Ensemble-based POS tagging of Italian

Anders Søgaard

Center for Language Technology
University of Copenhagen

Abstract. Simple learning algorithms are used to learn what labels to assign words based on the predictions provided by two non-optimized part-of-speech taggers, a tagger with a different tag set and, possibly, an earlier run of the classifier. Our accuracy on the tagged La Repubblica corpus of Italian newspaper articles is 3.6% higher than that of our best input tagger in general, and ∼16.4% better in tagging unknown words. Results are generally non-competitive because of non-optimized input taggers. In Sect. 3, we show (i) how results can be improved with the current input taggers (by ∼0.7% and ∼2.3, resp.), and (ii) how more competitive results can be obtained with more qualified input taggers.

Keywords: POS tagging, ensemble-based learning, unsupervised clusters, semi-supervised learning.

1 Introduction

A simple technique is presented for improving the results of part-of-speech (POS) taggers. The point of this work, however, is not as much to provide a competitive system as it is an attempt to provide a *proof-of-concept* for a novel meta-learning strategy.

Several meta-learning strategies have been applied to POS tagging in the literature, incl. bagging, boosting, random forests and majority voting of otherwise unrelated input taggers [1–3]. Our work is related to boosting and majority voting of input taggers in several ways and can to some extent be seen as a combination of these two techniques.

Boosting was introduced by Rob Schapire and Yoav Freund in 1990, e.g. [4], and is, put briefly, a set of techniques for iteratively learning weak classifiers that can be combined into a stronger *end classifier*. Each addition of a weak classifier increases the weights of misclassified examples and decreases the weights of correctly classified examples. Similarities between boosting and transformation-based learning, e.g. as in the Brill tagger [5], have been pointed out in [2]. The weak classifiers in boosting are often combined into a strong classifier in a majority voting scheme. In majority voting, the input classifiers may also, however, be arbitrarily related, e.g. off-the-shelf POS taggers. The end classifier simply classifies examples according to majority vote. In more refined models, the votes are weighted according to their accuracy.

The ensemble technique introduced here simply treats the combination of classifiers as a classification problem itself. The predicted classes of the input

classifiers, e.g. on a development section, are features to the end classifier. So, for example, the end classifier may learn rules of the form "if classifier A classifies i as x, and classifier B classifies i as y, i is likely to belong to the class z." Note that z does not have to equal x or y. Consequently, the oracle performance of our input classifiers, i.e. the performance of an oracle that always choses to rely on the best input classifier for any particular data instance, is not necessarily an upper bound of the performance of our end classifier.

Other advantages include that (i) our classifier can employ arbitrarily many features other than the input classifiers (Sect. 3.2). Consequently, it is more flexible. If our end classifier is confidence-rated, (ii) weights can be returned to our parsing algorithm for reparsing. Finally, (iii) the technique works even if you only have two input classifiers. (In fact, as shown in Sect. 3.4, it leads to improvements with only one input classifier.) It also seems to be more insensitive to the type of input classifiers. Bagging, for instance, only improves the performance of *unstable* learners, but the technique introduced here has been shown to work for a wide range of different learning algorithms, incl. both Naive Bayes and the k nearest neighbor algorithm (Sect 3.1).

The work most closely related to this is the so-called "contextual cues" in [1]. It seems that they essentially run a k nearest neighbor classifier on tags predicted by input classifiers. They do not consider other features, but for any word w_i in position i they also consider the predicted tags for words w_{i-1} and w_{i+1}. [6] and others also test second level classifiers, but typically without additional features, and with only small improvements over the best input tagger.

The structure of what follows: Sect. 2 describes how the submitted results were obtained. Sect. 3 describes possible improvements to our system, and finally, Sect. 4 summarizes the main contribution of this work.

2 Submitted results

As input or component taggers trained on the training data we used (a) the maximum-entropy-based POS tagger first described in [7] that comes with the maximum entropy library in [8], and (b) a version of the the transformation-based tagger introduced by Eric Brill [5]. Finally, we used the TreeTagger [9] with a pretrained model (different training data, different tag set). It is clear that this set of input classifiers would make majority voting pointless.

Our results and results of our component taggers on the development data:

Dev	**TA**	**UWTA**
MaxEnt	90.75	76.00
Brill	89.44	63.15
Tree(0)	93.96	89.32
Tree(1)	94.41	**90.36**
kNN(0)	94.70	88.98
kNN(1)	**94.78**	88.98

We implemented a classification tree algorithm and a k nearest neighbor algorithm and also tried to use both iteratively, i.e. where the end classifier's

predictions are used as input features in the subsequent round. Both algorithms converged on a near-optimal result after the second round (1). For this reason we chose to submit second round results for both algorithms.

