In our experiments, we use the WSJ corpus from Penn Treebank-3, split into the standard training (sections 0–18), development (sections 19–21), and test (sections 22-24) sets for POS tagging.

The tagging model we use has the property that all digits are treated as indistinguishable for all features. We therefore also make all digits indistinguishable in constructing tag dictionaries (by internally replacing all digits by "9"), since it does not seem sensible to give two different dictionary entries based on containing different digits, when the tagging model assigns them the same features.

2.1 The Tagging Model

The model structure, feature set, and learning method we use for POS tagging are essentially the same as those in our earlier work, treating POS tagging as a single-token independent multiclass classification task. Word-class-sequence features obtained by supervised clustering of the annotated training set replace the hidden tag-sequence features frequently used for POS tagging, and additional word-class features obtained by unsupervised clustering of a very large unannotated corpus provide information about words not occurring in the training set. For full details of the feature set, see our previous paper (Moore, 2014).

The model is trained by optimizing the multiclass SVM hinge loss objective (Crammer and Singer, 2001), using stochastic subgradient descent as described by Zhang (2004), with early stopping and averaging. The only difference from our previous training procedure is that we now use a tag dictionary to speed up training, while we previously used tag dictionaries only at test time.

Our training procedure makes multiple passes through the training data considering each training example in turn, comparing the current model score of the correct tag for the example to that of the highest scoring incorrect tag and updating the model if the score of the correct tag does not exceed the score of the highest scoring incorrect tag by a specified margin. In our new version of this procedure, we use the KN-smoothed tag dictionary described in Section 1.3. to speed up finding the highest scoring incorrect tag.

Recall that the KN-smoothed tag dictionary estimates a non-zero probability $p(t|w)$ for every possible word/tag pair, and that the possible tags for a given word are determinted by setting a threshold T on this probability. In each pass through the training set, we use the same probability distribution $p(t|w)$ determined from the statistics of the annotated training data, but we employ an adaptive method to determine what threshold T to use in each pass.

For the first pass through the training set, we set an initial threshold T_0 to the highest value such that for every token in the development set, $p(t|w) \geq T_0$, where t is the correct tag for the token and w is the word for the token. At the end of each training pass i, while evaluating the current model on the development set for early stopping using threshold T_{i-1}, we also find the highest probability threshold T_i such that choosing a lower threshold would not enable any additional correct taggings on the development set using the current model. This threshold will normally be higher than T_0, because we disregard tokens in the development set for which the correct tag would not be selected by the model resulting from the previous pass at any threshold. T_i is then used as the threshold for training pass $i + 1$. Whenever the selected threshold leaves only one tag remaining for a particular training example, we skip that example in training.

On the first pass through the training set, use of this method resulted in consideration of an average of 31.36 tags per token, compared to 45 total possible tags. On the second and all subsequent passes, an average of 10.48 tags were considered per token. This sped up training by a factor of 3.7 compared to considering all tags for all tokens, with no loss of tagging accuracy when a development-set-optimized KN-smoothed tag dictionary is also used at test time.

2.2 Tagging the Gigaword Corpus

To construct our new tag dictionary, we need an automatically-tagged corpus several orders of magnitude larger than the hand-tagged WSJ training set. To obtain this corpus we ran a POS tagger on the LDC English Gigaword Fifth Edition[3] corpus, which consists of more than 4 billion words of English text from seven newswire sources. We first removed all SGML mark-up, and performed sentence-breaking and tokenization using the Stanford CoreNLP toolkit (Manning et al, 2014). This produced 4,471,025,373 tokens of

[3] https://catalog.ldc.upenn.edu/LDC2011T07

Tag Dictionary	Accuracy	Tags/Token	Unambig	Tokens/Sec
Pruned KN-smoothed	97.31%	3.48	45.3%	69k
Unpruned semi-supervised	97.31%	1.97	51.7%	82k
Pruned semi-supervised	97.31%	1.51	66.8%	103k

Table 1: WSJ development set token accuracy and tagging speed for different tag dictionaries

6,616,812 unique words. We tagged this corpus using the model described in Section 2.1 and a KN-smoothed tag dictionary as described in Section 1.3, with a threshold $T = 0.0005$. The tagger we used is based on the fastest of the methods described in our previous work (Moore, 2014, Section 3.1). Tagging took about 26 hours using a single-threaded implementation in Perl on a Linux workstation equipped with Intel Xeon X5550 2.67 GHz processors.

