WIL: a First Order Logic Weak Learner for Boosting

Marco Botta

Dipartimento di Informatica, Università di Torino, Corso Svizzera 185 – 10149 Torino – Italy
botta@di.unito.it

Abstract. Boosting is a powerful and thoroughly investigated learning technique that improves the accuracy of any given learning algorithm by weighting training examples and hypotheses. Several authors contributed to the general boosting learning framework with theoretical and experimental results, mainly, or better exclusively, in the propositional learning framework. In this paper, we investigate the applicability of Freund and Schapire's AdaBoost.M1 algorithm to a first order logic weak learner.

1 Introduction

This paper focuses on supervised learning from examples in first order logic languages [20,4], also called *relational learning* [23], which has been extensively studied from a theoretical and experimental point of view in Inductive Logic Programming (ILP) [21]. In particular, we are interested in the classification task of structured instances, i.e. the recognition of an object or event as an instance of a given class. Medical and fault diagnosis, prognosis, image recognition, text categorization, adaptive user profiling, can be seen as specific instances of such a classification task.

Encoding classification theories in First Order Logic (FOL) is appealing for two main reasons. First, FOL can face problems that cannot be reduced to propositional logics, such as learning from structured data of finite but unconstrained size, or handling recurrent structures. Second, even when a problem can be reduced to a propositional setting, the solutions found in FOL are often more abstract and simpler than the corresponding ones in propositional logics. A wide spectrum of learning algorithms, based on different approaches ranging from logic induction [22,23,5] to Artificial Neural Networks [6] and Genetic Algorithms [2,15], are now available for this purpose.

All these approaches favor readability and interpretability of the learned classification theories. However, it often happens that the learned knowledge is not as meaningful and understandable as expected by a human expert. Moreover, when error rates are high, it is questionable whether what has been learned should be considered "knowledge", thus how relevant is to obtain readable theories in these cases.

In the propositional framework, several recent approaches, such as boosting [14], bagging [8] and arching [9], favor, instead, classification accuracy by somehow combining the theories acquired by a so-called *weak learner*. These methods generally improved performances with respect to sophisticated learners [13], despite the readability of the learned classification theory.

In this paper, we investigate the effectiveness of one of the above-mentioned methods, namely boosting, in a first order logic setting. In learning problems where several structured features need be discovered and combined to obtain highly accurate hypotheses, powerful FOL learners require excessive computational resources to find such a solution, and even when enough resources can be allocated, there is no guarantee that a readable and meaningful solution is found. The main idea that guided us in the presented framework, is that a FOL weak learner can be used to learn simple structured features and boosting is used to find a good combination of these features.

To this aim, we first designed a simple weak learner, able to build classification theories expressed in a first order logic language. Then, we tried to boost it with the simplest AdaBoost algorithm [14] (AdaBoost.M1), renouncing to the readability of the classification theory. Quinlan [25] performed a similar study by boosting FFOIL, an adaptation of FOIL [23] for functional relations, but we reached a somewhat different conclusion by comparing the obtained results to the best published ones.

The paper is organized as follows: Section 2 introduces the weak learner WIL, Section 3 briefly overviews the AdaBoost.M1 algorithm, Section 4 describes the experimental setting and results obtained, and Section 5 presents some discussion and concludes the paper.

2 The Weak Learner

In order for the boosting methodology to be affordable and convenient, the weak learning algorithm should be fast. In the propositional framework, there are a quite large number of learning algorithms (see, for instance, [7,24]) that meet this constraint. In first order logic, learning algorithms tend to be quite slow, due to the complexity of the search space they have to explore. We devised an extremely straightforward learning algorithm, called WIL (Weak Inductive Learner), that builds a set of decision rules, resembling the way FOIL [23] works. First, let us briefly introduce the learning framework.

