Multi-class problems and discretization in ICL
Extended abstract

Wim Van Laer [1], Sašo Džeroski [2,3] and Luc De Raedt [1]
(1) Department of Computer Science, Katholieke Universiteit Leuven
Celestijnenlaan 200A, B-3001 Heverlee, Belgium
(2) Department of Intelligent Systems, Jožef Stefan Institute
Jamova 39, 1000 Ljubljana, Slovenia
(3) Institute of Computer Science
Foundation for Research and Technology-Hellas
P.O.Box 1385, 711 10 Heraklion, Crete, Greece
Email:{WimV,LucDR}@cs.kuleuven.ac.be, saso.dzeroski@ijs.si

June 13, 1996

Abstract

Handling multi-class problems and real numbers is important in practical applications of machine learning to KDD problems. While attribute-value learners address these problems as a rule, very few ILP systems do so. The few ILP systems that handle real numbers mostly do so by trying out all real values that are applicable, thus running into efficiency or overfitting problems. This paper discusses some recent extensions of ICL that address these problems.

ICL, which stands for Inductive Constraint Logic, is an ILP system that learns first order logic formulae from positive and negative examples. The main charateristic of ICL is its view on examples. These are seen as interpretations which are true or false for the clausal target theory (in CNF).

We first argue that ICL can be used for learning a theory in a disjunctive normal form (DNF). With this in mind, a possible solution for handling more than two classes is given (based on some ideas from CN2). Finally, we show how to tackle problems with continuous values by adapting discretization techniques from attribute value learners.

1 Introduction

Recently, a novel ILP system called ICL was developed in Leuven (see [8; 1]). The system does not employ the traditional ILP semantics as proposed by Plotkin, in which examples are clauses that are (resp. are not) entailed by the target theory, but rather the view that examples are logical interpretations that are true (resp. false) in the unknown target theory. This view originates from computational learning theory, where it was originally applied to boolean concept-learning but recently upgraded towards first order logic [14; 7].

Because of its origin, the ICL system can be considered an upgrade of the attribute value learning system CN2 ([4; 3]). However, whereas CN2 learns boolean concepts in DNF form, ICL learns first order clausal theories in CNF form. Nevertheless, many of the other aspects of CN2 are inherited, including the efficiency.

In this paper, we import further features of attribute value learning into the ILP system ICL. First, it is shown that ICL can also learn DNF concepts. Secondly, CN2's way of handling multi-class (i.e. more than 2 classes) problems is also mapped to ICL, and this both for DNF as for CNF. Thirdly, and most importantly, ICL (as many other ILP systems) has a problem handling real numbers. In attribute value learning, discretization has recently received a lot of attention (cf. [2; 9]) and proven its use. The advantages of discretization w.r.t. classical attribute value techniques (such as TDIDT) for handling numbers are two-fold: they are more efficient, and sometimes also yield more accurate results. Here, it is shown how Fayyad and Irani's discretization technique can be modified for use in ILP and more specifically in ICL.

Though we do not report on any experiments in this short note, it is our intention to report on such experiments at the workshop. Currently, we are carrying out experiments on the mutagenesis data.

2 Inductive Constraint Logic: overview

An overview of ICL can be found in [8] and [1]. Here, we shortly describe the framework of ICL and expand on some practical aspects.

2.1 Framework

We will use some notions of first order logic and model theory (for an introduction, see [12; 11]). Definitions of the concepts used here can be found in [8].

ICL is a classification system that learns a clausal theory which discriminates as good as possible between two classes of examples (let's say positives and negatives). Examples are seen as interpretations I of the target theory T. A positive example (interpretation) P is a model of the target theory T (i.e. P is a model for all clauses C in T) and a negative example N is not a model of T. One also says that a clause/theory is true/false in an example/interpretation, or even that an example is true for a clause, or covered by a clause. This view that examples are interpretations and that the aim is to discriminate between two classes of examples, is similar to classical learning from positive and negative examples and originates from [7].

Note that all variables in the clauses are universally quantified. So an interpretation I is a model for a clause iff for all grounding substitutions θ of c : $\mathit{body}(c)\theta \subset I \rightarrow \mathit{head}(c)\theta \cap I \neq \emptyset$. This is easily checked by a theorem prover like Prolog.

