Boosting Applied to Word Sense Disambiguation

Gerard Escudero, Lluís Màrquez, and German Rigau

TALP Research Center
Software Department. Catalan Polytechnical University
Jordi Girona Salgado 1–3. E-08034, Barcelona. Catalonia
{escudero,lluism,g.rigau}@lsi.upc.es

Abstract. In this paper Schapire and Singer's AdaBoost.MH boosting algorithm is applied to the Word Sense Disambiguation (WSD) problem. Initial experiments on a set of 15 selected polysemous words show that the boosting approach surpasses Naive Bayes and Exemplar–based approaches, which represent state–of–the–art accuracy on supervised WSD. In order to make boosting practical for a real learning domain of thousands of words, several ways of accelerating the algorithm by reducing the feature space are studied. The best variant, which we call LazyBoosting, is tested on a medium/large sense–tagged corpus containing 192,800 examples of the 191 most frequent and ambiguous English words. Again, boosting compares favourably to the other benchmark algorithms.

Keywords: Natural Language Learning, Word Sense Disambiguation, Supervised Learning, Boosting algorithms.

Type of submission: Long paper

Word count: about 5,800 words

Contact person: Lluís Màrquez
lluism@lsi.upc.es
Tel: +34 93 4015652
Fax: +34 93 4017014

Address: Mòdul C6, Campus Nord
Universitat Politècnica de Catalunya (UPC)
Jordi Girona Salgado 1–3
Barcelona E-08034, Catalonia (Spain)

1 Introduction

1.1 Word Sense Disambiguation

Word Sense Disambiguation (WSD) is the problem of assigning the appropriate meaning (sense) to a given word in a text or discourse. This meaning is distinguishable from other senses potentially attributable to that word. Resolving the ambiguity of words is a central problem for language understanding applications and their associated tasks [12], including, for instance, machine translation, information retrieval and hypertext navigation, parsing, speech synthesis, spelling correction, reference resolution, automatic text summarization, etc.

WSD is one of the most important open problems in the Natural Language Processing (NLP) field. Despite the wide range of approaches investigated and the large effort devoted to tackle this problem, it is a fact that to date no large–scale, broad coverage and highly accurate word sense disambiguation system has been built.

The most successful current line of research is the corpus–based approach in which statistical or Machine Learning (ML) algorithms have been applied to learn statistical models or classifiers from corpora in order to perform WSD. Generally, supervised approaches (those that learn from a previously semantically annotated corpus) have obtained better results than unsupervised methods on small sets of selected highly ambiguous words, or artificial pseudo–words. Many standard ML algorithms for supervised learning have been applied, such as: Naive Bayes [21, 23], Exemplar–based learning [20, 21, 10], Decision Lists [31], Neural Networks [28], etc. Further, Mooney [19] provides a comparative experiment on a very restricted domain between all previously cited methods but also including Decision Trees and Rule Induction algorithms. Unfortunately, there have been very few direct comparisons of alternative methods on identical test data. However, it is commonly accepted that Naive Bayes, Neural Networks and Exemplar–based learning represent state–of–the–art accuracy on supervised WSD.

Supervised methods suffer from the lack of widely available semantically tagged corpora, from which to construct really broad coverage systems. This is known as the "knowledge acquisition bottleneck" [11]. Ng [22] estimates that the manual annotation effort necessary to build a broad coverage semantically annotated corpus would be about 16 man-years. This extremely high overhead for supervision and, additionally, the also serious overhead for learning/testing many of the commonly used algorithms when scaling to real size WSD problems, explain why supervised methods have been seriously questioned.

Due to this fact, recent works have focused on reducing the acquisition cost as well as the need for supervision of corpus–based methods for WSD. Consequently, the following three lines of research can be found: 1) The design of efficient example sampling methods [7, 10]; 2) The use of lexical resources, such as WordNet [18], and WWW search engines to automatically obtain from Internet arbitrarily large samples for word senses [13, 17]; 3) The use of unsupervised EM–like algorithms for estimating the statistical model parameters [23]. It is also our belief that this body of work, and in particular the second line, provide

enough evidence towards the "opening" of the acquisition bottleneck in the near future. For that reason, it is worth further investigating the application of new supervised ML methods to better resolve the WSD problem.

1.2 Boosting algorithms

The main idea of boosting algorithms is to combine many simple and moderately accurate hypotheses (called weak classifiers) into a single, highly accurate classifier for the task at hand. The weak classifiers are trained sequentially and, conceptually, each of them is trained on the examples which were most difficult to classify by the preceding weak classifiers.

The AdaBoost.MH algorithm applied in this paper [26] is a generalization of Freund and Schapire's AdaBoost algorithm [9], which has been (theoretically and experimentally) studied extensively and which has been shown to perform well on standard machine–learning tasks using also standard machine–learning algorithms as weak learners [24, 15, 8, 9, 6, 2].

Regarding Natural Language (NL) problems, AdaBoost.MH has been successfully applied to Part–of–Speech (PoS) tagging [1], Prepositional–Phrase–attachment disambiguation [1], and, Text Categorization [27] with especially good results.