We assume the reader is familiar with standard logic programming terminology [19] and only recall here a few notions. Examples processed by WIL are represented by specifying their elementary components, called *objects*, plus a set of their *properties* such as "color" or "length", and *relationships*, such as "relative position". This representation is close to the one adopted in object oriented databases (see [16] for a more detailed description), and naturally regresses to attribute-value representation when instances are composed by a single object. Theories consists of sets of clauses with the following format:

$$Head \leftarrow L_1 \wedge \ldots \wedge L_n$$

where *Head* is a positive literal (a concept to learn) and each L_i is a positive or negated literal. Recursive clauses, i.e. clauses with the same literal both in the head and in the body, are not allowed. Moreover, clauses are *range-restricted*, i.e., variables in the head of a clause must also appear in its body.

A sketch of the algorithm is reported in Fig. 1.

Algorithm WIL

Input: set of N examples $<(Ex_1,y_1),\ldots,(Ex_N,y_N)>$ with labels $y_i \in Y = \{1,\ldots,k\}$. Note that examples may be structured.

```
   float γ specifying percentage of positive instances to cover
   a set Λ of literals
   integer μ specifying number of allowed literals in a clause (default ∞)
Set Theory = ∅
Do for j = 1, 2, …, k
    Set PosInstances = {(Ex_i,y_i) | y_i = y_j }, ToCover = |PosInstances|*γ
    While |PosInstances| > ToCover
        Set body = true
        Set clause = y_j ← body
        While clause covers some negative example ∧ |body| < μ
            find the literal L to add to the right-hand side of clause that maximizes an
                Information Gain measure. All variabilizations of each literal in Λ
                consistent with body are considered and scored in this step.
            update body = body ∧ L
        End
        Remove from PosInstances examples covered by clause
        Add clause to Theory: Theory = Theory ∪ {clause}
    End
Endfor
```

Fig. 1. The first-order weak learning algorithm WIL.

The inner loop of WIL is responsible to find a clause as consistent and complete as possible. Once such a clause is found, it is added to the theory, the PosInstances set is updated by removing the covered instances, and these steps repeated until enough positive instances are covered. This process is performed for each concept to learn, so WIL solves *k* binary problems (one-vs-all approach [1]) and simply combines their outputs according to the following scheme: a score, that measures how good a clause is, is associated to every clause; then, the label(s) predicted by clause(s) with the highest score is(are) associated to the instance. Notice that, in this way, multiple labels can be assigned to an instance and no tie breaking is needed. Moreover, it should be pointed out that by setting $\gamma = \frac{1}{|PosIntances|}$ only one clause is built in the inner loop, and by furthermore setting μ to 1, clauses will have only one literal, thus mimicking the *FindDecRule* and *FindAttrTest* algorithms [13], respectively, designed for the propositional framework.

3 Boosting

Boosting is a general method aimed at improving the accuracy of any given learning algorithm, by combining simple hypotheses which only perform slightly better than 50% error rate. The combined hypothesis is a weighted vote of the simple hypotheses and has a strong provable performance guarantee, in terms of its prediction error [26].

In [14], Freund and Schapire introduced the AdaBoost algorithm, which solved many of the practical difficulties of the earlier boosting algorithms. In particular, the AdaBoost.M1 algorithm is the base of the presented approach, and is sketched in Fig. 2.

Algorithm AdaBoost.M1

Input: set of N examples <(Ex_1,y_1),…,(Ex_N,y_N)> with labels $y_i \in Y = \{1,\ldots,k\}$
distribution D over the N examples
weak learner algorithm **WIL**
integer *T* specifying number of iterations

Initialize the weight vector $w_i^1 = D_i$ for i = 1,…,N.