In ICL, a definite clause background theory can be used as follows. Rather than starting from complete interpretations of the target theory, examples are a kind of partial interpretations I (i.e. sets of facts) and are completed by taking the minimal Herbrand model $M(B \cup I)$ of background theory B plus the partial interpretation

I. Note that the completion need not be computed explicitly, but can be left to the theorem-prover[1].

Formally, the setting for ICL thus becomes:

- **Given**
  - P is a set of interpretations such that for all $p \in P$, $M(B \cup p)$ is a true interpretation of the unknown target theory (positive examples);
  - N is a set of interpretations such that for all $n \in N$, $M(B \cup n)$ is a false interpretation of the unknown target theory (negative examples);
  - a definite clause background theory B;
  - a language L_H which defines the set of a priori acceptable clauses in hypothesis;
- **Find**: a clausal theory $H \subset L_H$ such that
  - for all $p \in P$, $M(B \cup p)$ is a true interpretation of H (Completeness);
  - for all $n \in N$, $M(B \cup n)$ is a false interpretation of H (Consistency).

2.2 Practice

An overview of the ICL algorithm can be found in [8]. In short, ICL uses a covering approach and beam search to find solutions in CNF form. In this way, ICL is a kind of dual first order version of the DNF-learner CN2.

To specify the hypothesis language, ICL uses the same declaritive bias as CLAUDIEN, i.e. DLAB (declaritive language bias). DLAB is a formalism to specify an intensional syntactic definition of the language L_H (a hypothesis is a conjunction of clauses, and DLAB specifies the allowed syntax for the clauses). This is automatically translated in a refinement operator (under θ-subsumption) for the specified language which is used by ICL to traverse the search space. For a complete overview of DLAB, we refer to the CLAUDIEN paper [5].
A small example:

```
{false
 <--
 0-len:[len-len:[lumo(Lumo), lt(Lumo, 1-1:[-1, -2])],
        len-len:[atom(A1, Elem1, Type1, Charge1),
                 lt(Charge1, 1-1:[-0.2, -0.1, 0, 0.1])
                ]
       ]
}
```

Min-Max:List means that at least Min and at most Max literals of List are allowed (`len` is the length of `List`). Note that lt(Lumo, 1-1:[-1,-2]) is a short notation for 1-1:[lt(Lumo, -1), lt(Lumo, -2)].

Currently, there is a working implementation of ICL in ProLog by BIM. Several heuristics (based on CN2) are incorporated to handle noise. This means that the learned theory doesn't need to be strictly complete and consistent with the training set: some clauses might not cover all positives, and not all negatives need to be excluded by some clause in the theory. Several parameters can be set to tune the heuristics.

[1] Using Prolog, one can assert background theory and interpretation into the knowledge base, and run the query $? - body(c)$, *not* $head(c)$, in order to test whether a clause c makes an interpretation true or not.

3 Disjunctive Normal Form

ICL learns a hypothesis in conjunctive normal form (CNF). This is mainly inherited from its older twin system CLAUDIEN. But as argued in [6] (cf. [13]), we can learn DNF with a CNF learner.

Classification systems such as CN2 learn a hypothesis that discriminates between examples that belong to a given class, and those that do not. The concept description is then usually of the form

$$c \leftarrow l_{1,1} \wedge \cdots \wedge l_{1,n_1}$$

$$c \leftarrow l_{2,1} \wedge \cdots \wedge l_{2,n_2}$$

$$\vdots$$

$$c \leftarrow l_{k,1} \wedge \cdots \wedge l_{k,n_2}$$

so class c is described by the following DNF formula:

$$(l_{1,1} \wedge \ldots \wedge l_{1,n_1}) \vee \cdots \vee (l_{k,1} \wedge \ldots \wedge l_{k,n_1})$$

Notice that in this DNF, the variables (if any) are existentially quantified. Each conjunct can be seen as a sufficient condition for class.

ICL learns a class description (theory) in conjunctive normal form (CNF). A clausal theory T_c for a class c is of the following form:

$$l_{1,1} \vee \ldots \vee l_{1,n_1} \leftarrow l_{1,n_1+1} \wedge \cdots \wedge l_{1,m_1}$$

$$\ldots$$

$$l_{k,1} \vee \ldots \vee l_{k,n_k} \leftarrow l_{k,n_k+1} \wedge \cdots \wedge l_{k,m_k}$$

This corresponds to the following conjunctive normal form (CNF):

$$l_{1,1} \vee \ldots \vee l_{1,n_1} \vee \neg l_{1,n_1+1} \vee \cdots \vee \neg l_{1,m_1}$$

$$\wedge$$

$$\ldots$$

$$\wedge$$

$$l_{k,1} \vee \cdots \vee l_{k,n_k} \vee \neg l_{k,n_k+1} \vee \cdots \vee \neg l_{k,m_k}$$

Notice that the variables appearing in this CNF are universally quantified. Each disjunct corresponds to a necessary condition for class c.