The Text Categorization domain shares several properties with the usual settings of WSD, e.g.: very high dimensionality (typical features consist on testing the presence/absence of concrete words), presence of many irrelevant and highly dependent features, and the fact that both, the learned concepts and the examples, reside very sparsely in the feature space. Therefore, the application of AdaBoost.MH also to WSD seems to be a promising choice. It has to be noted that, apart from the excellent results obtained on NL problems, AdaBoost.MH has the advantages of being theoretically well founded and relatively easy to implement.

The paper is organized as follows: Section 2 is devoted to explain in detail the AdaBoost.MH algorithm. Section 3 describes the domain of application and the initial experiments performed on a reduced set of words. In Section 4 several alternatives are explored for accelerating the learning process by reducing the feature space of the WSD problem. The best alternative is fully tested in Section 5. Finally, Section 6 concludes and outlines some directions for future work.

2 The Boosting Algorithm AdaBoost.MH

In this section the Schapire and Singer's AdaBoost.MH algorithm is described for multiclass multi–label classification, using exactly the same notation given by the authors in [26, 27].

As previously said in the introduction, the purpose of boosting is to find a highly accurate classification rule by combining many weak hypotheses (or weak rules), each of which may be only moderately accurate. It is assumed the

existence of a separate procedure called the WeakLearner for acquiring the weak hypotheses. The boosting algorithm finds a set of weak hypotheses by calling the weak learner repeatedly in a series of T rounds. These weak hypotheses are then combined into a single rule called the combined hypotheses.

Let $S = \{(x_1, Y_1), \ldots, (x_m, Y_m)\}$ be the set of m training examples, where each instance x_i belongs to a instance space $\mathcal{X}$ and each Y_i is a subset of a finite set of labels or classes $\mathcal{Y}$. The size of $\mathcal{Y}$ is denoted by $k = |\mathcal{Y}|$.

The pseudo–code of AdaBoost.MH is presented in figure 1[1]. AdaBoost.MH maintains an $m \times k$ matrix of weights as a distribution D over examples and labels. The goal of the WeakLearner algorithm is to find a weak hypothesis with moderately low error with respect to these weights. Initially, the distribution D_1 is uniform, but the boosting algorithm updates the weights on each round to force the weak learner to concentrate on the pairs (examples,label) which are hardest to predict.

---

procedure AdaBoost.MH (**in**: $S = \{(x_i, Y_i)\}_{i=1}^{m}$)

S is the set of training examples

Initialize distribution D_1 (for all i, $1 \leq i \leq m$, and all l, $1 \leq l \leq k$)

$D_1(i, l) = 1/(mk)$

for t:=1 **to** T **do**

Get the weak hypothesis $h_t : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$

h_t = WeakLearner (X, D_t);

Update distribution D_t (for all i, $1 \leq i \leq m$, and all l, $1 \leq l \leq k$)

$$D_{t+1}(i, l) = \frac{D_t(i, l)\exp(-Y_i[l]h_t(x_i, l))}{Z_t}$$

Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution)

end-for

return the combined hypothesis: $f(x, l) = \sum_{t=1}^{T} h_t(x, l)$

end AdaBoost.MH

Fig. 1. The AdaBoost.MH algorithm

More precisely, let D_t be the distribution at round t, and $h_t : \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ the weak rule acquired according to D_t. The sign of $h_t(x, l)$ is interpreted as a prediction as to whether the label l should or should not be assigned to the example x. The magnitude of the prediction $|h_t(x, l)|$ is interpreted as a measure of confidence in the prediction. To properly understand the updating formula we need to define this last piece of notation. Given $Y \subseteq \mathcal{Y}$ and $l \in \mathcal{Y}$,

[1] The general formulation of AdaBoost.MH takes into account an additional parameter α_t. However, the way in which the WeakLearner will be defined implies that α_t must be set to 1 [27], and so it has no effect at all.

$$Y[l] = \begin{cases} +1 & \text{if } l \in Y \\ -1 & \text{if } l \notin Y \end{cases}$$

Now, it becomes clear that the updating function increases (decreases) the weights $D_t(i,l)$ for which h_t makes a good (bad) prediction, and that this variation is proportional to $|h_t(x,l)|$.

Note that WSD is not a multi–label classification problem since a unique sense is expected for each word in context. In our implementation, the algorithm runs exactly in the same way as explained above, except that sets Y_i are reduced to a unique label, and that the combined hypothesis is forced to output a unique label, which is the one that maximizes $f(x,l)$.

Up to now, it only remains to be defined the form of the WeakLearner. Schapire and Singer [26] prove that the Hamming loss of the AdaBoost.MH algorithm on the training set (i.e. the fraction of training examples i and labels l for which the sign of $f(x_i,l)$ differs from $Y_i[l]$) is at most $\prod_{t=1}^{T} Z_t$, where Z_t is the normalization factor computed on round t. This upper bound is used in guiding the design of the WeakLearner algorithm, which attempts to find a weak hypothesis h_t that minimizes:

$$Z_t = \sum_{i=1}^{m} \sum_{l \in \mathcal{Y}} D_t(i,l) \exp(-Y_i[l] h_t(x,l)) \, .$$

2.1 Weak Hypotheses for WSD

As in [1], very simple weak hypotheses are used that test the value of a boolean predicate and make a prediction based on that value. The predicates used, which are described in section 3.1, are of the form "$f = v$" where f is a feature and v is a value (e.g.: "previous_word = *hospital*"). Formally, based on a given predicate p, our interest lies on weak hypotheses h which make predictions of the form:

$$h(x,l) = \begin{cases} c_{0l} & \text{if } p \text{ holds in } x \\ c_{1l} & \text{otherwise} \end{cases}$$

where the c_{jl}'s are real numbers.