Do for $t = 1, 2, \ldots, T$

1. Set $D^t = \dfrac{\mathbf{w}^t}{\sum_{i=1}^{N} w_i^t}$
2. Call **WIL** providing it with N resampled instances according to distribution D^t; get back a hypothesis $h_t : X \to Y$
3. Calculate the error of
$$h_t : \varepsilon_t = \sum_{i=1}^{N} D_i^t \left\| h_t(x_i) \neq y_i \right\| .$$
If $\varepsilon_t > ½$, then set $T = t-1$ and abort the loop.
4. Set $\beta_t = \varepsilon_t /(1-\varepsilon_t)$.
5. Set the new weights vector to be
$$w_i^{t+1} = w_i^t \beta_t^{1-\|h_t(x_i)\neq y_i\|}$$

Output the hypothesis
$$h_f(x) = \arg\max_{y \in Y} \sum_{t=1}^{T} \left(\log \frac{1}{\beta_t} \right) \left\| h_t(x) = y \right\|$$

Fig. 2. Simple multi-class version of AdaBoost. For any predicate π, $\|\pi\|$ is 1 if π holds, and 0 otherwise.

The algorithm takes as input a training set $(Ex_1,y_1), \ldots, (Ex_N,y_N)$ where each Ex_i belongs to an instance space X, and each label y_i is in some label set Y. A weak learning algorithm is called at every round to find a weak hypothesis $h_t : X \to Y$ appropriate for distribution D^t. The main idea is that, on each round, the weights of incorrectly classified examples are increased so that the weak learning algorithm is forced to focus on the hard examples in the training set. The goodness of a weak hypothesis is measured by its error
$$\varepsilon_t = \Pr_{i \sim D^t} \left[h_t(x_i) \neq y_i \right] = \sum_{i : h_t(x_i) \neq y_i} D_i^t .$$

The error is measured with respect to the distribution D^t on which the weak learner was trained. Once the weak hypothesis h_t has been received, AdaBoost.M1 chooses a parameter β_t measuring the importance of that hypothesis. The final hypothesis h_f for a given instance x outputs the label that maximizes the sum of the weights of the weak hypotheses predicting that label.

When the weak learner is not able to directly use the distribution D^t on the training examples, a subset of the training examples can be sampled according to D^t, and these

(unweighted) resampled examples can be used to train the weak learner. This is the strategy used to train our weak learner: at every round *t*, exactly N instances are randomly sampled with replacement according to distribution D^t. Moreover, in the algorithm reported in Fig. 2, ties (i.e., situation where more than one label has maximum sum of weights) are broken arbitrarily, whereas our implementation allows ambiguous classifications, as previously explained.

4 Experimental Results

We conducted a number of experiments on a collection of 5 first-order learning problems that show different properties and seemed to be a good test bench for the presented algorithm. Moreover, we experimentally checked the correctness of the implementation by running the algorithm on a propositional learning problem. A brief description of the selected learning problems follows.

Splice-junction. The first problem we considered, the *Splice Junctions* dataset, comes from molecular biology and has been provided by Jude Shavlik [28]. The problem is that of identifying boundaries between coding (exons) and non-coding (introns) regions of genes occurring in eukaryote DNA. The dataset consists of 3190 DNA sequences, collected from the GeneBank. Each sequence is represented as a string of length 60 from the alphabet {a,t,c,g}. The sequences are labeled according to three classes. Sequences containing a donor site belong to the E/I class (25% of the dataset); the ones containing an acceptor site to the class I/E (25% of the dataset); finally, sequences that do not contain any site belong to the N (Neither) class (50% of the dataset).

Mutagenesis. The first FOL problem we considered is represented by the *Mutagenesis* dataset, a challenging problem widely used in the ILP community for testing induction algorithms in FOL [18]. More recently, the problem has been proposed as a *real* Data Mining application [10]. The problem consists in learning rules for discriminating substances (aromatic and etheroaromatic nitro compounds occurring in car emissions) having carcinogenic properties on the basis of their chemical structure. In particular, the carcinogenicity of a molecule is known to be some how correlated to its mutagenic activity. So, the task consists in predicting the mutagenicity of a compound. The dataset contains 188 aromatic compounds (each example describes a single compound), of which 118 present positive levels of mutagenicity and therefore are labeled as positive examples, and the remaining 60 form the negative ones. The difficulty mainly lies in the complexity of matching formulas in FOL, which strongly limits the exploration capabilities of any induction system.