The submitted results contain a serious bug. One of the features was simply omitted in the test data. In the table below, we present the official results (o) and the results obtained using our corrected procedure (c), incl. positions and positions wrt. unknown words (UW) out of the 11 runs submitted to the EVALITA 2009 POS Tagging Evaluation Task. Test results for the component taggers are also presented:

Test	**TA**(o)	**UWTA**(o)	Pos	UW-Pos
Tree(o)	91.60	86.03	10	6
kNN(o)	91.64	86.14	11	5
Tree(c)	91.54	85.91	11	8
kNN(c)	**93.27**	**87.17**	9	5
MaxEnt	89.21	70.79	-	-
Brill	89.67	66.09	-	-

3 Subsequent work

3.1 Other classifiers

In our subsequent work on this ensemble-based technique we first tested a wider set of simple learning algorithms. Bayes(k) is Naive Bayes with k-estimates, and the combinations in the last three rows combine weights of different learners weighted wrt. their predicted accuracy. Our best scores with these simple features take us to 5th position and 4th position for unknown words.

Test	**TA**	**UWTA**
Tree	92.01	86.60
kNN	93.19	87.06
Bayes(5)	93.60	87.40
Bayes(10)	93.64	87.40
Bayes(10)/Tree	93.39	88.09
Bayes(10)/kNN	**93.72**	87.74
Bayes(10)/kNN/Tree	93.47	**88.32**

3.2 Other features

Since our set-up is so flexible, it was easy to add additional features to our model too. In our subsequent experiments we made use of suffixes, unsupervised word clusters and WordNet semantic fields.

Suffixes For suffixes, we simply used the last three letters in each word.

Test: Suff	**TA**	**UWTA**
Tree	92.01	86.71
kNN	93.66	**88.77**
Bayes(10)/kNN	**93.88**	88.66
Bayes(10)/kNN/Tree	93.49	88.66

Unsupervised clusters For unsupervised clusters, we used the Java implementation of the Biemann unsupervised graph-based clustering algorithm [10], trained on a 9M sentence Italian corpus (Leipzig University).

Test: Clust	**TA**	**UWTA**
Tree	91.83	86.37
kNN	93.19	87.40
Bayes(10)/kNN	**93.58**	**87.97**
Bayes(10)/kNN/Tree	93.23	87.51

The unsupervised clusters apparently lead the classification tree off track, while it does help the k nearest neighbor algorithm's classification of unknown words. This apparently transfers to the ensemble of Naive Bayes with 10-estimates and k nearest neighbor, but the overall accuracy of this ensemble decreases a bit. The drop in classification tree performance hurts the bigger ensemble considerably.

Suffixes *and* unsupervised clusters were tested to see if the positive improvements were accummulative:

Test: Suff+Clust	**TA**	**UWTA**
Tree	91.83	86.83
kNN	93.39	88.43
Bayes(10)/kNN	**93.76**	**88.77**
Bayes(10)/kNN/Tree	93.45	88.55

Results are mixed. The smaller ensemble does accummulate precision wrt. unknown words, while the k nearest neighbor algorithm doesn't. Oddly enough we also obtain our best result wrt. unknown words for classification trees this far. Obviously this highlights the unstability of classification trees. Some experiments were conducted using random forests, but with non-competitive results.

WordNet In our next experiment we imported semantic fields from the Italian MultiWordNet [11]. Naïvely we used the first semantic field for the first synsem associated with a particular lemma in the database. No lemmatization was used, so very few features were extracted. In spite of this serious limitation, the WordNet features did improve accuracy considerably:

Test: WordNet	**TA**	**UWTA**
Tree	92.05	86.60
kNN	93.41	87.51
Bayes(10)/kNN	**93.66**	**88.09**
Bayes(10)/kNN/Tree	93.45	87.86

Combining suffixes and WordNet synsems gave some of our best results:

Test: WordNet+Suff	**TA**	**UWTA**
Tree	92.01	86.71
kNN	93.62	**89.32**
Bayes(10)/kNN	**93.92**	88.89
Bayes(10)/kNN/Tree	93.58	88.32

3.3 Better input classifiers

SVMTool [12] (model 4) in its default setting obtains much better results on both the development section (92.30/72.79) and test section (92.09/72.74) than any of our original input taggers. Since SVMTool has relatively low accuracy wrt. unknown words, best results were obtained using unsupervised clusters:[1]

Test: WordNet+Suff+Clust	**TA**	**UWTA**
Tree	93.07	84.54
kNN	94.04	**89.35**
Bayes(10)/kNN	94.00	88.43
Bayes(10)/kNN/Tree	94.00	87.63