For a given predicate p, and with the minimization of Z_t in mind, the values c_{jl} should be calculated as follows. Let X_1 be the subset of examples for which the predicate p holds and let X_0 be the subset of examples for which the predicate p does not hold. Let $[\![\pi]\!]$, for any predicate π, be 1 if π holds and 0 otherwise. Given the current distribution D_t, the following real numbers are calculated for each possible label l, for $j \in \{0,1\}$, and for $b \in \{+1,-1\}$:

$$W_b^{jl} = \sum_{i=1}^{m} D_t(i,l) [\![x_i \in X_j \wedge Y_i[l] = b]\!] \, .$$

In words, W_{+1}^{jl} (W_{-1}^{jl}) is the weight (with respect to the distribution D_t) of the training examples in partition X_j which are (are not) labelled by l.

It can be shown [26] that Z_t is minimized for a particular predicate by choosing:

$$c_{jl} = \tfrac{1}{2} \ln\left(\frac{W_{+1}^{jl}}{W_{-1}^{jl}}\right) .$$

These settings imply that:

$$Z_t = 2 \sum_{j \in \{0,1\}} \sum_{l \in \mathcal{Y}} \sqrt{W_{+1}^{jl} W_{-1}^{jl}}\,.$$

Thus, the predicate p chosen is that for which the value of Z_t is smallest.

Very small or zero values for the parameters W_b^{jl} cause c_{jl} predictions to be large or infinite in magnitude. In practice, such large predictions may cause numerical problems to the algorithm, and seem to increase the tendency to overfit. As suggested in [27], smoothed values for c_{jl} have been used (with the ϵ parameter set to $1/mk$).

3 Applying Boosting to WSD

3.1 Corpus

In our experiments the boosting approach has been evaluated using a corpus containing 192,800 semantically occurrences annotated[2] of 121 nouns and 70 verbs. These correspond to the most frequent and ambiguous English words. This corpus was collected by Ng and colleagues [20] and it is available from the Linguistic Data Consortium (LDC)[3].

There has been a lot of debate about the convenience of using WordNet senses as a reference for the WSD task. In particular, the granularity of such senses seems to be a drawback. The philosophical and practical questions about what is the correct definition of a sense fall beyond the scope of this work, which simply aims to evaluate some supervised learning algorithms on a medium/large size semantically tagged corpus.

For our first experiments, a group of 15 words (10 nouns and 5 verbs) which frequently appear in the related WSD literature has been selected . These words are described in the left hand–side of table 1. Since our goal is to acquire a classifier for each word, each row represents a classification problem. The number of classes (senses) ranges from 4 to 30, the number of training examples ranges from 373 to 1,500 and the number of attributes ranges from 1,420 to 5,181. The MFS column of the right hand–side of table 1, shows the percentage of the most frequent sense for each word, i.e. the accuracy that a naive "Most–Frequent–Sense" classifier would obtain.

The binary–valued attributes used for describing the examples correspond to the binarization of seven features referring to a very narrow linguistic context. Let "$w_{-2}\ w_{-1}\ w\ w_{+1}\ w_{+2}$" be the context of 5 consecutive words around the word w to be disambiguated. The seven features mentioned above are exactly those used in [21]: w_{-2}, w_{-1}, w_{+1}, w_{+2}, (w_{-2}, w_{-1}), (w_{-1}, w_{+1}), and (w_{+1}, w_{+2}), where the last three correspond to collocations of two consecutive words.

It is known that richer information contributes to improve the performance on the WSD task (e.g. part–of–speech tags of the neighbouring words, content

[2] These examples are tagged with a set of labels corresponding, with minor changes, to the senses of WordNet 1.5 [18].

[3] LDC address: `ldc@unagi.cis.upenn.edu`

words occurring in a broader window—even including information of previous and following sentences— syntactic relations, etc.). However, as in [21], our main goal is not to investigate which information leads to obtain the best results on the task but to compare the performance of different learning algorithms in a fixed setting.

3.2 Experimental Methodology and Benchmark Algorithms

AdaBoost.MH has been compared to the following algorithms:

- **Naive Bayes.** The naive Bayesian classifier has been used in its most classical setting [5]. To avoid the effect of zero counts when estimating the conditional probabilities of the model, a very simple smoothing technique has been used, which was proposed in [21].Hereinafter, this naive Bayesian classifier will be referred to as NB.
- **Exemplar–based learning.** In our implementation all examples are stored in memory and the classification of a new example is based on a k–NN algorithm, which uses Hamming distance to measure closeness (in doing so, all examples are examined). If k is greater than 1, the resulting sense is the weighted majority sense of the k nearest neighbours (each example votes its sense with a strength proportional to its closeness to the test example). Ties are resolved in favour of the most frequent sense among all those tied. Hereinafter, this algorithm will be referred to as EB_k.