2.3 Extracting the Tag Dictionary

We extracted a Ratnaparkhi-like tag dictionary for the 957,819 words with 10 or more occurrences in our corpus. Tokens of all other words in the corpus were treated as unknown word tokens and used to define a set of 24 tags[4] to be used for words not explicitly listed in the dictionary. To allow pruning the dictionary as described in Section 1.4, for each word (including the unknown word), we computed a probability distribution $p(t|w)$ using unsmoothed relative frequencies. As noted above, we treated all digits as indistinguishable in constructing and applying the dictionary.

3 Experimental Results

Tagging the WSJ development set with an unpruned semi-supervised tag dictionary obtained from the automatic tagging of the English Gigaword corpus produced the same tagging accuracy as allowing all tags for all tokens or using the pruned KN-smoothed tag dictionary used in tagging the Gigaword corpus. Additional experiments showed that we could prune this dictionary with a threshold on $p(t|w)$ as high as $T = 0.0024$ without decreasing development set accuracy. In addition to applying this threshold to the tag probabilities for all listed words, we also applied it to the tag probabilities for unknown words, leaving 13 possible tags[5] for those.

Tagging the WSJ development set with these two dictionaries is compared in Table 1 to tagging with our previous pruned KN-smoothed dictionary. The second column shows the accuracy per tag, which is 97.31% for all three dictionaries. The third column shows the mean number of tags per token allowed by each dictionary. The fourth column shows the percentage of tokens with only one tag allowed, which is significant since the tagger need not apply the model for such tokens—it can simply output the single possible tag.

The last column shows the tagging speed in tokens per second for each of the three tag dictionaries, using the fast tagging method we previously described (Moore, 2014), in a single-threaded implementation in Perl on a Linux workstation equipped with Intel Xeon X5550 2.67 GHz processors. Speed is rounded to the nearest 1,000 tokens per second, because we measured times to a precision of only about one part in one hundred. For the pruned KN-smoothed dictionary, we previously reported a speed of 49,000 tokens per second under similar conditions. Our current faster speed of 69,000 tokens per second is due to an improved low-level implementation for computing the model scores for permitted tags, and a slightly faster version of Perl (v5.18.2).

The most restrictive tag dictionary, the pruned semi-supervised dictionary, allows only 1.51 tags per token, and our implementation runs at 103,000 tokens per second on the WSJ development set. For our final experiments, we tested our tagger with this dictionary on the standard Penn Treebank WSJ test set and on the Penn Treebank-3 parsed Brown corpus subset, as an out-of-domain evaluation. For comparison, we tested our previous tagger and the fast version (english-left3words-distsim) of the Stanford tagger (Toutanova et al., 2003; Manning, 2011) recommended for practical use on the Stanford tagger website, which we found to be by far the fastest of the six publicly available taggers tested in our previous work (Moore, 2014). The results of these tests are shown in Table 2.[6]

[4] CC, CD, DT, FW, IN, JJ, JJR, JJS, MD, NN, NNP, NNPS, NNS, PRP, RB, RBR, RP, UH, VB, VBD, VBG, VBN, VBP, and VBZ

[5] CD, FW, JJ, NN, NNP, NNPS, NNS, RB, VB, VBD, VBG, VBN, and VBZ

[6] In Table 2, "OOV" has the standard meaning of a token

Tagger	WSJ Tokens/Sec	All WSJ Accuracy	OOV WSJ Accuracy	Brown Tokens/Sec	All Brown Accuracy	OOV Brown Accuracy
This work	102k	97.36%	91.09%	96k	96.55%	89.25%
Our previous	51k/54k/69k	97.34%	90.98%	40k/43k/56k	96.54%	89.36%
Stanford fast	80k	96.87%	89.69%	50k	95.53%	87.38%

Table 2: WSJ test set and Brown corpus tagging speeds and token accuracies

For our previous tagger, we give three speeds: the speed we reported earlier, a speed for a duplicate of the earlier experiment using the faster version of Perl that we use here, and a third measurement including both the faster version of Perl and our improved low-level tagger implementation.