There exists a simple relationship between CNF and DNF descriptions. Suppose we learn a class $\oplus$. For each positive example, the above CNF formula must be true. This means that whenever the CNF is false in an example, the example is a negative one. The negation of the CNF, is the following DNF:

$$\neg l_{1,1} \wedge \ldots \wedge \neg l_{1,n_1} \wedge l_{1,n_1+1} \wedge \ldots \wedge l_{1,m_1}$$

$$\vee$$

$$\ldots$$

$$\vee$$

$$\neg l_{k,1} \wedge \ldots \wedge \neg l_{k,n_k} \wedge l_{k,n_k+1} \wedge \ldots \wedge l_{k,m_k}$$

Notice that because of the negation, all variables are now existentially quantified. Whenever this DNF is true, the above CNF is false, so the example is negative. The DNF therefore expresses sufficient conditions for the negative class.

We can conclude that for each CNF description for a class, there exists a corresponding DNF description of the complementary class, which has the same structure, but with $\vee/\wedge$, $\exists/\forall$ and positive/negative literals switched.

This works fine if only 2 classes appear in the training set: the CNF theory which ICL learns for a class, can easily be transformed to the corresponding DNF description which can be used as a DNF theory for the complementary class. If however, more than two classes exist, the complementary class contains all the other classes (thus not class).

4 Multi-class problems

Up to now, we have applied ICL mainly on problems with 2 classes. So given a set of positive and negative examples for a class c, ICL learns a theory $T^c = (C_1 \wedge \ldots \wedge C_n)$ that differentiates as good as possible between the examples of that class (positives) and the others (negatives). With this theory T^c, an unseen example is classified as class c if the example is a model for the theory T^c, i.e. all clauses C_i of the theory are true in the example. Otherwise the example is assumed not to be of class c.

In many applications with two classes, this is sufficient. But if we have m classes, it's not sufficient to learn just one theory (one would only be able to discriminate between one class and all the others). We should learn m theories, one for each class. The question then is, how to apply this set of theories to an unseen example, in order to predict its class? (Note that this is also useful with two classes: we learn two theories, one for each class, which might be very helpful in domains with a lot of noise).

Therefore, we looked at CN2 with unordered rules (see [3]). Here, one learns a theory (in DNF form, a set of rules) for each class in turn. For a given example, all rules are tested. If only rules for one class succeed, this class is predicted. If no rule applies, the default class (majority class in training set) is predicted. Otherwise, one uses a probabilistic method to resolve the clash. One chooses the most probable class given the rules that applied. To this end, one stores with each rule the distribution of covered examples among the classes. More details can be found in the CN2 paper [3].

In the previous section, we showed that we can learn DNF formulae with ICL: learn a clausal theory for the complementary classes and transform the CNF to a DNF. Thus for our multi-class problem, we can learn a theory for each complementary class, and then use the decision procedure of CN2.
For example, if we have three classes c_1, c_2 and c_3, we need to learn theories for class (c_1 or c_2), for class (c_1 or c_3), and for class (c_2 or c_3).

If we apply this on an application with two classes, we can use the dual of the above procedure. In this case, each class is the complement of the other class. With each clause (complement of the rules), we keep the distribution of uncovered (instead of covered) examples among the classes. To solve a clash (in which case both clausal theories contain a clause that fails), one chooses the class which is least rejected.
At present, only the two-class dual classification procedure is implemented. An extension to more than two classes is under way.

5 Discretization

The motivation for discretizing numeric data is two fold and based on the findings in attribute value learning. On the one hand, there is an efficiency concern. On the other hand, by discretising the data, one may sometimes obtain higher accuracy rates.

Current procedures in ILP to handle numbers, such as those by FOIL, and those employed using the older versions of CLAUDIEN, are quite expensive. The reason is that for each candidate clause, all values for a given numeric variable have to be generated and considered in tests. In large databases, the number of such values can be huge, resulting in a high branching factor in the search. Furthermore, in these approaches, discretization is done at runtime, i.e. it is repeated for every candidate clause. If one clause is a refinement of another one, a lot of redundant work may be done.

What we propose is to generate beforehand (i.e. before the ILP systems starts) some interesting tresholds to test upon. This makes that tresholds are computed only once (instead of once for each candidate clause considered), and secondly, that the number of interesting tresholds (to be considered when refining clauses) is kept to a minimum, yielding a smaller branching factor. This has also yielded positive results in attribute value learning, cf. [2].