The comparison of algorithms has been performed in series of controlled experiments using exactly the same training and test sets for each method. The experimental methodology consisted on a 10-fold cross-validation. All accuracy/error rate figures appearing in the paper are averaged over the results of the 10 folds. The statistical tests of significance have been performed using a 10-fold cross validation paired Student's t-test [4] with a confidence value of: $t_{9,0.975} = 2.262$.

3.3 Results

Figure 2 shows the error rate curve of AdaBoost.MH, averaged over the 15 reference words, for an increasing number of weak rules per word. This plot shows that the error obtained by AdaBoost.MH is lower than those obtained by NB and EB_{15} (k=15 is the best choice for that parameter from a number of tests between k=1 and k=30) for a number of rules above 100, and that the error rate monotonically, but also slightly, decreases, as it approaches the maximum number of rules reported[4].

According to the plot in figure 2, it seems that there is no overfitting while increasing the number of rules per word. However, if the same curves are studied for individual words (cf. figure 3) different behaviours can be observed. For the words of the first plot, the error rate is still decreasing at the point of 250 rules. However, for the words of the second plot the error rate observes no variation

[4] The maximum number of rounds considered is 750, merely for efficiency reasons.

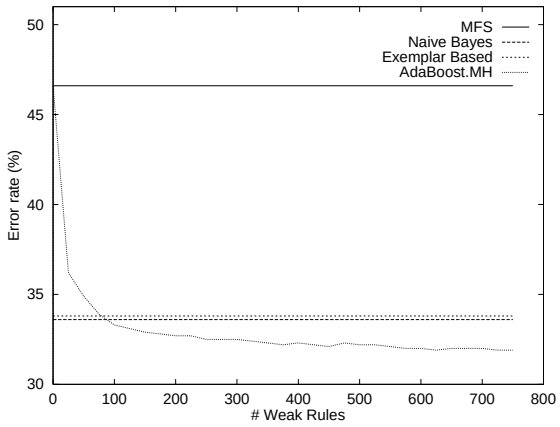

Fig. 2. Error of AdaBoost.MH related to the number of weak rules

and slightly oscillates over the optimal value, while for the words of the third plot a slight overfitting is observed and the error rate increases above 50–75 rules.

These results suggest that the number of rules to acquire should be adjusted for each individual word. To see the potential improvement that such an adjustment could provide the accuracy obtained has been calculated by selecting the best number of rules (between 25 and 750, with steps of 25) for each fold of each word. These optimal choices give an upper bound of 69.72% for the accuracy of AdaBoost.MH, which is significantly better than the 68.1% obtained by the algorithm when learning exactly 750 rules per word. Hereinafter, the latter will be called AB_{750}.

The adjustment of the number of rounds can be done by cross–validation on the training set, as suggested in [1]. However, in our case this cross–validation inside the cross–validation of the general experiment would generate a prohibitive overhead. Instead, a very simple stopping criterion (sc) has been used, which consists on stopping the acquisition of weak rules whenever the error rate on the training set falls below 5%, with an upper bound of 750 rules. This variant, which is referred to as AB_{sc}, obtained comparable results to AB_{750} but generating only 370.2 weak rules per word on average, which represents a very moderate storage requirement for the combined classifiers.

The numerical information corresponding to this experiment is included in table 1. This table shows the accuracy results, detailed for each word, of NB, EB_1, EB_{15}, AB_{750}, and AB_{sc}. The best result for each word is printed in boldface.

As it can be seen, in 14 out of 15 cases the best results correspond to the boosting algorithms. When comparing global results, accuracies of either AB_{750} or AB_{sc} are significantly greater than those of any of the other methods. Finally, note that accuracies corresponding to NB and EB_{15} are comparable (as suggested in [21]), and that the use of k's greater than 1 is crucial for making Exemplar–based learning competitive on WSD.