Train-check-out. This is an artificial dataset built to test the generalization ability of FONN [6], a learning algorithm that refines numerical constants in first order classification rules. The problem consists in deciding whether a train must not be allowed to transit on a given line (check-out procedure followed by a railway inspector), depending on the characteristics of the line. Two instances of the problem, *Trains2* and *Trains3* [6], of different complexity, have been considered in the experiments.

SpeechRecognition. The fourth learning problem concerns the recognition of the ten digits spoken in Italian [3] as isolated words, starting from the time evolution of two rough features, i.e., the zero-crossing and the total energy of the signal. The problem was chosen because it is sufficiently realistic to be a good test-bed, it is a really hard

instance of such a kind of problems and has been previously treated by the authors. The features are extracted from the signal using classical signal processing algorithms and are then described using a set of primitives, as proposed by DeMori et al. [11]. The learning set consists of 219 instances, while 100 instances are used for testing.

ArtificialCharacters. The last learning problem concerns an artificial dataset that bears many resemblances to a real-world one: ten capital letters of the English alphabet have been chosen and described in terms of segments, as though acquired from a tablet. Each segment is described by the initial and final coordinates (x,y) in a Cartesian plane. From these basic features, other features can be extracted, such as the length of a segment, its orientation, its preceding and following segments, and so on. Some of these features are numerical by nature, whereas others are categorical. This dataset has been used to extensively test the capabilities of Smart+ [5] running with several configurations. Here, we took the original dataset of 6000 instances[1] and split it into 6 folds of 1000 instances each (100 instances per class), that are used for learning, while an independent test set of 10000 instances (1000 per class), is used for testing.

Table 1 reports the results obtained on the 6 problems by running AdaBoost.M1 for 50 rounds, evaluating performances at the first (WIL no boosting) and last round (WIL + boosting). WIL is called with $\gamma = 0.7$ (so that it must cover at least 70% of positive instances). For *Trains2*, *Trains3* and *ArtificialCharacters*, AdaBoost.M1 has been run on each learning set, the learned theory tested on the testset and results are averaged. For *Splice-junctions* and *Speech*, we performed 10 runs of AdaBoost.M1 (remind that we use resampling) and averaged the results. Finally, for *Mutagenesis* we performed 10-fold cross-validation.

Table 1. Results obatined on the test benches. In the testing method column LS stands for Learning Set, whereas TS stands for test set. ± indicates standard deviation. Arrows state whether error is decreasing, increasing or stable in the last 10 runs. In the last column, numbers in square bracket are references to papers publishing the best performances.

Learning Problem	Testing Method	WIL (No boosting)	WIL + Boosting	Best Published Result (Error rate %)
Splice-junctions	2190 LS + 1000 TS	24.1	4.2±0.27 ↓	4.0 [13]
Trains2	5x500 LS + 10000 TS	19.92±1.38	11.6±0.80 ≈	19.97±1.9 [6]
Trains3	5x500LS + 10000 TS	18.17±4.14	24.59±0.65 ↑	12.76±5.4 [6]
Mutagenesis	10 folds cross validation	27.22±10.37	7.7±6.6 ↓	6.4 [27]
Speech	219 LS + 100 TS	36.7±3.35	16±2.19 ↓	18.0 [6]
Artificial Character	6x1000 LS + 10000 TS	30.74±1.29	5.65±0.78 ↓	1.08±1.25 [6]

[1] Available in the ML repository at UCI.

Results on the *splice-junction* dataset are similar to the best ones found by AdaBoost and allowed us to experimentally establish the correctness of the implementation (the WIL algorithm naturally adapts itself to the propositional framework).

By comparing the results reported in columns three and four of Table 1, it is clear that boosting greatly improves performances of the weak learner WIL on 5 out of 6 problems (error is reduced by 324% on average). This confirms Quinlan's statement that 'boosting is advantageous for first-order learning in general' [25]. This result is also an indication to confirm our claim that boosting can be used to find a good combination of simple structured features learned by a FOL weak learner and suggests to deeply investigate this possibility.