With the pruned semi-supervised dictionary, our new tagger has slightly higher all-token accuracy than our previous tagger on both the WSJ test set and Brown corpus set, and it is much more accurate than the fast Stanford tagger. The accuracy on the standard WSJ test set is 97.36%, one of the highest ever reported. The new tagger is also much faster than either of the other taggers, achieving a speed of more than 100,000 tokens per second on the WSJ test set, and almost 100,000 tokens per second on the out-of-domain Brown corpus data.

4 Conclusions

Our method of constructing a tag dictionary is technically very simple, but remarkably effective. It reduces the mean number of possible tags per token by 57% and increases the number of unambiguous tokens by by 47%, compared to the previous state of the art (Moore, 2014) for a tag dictionary that does not degrade tagging accuracy. When combined with our previous work on fast high-accuracy POS tagging, this tag dictionary produces by far the fastest POS tagger reported with anything close to comparable accuracy.

References

Stanley F. Chen and Joshua T. Goodman. 1999. An empirical study of smoothing techniques for language modeling. *Computer Speech and Language*, 13(4):359–393.

Kenneth Ward Church. 1988. A stochastic parts program and noun phrase parser for unrestricted text. In *Proceedings of the Second Conference on Applied Natural Language Processing of the Association for Computational Linguistics*, February 9–12, Austin, Texas, USA, 136–143.

Koby Crammer and Yoram Singer. 2001. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292.

Doug Cutting, Julian Kupiec, Jan Pedersen, and Penelope Siburn. 1992. A practical part-of-speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing of the Association for Computational Linguistics*, March 31–April 3, Trento, Italy, 133–140.

Geoffrey Leech, Roger Garside, and Eric Atwell. 1983. The automatic tagging of the LOB corpus. *ICAME Journal: International Computer Archive of Modern and Medieval English Journal*, 7:13–33.

Christopher D. Manning. 2011. Part-of-speech tagging from 97% to 100%: Is it time for some linguistics? In Alexander Gelbukh (ed.), *Computational Linguistics and Intelligent Text Processing, 12th International Conference, CICLing 2011, Proceedings, Part I. Lecture Notes in Computer Science 6608*, Springer, 171–189.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, June 23–24, Baltimore, Maryland, USA, 55–60.

Mitchell P. Marcus, Beatrice Santorini, and Mary A. Marcinkiewicz. 1993. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330.

David McClosky, Eugene Charniak, and Mark Johnson. 2006. Effective Self-Training for

of a word not occurring in the annotated training set. Many of these tokens do match words in our large semi-supervised tag dictionary, however.

Parsing. In *Proceedings of the 2006 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, June 4–9, New York, New York, USA, 152-159.

Bernard Merialdo. 1994. Tagging English text with a probabalistic model. *Computational Linguistics*, 20(2):155–171.

Robert C. Moore. 2014. Fast high-accuracy part-of-speech tagging by independent classifiers. In *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, August 23–29, Dublin, Ireland, 1165–1176.

Adwait Ratnaparkhi. 1996. A maximum entropy model for part-of-speech tagging. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, May 17–18, Philadelphia, Pennsylvania, USA, 133–142.

Kristina Toutanova, Dan Klein, Christopher D. Manning, and Yoram Singer. 2003. Feature-rich part-of-speech tagging with a cyclic dependency network. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*, May 27–June 1, Edmonton, Alberta, Canada, 173–180.

Tong Zhang. 2004. Solving large scale linear prediction problems using stochastic gradient descent algorithms. In *Proceedings of the 21st International Conference on Machine Learning*, July 4–8, Banff, Alberta, Canada, 919–926.