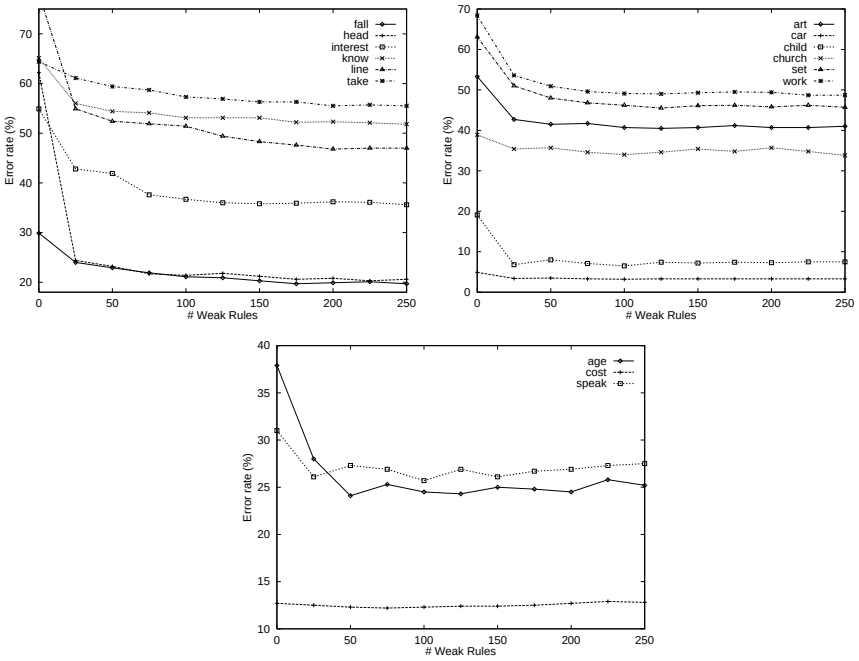

Fig. 3. Error of AdaBoost.MH on each word related to the number of weak rules

4 Making Boosting Practical for WSD

Up to now, it has been seen that AdaBoost.MH is a simple and competitive algorithm for the WSD task, achieving an accuracy performance superior to that of the Naive Bayes and Exemplar–based algorithms tested in this paper. However, AdaBoost.MH has the drawback of its computational cost, which makes the algorithm not scale properly to real WSD domains of thousands of words.

The space and time–per–round requirements of AdaBoost.MH are $\mathcal{O}(mk)$ (recall that m is the number of training examples and k the number of senses), not including the call to the weak learner. This cost is unavoidable since AdaBoost.MH is inherently sequential (in order to learn the $(t+1)$-th weak rule it needs the calculation of the t-th weak rule, which properly updates the matrix D_t). Further, inside the WeakLearner there is another iterative process that examines, one by one, all attributes to decide which is the one that minimizes Z_t. Since there are thousands of attributes this is also a time consuming part, which can be straightforwardly spedup either by reducing the number of attributes or by relaxing the need of examining all attributes at each iteration.

4.1 Accelerating the WeakLearner

Four methods for reducing the cost of searching weak rules have been tested. The first three, consisting on aggressively reducing the feature space, are frequently applied by Text Categorization systems [30]. The fourth consists on reducing

		Number of			Accuracy (%)					
Word	POS	Senses	Examp.	Attrib.	MFS	NB	EB_1	EB_{15}	AB_{750}	AB_{sc}
age	n	4	493	1662	62.1	73.8	71.4	71.0	**74.7**	74.0
art	n	5	405	1557	46.7	54.8	44.2	58.3	57.5	**62.2**
car	n	5	1381	4700	95.1	95.4	91.3	95.8	**96.8**	96.5
child	n	4	1068	3695	80.9	86.8	82.3	89.5	**92.8**	92.2
church	n	4	373	1420	61.1	62.7	61.9	63.0	**66.2**	64.9
cost	n	3	1500	4591	87.3	86.7	81.1	87.7	87.1	**87.8**
fall	v	19	1500	5063	70.1	76.5	73.3	79.0	**81.1**	80.6
head	n	14	870	2502	36.9	76.9	70.0	76.9	**79.0**	**79.0**
interest	n	7	1500	4521	45.1	64.5	58.3	63.3	**65.4**	65.1
know	v	8	1500	3965	34.9	47.3	42.2	46.7	**48.7**	**48.7**
line	n	26	1342	4387	21.9	51.9	46.1	49.7	**54.8**	54.5
set	v	19	1311	4396	36.9	**55.8**	43.9	54.8	**55.8**	**55.8**
speak	v	5	517	1873	69.1	**74.3**	64.6	73.7	72.2	73.3
take	v	30	1500	5181	35.6	44.8	39.3	46.1	**46.7**	46.1
work	n	7	1469	4923	31.7	**51.9**	42.5	47.2	50.7	50.7
Avg.	nouns	8.6	1040.1	3978.5	57.4	71.7	65.8	71.1	**73.5**	73.4
	verbs	17.9	1265.6	4431.9	46.6	57.6	51.1	58.1	**59.3**	59.1
	all	12.1	1115.3	4150.0	53.3	66.4	60.2	66.2	**68.1**	68.0

Table 1. Set of 15 reference words and results of the main algorithms

the number of attributes that are examined at each iteration of the boosting algorithm.

1. **Frequency filtering.** This method consists on simply discarding those features corresponding to events that occur less than N times in the training corpus. The idea beyond that criterion is that frequent events are more informative than rare events. This method will be referred to as "Freq" in the following experiment.
2. **Local frequency filtering.** This method works similarly to Freq but considers the frequency of events locally, at the sense level. More particularly, it selects the N most frequent features of each sense. This technique [29] will be referred to as "LFreq" hereinafter.
3. **RLM ranking.** This third method consists on making a ranking of all attributes according to the RLM distance measure [14], which has been commonly used for attribute selection in decision tree induction algorithms, and selecting the N most relevant features. This measure, belonging to the distance–based and to the information–based families of attribute selection functions, has been selected because it showed better performance than seven other alternatives in an experiment of decision tree induction for PoS tagging [16].
4. LazyBoosting. The last method does not filter out any attribute but reduces the number of them that are examined at each iteration of the boosting algorithm. More specifically, a small proportion p of attributes are randomly selected and the best weak rule is selected among them. The idea in this method is that if the proportion p is not too small probably a sufficiently

good rule can be found at each iteration. Besides, the chance for a good rule to appear in the whole learning process is very high. Another important characteristic is that no attribute has to be discarded and so we avoid the risk of eliminating relevant attributes or attributes that may contribute to marginally improve overall performance. This method will be referred to as LazyBoosting in reference to the work by Samuel and colleagues [25], who applied the same technique for accelerating the learning algorithm in a Dialogue Act tagging system.