However, by comparing the results reported in columns four to those in column five of Table 1, only in one case (*Trains2*) out of 6, boosting WIL outperformed a fully featured (i.e., explicitly configured to solve the problem) first-order learning system. In one case (*Trains3*), boosting degrades performances: this is a somewhat unexpected result given the kind of learning problem. Also Quinlan [25] noted that boosting sometimes degrades performances; in this case, it might be due to small learning sets, but a deeper investigation is required.

It should be pointed out that the best result for the *Mutagenesis* dataset has been obtained by STILL as its best hit with very careful setting of the parameters, while average error rates are around 11%, as also obtained by other systems (see [27] for further details). Also the results on the *ArtificialCharacter* dataset need a further comment. The best results have been obtained by running a powerful learning system such as Smart+ provided with qualitative background knowledge and a more expressive representation language, and refining the learned knowledge with NTR [6] for a total running time of several hours on a Pentium II 400 Mhz processor. Each round of AdaBoost.M1 took about 35 seconds on a Pentium III 600 Mhz processor. This means that there should be room for improvements by performing a larger number of runs.

For what concerns the running times, WIL is as fast as expected: for instance, on the *mutagenesis* dataset it took 10.5 CPU seconds per round, on average, on the *Trains2* dataset it took 4 CPU seconds per round, whereas on the S*peech* dataset it took 6.3 CPU seconds per round.

5 Discussion and Conclusions

From the reported results, boosting seems effective also in solving problems represented in first order logic languages, at the cost of interpretability of the acquired knowledge. The weak learner WIL, being really weak, is greatly improved by boosting, but as Quinlan's results show [25] it is not clear how much boosting a more sophisticated learner can increase performances. A more thorough analysis is required, because it seems that the better the weak learner, the less improvement boosting provides. Furthermore, as pointed out above, performances after 50 rounds are still worse than the best published ones in most cases. As reported in Table 1, on four cases performances constantly decrease in the last 10 rounds, so there is hope to get close to the best results with more rounds.

A second observation concerns the *Trains2* learning problems: this were specifically designed to test the ability of a first order learning system to deal with numerical terms, such as thresholds and parameters, that are difficult to estimate for an expert. In particular, the classification criteria are exclusively based on numerical features. In the experiments performed with AdaBoost.M1, we discretized each numerical attribute in 10 equal-width intervals. It seems that the combination of theories performed by boosting helps overcoming the non-optimal discretization, adding more value to the results in this case. Unfortunately, this is not the case for the *Trains3* problem, where the classification criteria are based on both categiorical and numerical features.

Another issue concerns multi-class learning problems, such as the *Speech* and *ArtificialCharacter* datasets. The classification strategy adopted by WIL is quite simple; by using more sophisticated combination strategies, such as those suggested by Dietterich & Bakiri [12] and Allwein et al. [1] that are orthogonal to the application of boosting, performances might further be improved.

We presented a simple adaptation of AdaBoost to first order logic representation languages with the aim to provide a way to combine structured features easily discovered by a simple weak learner. Future work will be devoted to study the convergence property of the algorithm on long runs (1000 rounds). Moreover, one interesting aspect of AdaBoost is that it calls the weak learner several times on different sampling of the learning instances, focusing the search on the most difficult ones. An investigation of the effects of this strategy on really hard relational problems may reveal a way to overcome the limitations of greedy search heuristics [17].

Acknowledgements

The author is grateful to Attilio Giordana and Roberto Esposito for their helpful comments and discussions during the development and writing process.