3.4 Semi-supervised learning

In this experiment, we only used two features, words and SVMTool predictions. SVMTool was first run on the first part of our Italian corpus; the excerpt chosen equals our training data in size (∼150k). We then trained three (n in the general case) classifiers – Bayes(3), Tree and kNN – on this unlabeled data and adopted the following simple semi-supervised learning strategy: if two $(n-1)$ classifiers agree on a data instance, assign it their prediction and include it in the training data of the third (nth) classifier. The procedure is continued until no more examples are added to the labeled data sets. The strategy is tentatively called *n-ary co-training*. Only overall accuracies are listed, along with the number of labeled examples in the initial and the final round. Improvements in average score are modest, but noticeable.

	Round 1	Labeled data	Round 5	Labeled data
Bayes(3)	93.23	4867	93.25	6167
Tree	93.31	4867	93.27	5567
kNN	92.62	4867	93.07	9937
AV	93.05	4867	93.20	7224

3.5 Future work

Some of our input classifiers have very low accuracy wrt. unknown words. It might improve overall accuracy if we used a *simpler model to classify unknown words*, i.e. removing less probable predictions (MaxEnt and Brill).

Our model does not *reparse* the input using our confidence-rated classifier. This would no doubt also increase overall performance.

Even in the absence of reparsing, it is possible to do *reranking* of a set of high-quality hypotheses. Say, for instance, a threshold $0 < \tau < 1$ is introduced such that if the weight associated with the 2nd best hypothesis $\mathbf{w}(h_2)$ for a word is

[1] It is important to note the importance of classifier diversity. If in the current set-up the Brill tagger is replaced with SVMTool model 0, alongside SVMTool model 4, which produce very similar results, our accuracy wrt. unknown words decreases considerably.

close to that of the best one, i.e. $\frac{\mathbf{w}(h_2)}{\mathbf{w}(h_1)} \geq \tau$, all candidate labelings of a sentence with the word's best and 2nd best hypotheses are ranked and included in an (average) n-best list. The candidate list can be limited to the n best hypotheses on average by varying the threshold.

Finally, the entire official training section is used for training our input classifiers, and only the development section is used for adjusting the weights of our end classifier. A more balanced split might also improve our overall accuracy.

4 Conclusion

Some corrected and improved results were presented for the ensemble-based system we used in the EVALITA 2009 POS Tagging Evaluation Task. Using our poor input classifiers, our best results rank 5 in general (out of 11 submitted runs) and 4 wrt. unknown words; our official results rank 10 and 5, resp.

The main point of this work, however, was to provide a proof-of-concept for classification-based ensemble learning. The advantages of this approach can be summarized as: (i) flexible inclusion of features, (ii) improvements over best input classifier even with only two classifiers (unlike majority voting), (iii) no upper bound from oracle performance, (iv) applicability to both stable and unstable learning algorithms.

References

1. Brill, E., Wu, J.: Classifier combination for improved lexical disambiguation. In: Proceedings of COLING-ACL (1998)
2. Abney, S., Schapire, R., Singer, Y.: Boosting applied to tagging and PP-attachment. In: Proceedings of EMNLP (1999)
3. Marquez, L., Rodriguez, H., Carmona, J., Montolio, J.: Improving POS tagging using machine-learning techniques. In: Proceedings of EMNLP (1999)
4. Schapire, R.: The strength of weak learnability. Machine Learning, vol. 5, issue 2 (1990)
5. Brill, E.: Transformation-based error-driven learning and natural language processing. Computational Linguistics, vol. 21, issue 4 (1995)
6. Sjoebergh, J.: Combining POS-taggers for improved accuracy on Swedish text. In: Proceedings of NODALIDA (2003)
7. Ratnaparkhi, A.: Maximum entropy models for natural language ambiguity resolution. PhD thesis, University of Pennsylvania (1998)
8. Zhang, L.: Maximum entropy modeling toolkit for Python and C++. University of Edinburgh (2004)
9. Schmidt, H.: Improvements in part-of-speech tagging with an application to German. In: Proceedings of ACL-SIGDAT (1995)
10. Biemann, C.: Unsupervised part-of-speech tagging employing efficient graph clustering. In: Proceedings of COLING/ACL Student Session (2006)
11. Pianta, E., Bentivogli, L., Girardi, C.: MultiWordNet: developing an aligned multilingual database. In: Proceedings of the 1st International Global WordNet Conference (2002)
12. Gimenez, J., Marquez, L.: SVMTool: a general POS tagger generator based on support vector machines. In: Proceedings of LREC (2004)