4.2 Comparing Methods

The four methods above have been compared on the set of 15 reference words. Figure 4 contains the average error–rate curves obtained by the four variants at increasing levels of attribute reduction. The top horizontal line corresponds to the MFS error rate, while the bottom horizontal line stands for the error rate of AdaBoost.MH working with all attributes. The results contained in figure 4 are calculated running the boosting algorithm 250 rounds for each word.

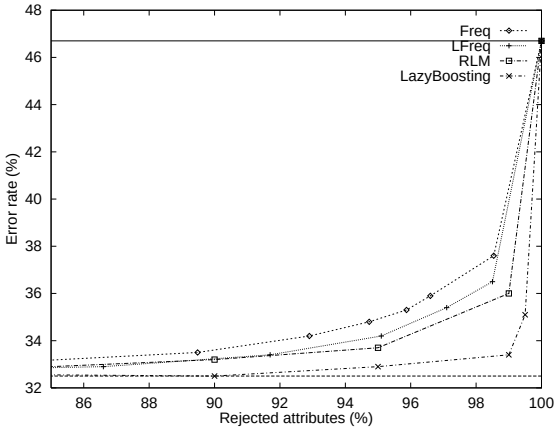

Fig. 4. Error obtained by the four methods, at 250 weak rules per word, with respect to the percentage of rejected attributes

The main conclusions that can be drawn are the following:

- All methods seem to work quite well since no important degradation is observed in performance for values lower than 95% in rejected attributes. This may indicate that there are many irrelevant or highly dependent attributes in our domain.
- LFreq is slightly better than Freq, indicating a preference to make frequency counts for each sense rather than globally.
- The more informed RLM ranking performs better than frequency–based reduction methods Freq and LFreq.

– LazyBoosting is better than all other methods, confirming our expectations: it is worth keeping all information provided by the features. In this case, acceptable performance is obtained even if only 1% of the attributes is explored when looking for a weak rule. The value of 10%, for which LazyBoosting still achieves the same performance and runs about 7 times faster than AdaBoost.MH working with all attributes, will be selected for the experiments in section 5.

Observing the same comparative plots obtained when training the boosting algorithm with a smaller number of rounds (cf. figure 5) helps understanding the reason why LazyBoosting works so well. When training with only 25 weak rules per word, the chance for the best rules to appear in the combined classifier is still small and so the LazyBoosting algorithm performs worse than all the rest (which work with their subset of selected best attributes). Increasing the number of weak rules per word makes also increase the proportion of good rules appearing in the combined classifier and, therefore, its performance. Training with 100 rounds per word (cf. 2nd plot in figure 5) represents the threshold from which LazyBoosting becomes the best choice.

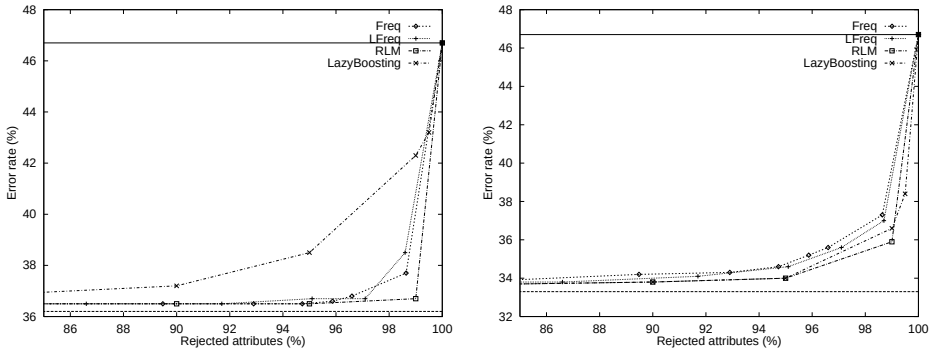

Fig. 5. Error obtained by the four methods, at 25 and 100 weak rules per word (1st and 2nd plot respectively), related to the percentage of rejected attributes

5 Evaluating LazyBoosting

The LazyBoosting algorithm has been tested on the full semantically annotated corpus with $p = 10\%$ and the same stopping criterion described in section 3.3, which will be referred as AB_{l10sc}. As previously said, this corpus contains examples of 121 nouns and 70 verbs. The average number of senses is 7.2 for nouns, 12.6 for verbs, and 9.2 overall. The average number of training examples is 933.9 for nouns, 938.7 for verbs, and 935.6 overall.

The AB_{l10sc} algorithm learned an average of 381.1 rules per word, and took about 4 days of CPU time to complete[5]. It has to be noted that this time includes the cross–validation overhead. Eliminating it, it is estimated that 4 CPU days would be the necessary time for acquiring a word–sense disambiguation boosting–based system covering about 2,000 words.