References

1. E. L. Allwein, R. E. Schapire and Y. Singer (2000). Reducing multiclass to binary: A unifying approach for margin classifiers. *Journal of Machine Learning Research, 1*, 113-141.
2. C. Anglano, A. Giordana, G. Lo Bello and L. Saitta (1998). An Experimental Evaluation of Coevolutive Concept Learning. *Proc. 15th Int. Conf. on Machine Learning* (Madison, WI, 1998), pp. 19-27.
3. F. Bergadano, A. Giordana and L. Saitta (1988). Learning Concepts in Noisy Environment. *IEEE Transaction on Pattern Analysis and Machine Intelligence, PAMI-10*, 555-578.
4. F. Bergadano, A. Giordana and L. Saitta (1991). *Machine Learning: An Integrated Framework and its Applications*. Hellis Horwood, Chichester, UK.
5. M. Botta and A. Giordana (1993). Smart+: A MultiStrategy Learning Tool. *Proc. of the 13th Int. Joint Conf. on Artificial Intelligence*, (Chambery, France, 1993), pp. 937-943.
6. M. Botta and R. Piola (2000). Refining Numerical Constants in Structured First Order Logic Theories. *Machine Learning Journal, 38*, 109-131.

7. L. Breiman, J. H. Friedman, R. A. Olshen and C. J. Stone (1984). *Classification and Regression Trees*. Wadsworth & Brooks.
8. L. Breiman (1996). Bagging Predictors. *Machine Learning, 24*, 123-140.
9. L. Breiman (1998). Arcing Classifiers. *The Annals of Statistics, 26*, 801-849.
10. L. Dehaspe, H. Toivonen and R. King (1998). Finding frequent substructures in chemical compounds. *Proc. of the 4th Int. Conf. On Knowledge Disciovery and Data Mining* (New York, NY, 1998), pp. 30-66.
11. R. DeMori, A. Giordana, P. Laface and L. Saitta (1984). An Expert System for Mapping Acoustic Cues into Phoenetic Features. *Information Sciences, 33*, 115-155.
12. T. G. Dietterich and G Bakiri (1995). Solving Multiclass Learning Problems via Error-Correcting Output Codes. *Journal of AI Research, 2*, 263-286.
13. Y. Freund and R. E. Schapire (1996). Experiments with a New Boosting Algorithm. *Proc. of the 13th Int. Conf. on Machine Learning*, (Bari, Italy, 1996), pp. 148-156.
14. Y. Freund and R. E. Schapire (1997). A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences, 55*, 119-139.
15. A. Giordana and F. Neri (1996). Search-intensive Concept Induction. *Evolutionary Computation, 3*, 375-416.
16. A. Giordana, F. Neri, L. Saitta and M. Botta (1997). Integrating Multiple Learning Strategies in First Order Logics. *Machine Learning, 27*, 209-240.
17. A. Giordana, L. Saitta and M. Botta (1999). An Experimental Study of Phase Transitions in Matching. *Proc. 16th Int. Joint Conf. on Artificial Intelligence*, (Stockholm, Sweden, 1999), pp. 1198-1203.
18. R. King, A. Srinivasan and M. Stenberg (1995). Relating chemical activity to structure: an examination of ILP successes. *New Generation Computing, 13*.
19. J. W. Lloyd (1987). *Foundations of Logic Programming*. Springer-Verlag.
20. R. S. Michalski (1983). A Theory and Methodology of Inductive Learning. *Artificial Intelligence, 20*, 111-161.
21. S. Muggleton (Ed.). (1992). *Inductive Logic Programming*. Academic Press, London. UK.
22. S. Muggleton (1995). Inverse Entailment and Progol. *New Generation Computing, 13*, 245-286.
23. J. R. Quinlan (1990). Learning Logical Definitions from Relations. *Machine Learning, 5*, 239-266.
24. J. R. Quinlan (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.
25. J. R. Quinlan (1996). Boosting First-Order Learning. *LNAI, 1160*, 143-155.
26. R. E. Schapire, Y. Freund, P. Bartlett and W. S. Lee (1998). Boosting the margin: A new explanation for the effectiveness of voting methods. *The Annals of Statistics, 26*, 1651-1686.
27. M. Sebag and C. Rouveirol (1997). Tractable Induction and Classification in First Order Logic via Stochastic Matching. *Proc. of the 15th Int. Joint Conf. On Artificial Intelligence*, (Nagoya, Japan, 1997), pp. 888-893.
28. G. Towell and J. Shavlik (1994). Knowledge Based Artificial Neural Networks. *Artificial Intelligence, 70*, 119-166.