The AB_{l10sc} has been compared again to the benchmark algorithms using the 10-fold cross validation methodology described in section 3.2. The average accuracy results are reported in the left hand–side of table 2. The best figures correspond to the LazyBoosting algorithm AB_{l10sc}, and again the differences are statistically significant using the 10-fold cross validation paired t test.

	Accuracy (%)				Wins–Ties–Losses	
	MFS	NB	EB_{15}	AB_{l10sc}	AB_{l10sc} vs. NB	AB_{l10sc} vs. EB_{15}
Nouns (121)	56.4	68.7	68.0	**70.8**	99(51) – 1 – 21(3)	100(68) – 5 – 16(1)
Verbs (70)	46.7	64.8	64.9	**67.5**	63(35) – 1 – 6(2)	64(39) – 2 – 4(0)
Average (191)	52.3	67.1	66.7	**69.5**	162(86) – 2 – 27(5)	164(107) – 7 – 20(1)

Table 2. Results of LazyBoosting and the benchmark methods on the 191–word corpus

The right hand–side of the table shows the comparison of AB_{l10sc} versus NB and EB_{15} algorithms, respectively. Each cell contains the number of wins, ties, and losses of competing algorithms. The counts of statistically significant differences are included in brackets. It is important to point out that EB_{15} only beats significantly AB_{l10sc} in one case while NB does so in five cases. Conversely, a significant superiority of AB_{l10sc} over EB_{15} and NB is observed in 107 and 86 cases, respectively.

Figure 6 shows again the same comparative information in the form of scatter plots. Each point in the scatter plot represents the accuracy achieved by the two competing algorithms on a given word, so there is one point for each of the 191 words. As it can be observed, the larger number of points lies above the line $y = x$ in both plots, which indicates that the accuracy of LazyBoosting is superior to that of NB and EB_{15}, respectively.

6 Conclusions and Future Work

In the present work, Schapire and Singer's AdaBoost.MH algorithm has been evaluated on the word–sense disambiguation task, which is one of the hardest open problems in Natural Language Processing. As it has been shown, the boosting approach outperforms Naive Bayes and Exemplar–based learning, which represent state–of–the–art accuracy on supervised WSD. In addition, a faster variant has been suggested and tested, which is called LazyBoosting. This allows

[5] The current implementation is written in PERL-5.003 and it was run on a SUN UltraSparc2 machine with 194Mb of RAM.

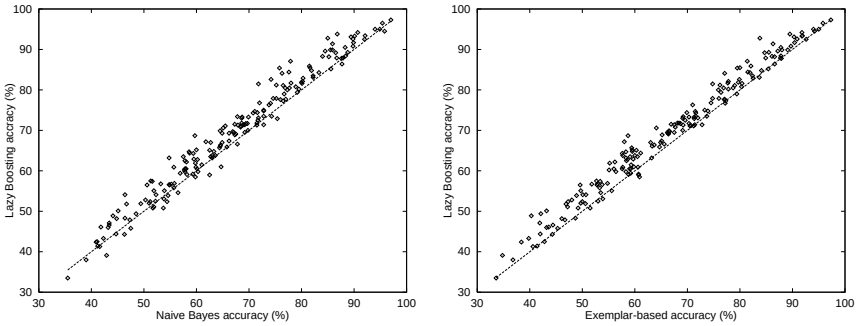

Fig. 6. Comparison of LazyBoosting versus NB and EB_{15} algorithms

to scale the algorithm to broad-coverage real WSD domains, and is as accurate as AdaBoost.MH.

However, further work is still to be done in many directions, including:

- Extensively evaluate AdaBoost.MH on the WSD task. This would include taking into account additional attributes, and testing the algorithms in other manually annotated corpora, and especially on sense–tagged corpora automatically obtained from Internet.
- Confirm the validity of the LazyBoosting approach on other language learning tasks in which AdaBoost.MH works well, e.g.: Text Categorization.
- It is known that mislabelled examples resulting from annotation errors tend to be hard examples to classify correctly, and, therefore, tend to have large weights in the final distribution. This observation allows both to identify the noisy examples and use boosting as a way to improve data quality [27, 1]. It is suspected that the corpus used in the current work is very noisy, so it could be worth using boosting to try to improve it.

Acknowledgments

This research has been partially funded by the Spanish Research Department (CICYT's BASURDE project TIC98–0423–C06) and by the Catalan Research Department (CIRIT's consolidated research group 1997SGR 00051, CREL's Catalan WordNet project and CIRIT's grant 1999FI 00773).

References

[1] Abney, S., Schapire, R.E. and Singer, Y.: *Boosting Applied to Tagging and PP–attachment.* In Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora, 1999.

[2] Bauer, E. and Kohavi, R.: *An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting and Variants.* Machine Learning

Journal. Special issue on IMLM for Improving and Scaling Machine Learning Algorithms, 1999.

[3] Breiman, L.: *Arcing Classifiers.* The Annals of Statistics, 26(3):801-849, 1998.

[4] Dietterich, T.G.: *Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms.* Neural Computation, 10(7), 1998.

[5] Duda, R. O. and Hart, P. E.: *Pattern Classification and Scene Analysis.* Wiley, New York, 1973.

[6] Dietterich, T.G.: *An Experimental Comparison of Three Methods for Constructing Ensembles of Decision Trees: Bagging, Boosting, and Randomization.* Machine Learning, to appear.

[7] Engelson, S.P. and Dagan, I.: *Minimizing Manual Annotation Cost in Supervised Training from Corpora.* In S. Wermter, E. Riloff and G. Scheler, editors, Connectionist, Statistical an Symbolic Approaches to Learning for Natural Language Processing, LNAI, 1040. Springer, 1996.

[8] Freund, Y. and Schapire, R.E.: *Experiments with a New Boosting Algorithm.* In Machine Learning: Proceedings of the Thirteenth International Conference, pp. 148-156, 1996.

[9] Freund, Y. and Schapire, R.E.: *A Decision–theoretic Generalization of On-line Learning and an Application to Boosting.* Journal of Computer and System Sciences, 55(1):119-139, 1997.

[10] Fujii, A., Inui, K., Tokunaga, T. and Tanaka, H.: *Selective Sampling for Example–based Word Sense Disambiguation.* Computational Linguistics, 24(4):573-598, 1998.

[11] Gale, W., Church, K.W. and Yarowsky, D.: *A Method for Disambiguating Word Senses in a Large Corpus.* Computers and the Humanities, 26, 1992.

[12] Ide, N. and Véronis, J.: *Introduction to the Special Issue on Word Sense Disambiguation: The State of the Art.* Computational Linguistics, 24(1):1-40, 1998.

[13] Leacock, C., Chodorow, M. and Miller, G.A.: *Using Corpus Statistics and WordNet Relations for Sense Identification.* Computational Linguistics, 24(1):147-165, 1998.

[14] López de Mántaras, R.: *A Distance–based Attribute Selection Measure for Decision Tree Induction.* Machine Learning, 6(1):81–92, 1991.

[15] Maclin, R. and Opitz, D.: *An Empirical Evaluation of Bagging and Boosting.* In Proceedings of the 14th National Conference on Artificial Intelligence, AAAI, 1997.

[16] Màrquez, L.: *Part-of-speech Tagging: A Machine Learning Approach based on Decision Trees.* Phd. Thesis, Software Department, Catalan Polytechnical University, July 1999.

[17] Mihalcea, R. and Moldovan, I.: *An Automatic Method for Generating Sense Tagged Corpora.* In Proceedings of the 16th National Conference on Artificial Intelligence, AAAI, 1999.

[18] Miller, G.A., Beckwith, R., Fellbaum, C., Gross, D. and Miller, K.: *Five Papers on WordNet.* Special Issue of International Journal of Lexicography, 3(4), 1990.

[19] Mooney, R.J.: *Comparative Experiments on Disambiguating Word Senses: An Illustration of the Role of Bias in Machine Learning.* In Proceedings of the 1st Conference on Empirical Methods in Natural Language Processing, EMNLP, 1996.

[20] Ng, H.T. and Lee, H.B.: *Integrating Multiple Knowledge Sources to Disambiguate Word Senses: An Exemplar-based Approach.* In Proceedings of the 34th Annual Meeting of the Association for Computational Linguistics, ACL, 1996.

[21] Ng, H.T.: *Exemplar-based Word Sense Disambiguation: Some Recent Improvements.* In Proceedings of the 2nd Conference on Empirical Methods in Natural Language Processing, EMNLP, 1997.

[22] Ng, H.T.: *Getting Serious about Word Sense Disambiguation.* In Proceedings of the SIGLEX Workshop "Tagging Text with Lexical Semantics: Why, What and How?", 1997.

[23] Pedersen, T. and Bruce, R.: *Knowledge Lean Word-Sense Disambiguation.* In Proceedings of the 15th National Conference on Artificial Intelligence, AAAI, 1998.

[24] Quinlan, J.R.: *Bagging, Boosting and C4.5.* In Proceedings of the 13th National Conference on Artificial Intelligence, AAAI, 1996.

[25] Samuel, K.: *Lazy Transformation-Based Learning.* In Proceedings of the 11th International Florida Artificial Intelligence Research Symposium Conference, pp. 235-239, 1998.

[26] Schapire, R.E. and Singer, Y.: *Improved Boosting Algorithms Using Confidence-rated Predictions.* Machine Learning, to appear.

[27] Schapire, R.E. and Singer, Y.: *BoosTexter: A Boosting-based System for Text Categorization.* Machine Learning, to appear.

[28] Towell, G. and Voorhees, E.M.: *Disambiguating Highly Ambiguous Words.* Computational Linguistics, 24(1):125-145, 1998.

[29] Weiss, S., Apte, C., Damerau, F., Johnson, D., Oles, F., Goetz, T. and Hampp, T.: *Maximizing Text-mining Performance.* IEEE Intelligent Systems, 1999.

[30] Yang, Y. and Pedersen, J.P.: *Feature Selection in Statistical Learning of Text Categorization.* In the 14th International conference on Machine Learning, pp. 412-420, 1997.

[31] Yarowsky, D.: *Decision Lists for Lexical Ambiguity Resolution: Application to Accent Restoration in Spanish and French.* In Proceedings of the 32nd annual Meeting of the Association for Computational Linguistics, ACL, 1994.