Though we present the procedure as applied in the ICL system, it also generalizes to other ILP sytems (working under the standard semantics). The discretization procedure is tied with the DLAB parameter of ICL, which defines the syntax of the clauses that may be part of a hypothesis. One such DLAB template is given in section 2.2. In this template, the user has specified some possible tresholds for ICL. But where do they come from? Up to now, the user had to specify them. In most applications, this is not straightforward. To this aim, we extended ICL with the possibility to produce itself possible thresholds (before the learning procedure is called).

When looking at the DLAB-templates (or at actual clauses in a hypothesis) it is often possible to identify a number of conceptually meanigful subclauses. E.g. in the above DLAB, $atom(A1, Elem1, Type1, Charge1)$.

We will call such subclauses *queries*. One could consider each such query that involves a numeric argument as a kind of numeric attribute. E.g. $Charge1$ in the example. There is one important difference with regard to attribute value learning, and it is that one example may have multiple values for such a numeric query or attribute.

In our approach to discretisation, the user has to identify the relevant queries. The resulting numeric attributes are then discretised using a simple modification of Fayyad and Irani's method, and the result is fed back into the DLAB template. At this moment, the syntax is as follows:

```
dlab_template(
'false
 <--
 0-len:[len-len:[lumo(Lumo), lt(Lumo, c_lumo)],
        len-len:[atom(A1, Elem1, Type1, Charge1),
                 lt(Charge1, c_charge)
                ]
       ]
').
dlab_query(c_lumo, 1-1, discretize(lumo(Lumo), Lumo)).
dlab_query(c_charge, 1-1,
           discretize(atom(A1, Elem1, Type1, Charge1), Charge1)).
```

The details of the Fayyad and Irani's method can be found in [10] and [9]. The only differences we apply are :

- due to the fact that one example may have multiple values for a numeric attribute, we use sum of weights instead of numbers of examples in the appropriate places of Fayyad and Irani's formulae (i.e. when we count the real values that are less than a threshold, we sum their weights - in the AV case all values have weight 1 as each example has only one value for one attribute). The sum of the weights of all values for one numeric attribute or query in one example always equals one.

- the stopping criterion of Fayyad and Irani's discretization method is very strict, in the sense that the method generates very few subintervals. We have modified the stopping criterion (discretization stops if `LeftEq < RightEq`) in the following way: we always generate at least one threshold. At this point, note the values LeftEq0 and RightE0 in the stopping criterion. If LeftEq0 < RightEq0, the threshod would have not been accepted by the Fayyad and Irani criterion. In this case we soften the criterion to LeftEq < (LeftEq0/RightEq0)*RightEq for the following recursive calls. Otherwise, we just use the original criterion.

Finally we should note that when working with a training and a test set (i.e. for cross validation), discretization should only use the training set. It would be unfair to use the test set as well in the discretization process.

6 Conclusion

We have indicated how several well-established attribute value learning techniques can be upgraded for use in the ILP system ICL. This includes the relation between CNF and DNF, discretization and multi-class problems

We don't have concrete results, but preliminary experiments indicate that ICL with discretization performs as well as ICL without discretization and is more efficient than the latter. More detailed experimental results are planned for the full paper.

Acknowledgements

Wim Van Laer and Luc De Raedt are supported by the Belgian National Fund for Scientific Research. Sašo Džeroski is supported by the Slovenian Ministry of Science and Technology and ERCIM - The European Research Consortium for Informatics and Mathematics. This research is also part of the ESPRIT project no. 20237 on Inductive Logic Programming II.

References

[1] H. Blockeel, W. Van Laer, and L. De Raedt. Inductive constraint logic and the mutagenesis problem. In *Proceedings of the 5th Belgian-Dutch Conference on Machine Learning (BENELEARN '95)*, 1995.

[2] J. Catlett. On changing continuous attributes into ordered discrete attributes. In Yves Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 164–178. Springer-Verlag, 1991.

[3] P. Clark and R. Boswell. Rule induction with cn2: Some recent improvements. In Yves Kodratoff, editor, *Proceedings of the 5th European Working Session on Learning*, volume 482 of *Lecture Notes in Artificial Intelligence*, pages 151–163. Springer-Verlag, 1991.

[4] P. Clark and T. Niblett. The CN2 algorithm. *Machine Learning*, 3(4):261–284, 1989.

[5] L. De Raedt and L. Dehaspe. Clausal discovery. Forthcoming, 1995.

[6] L. De Raedt, L. Dehaspe, W Van Laer, H. Blockeel, and M. Bruynooghe. On the duality of cnf and dnf, or how to learn cnf using a dnf learner. Unpublished, 1995.

[7] L. De Raedt and S. Džeroski. First order jk-clausal theories are PAC-learnable. *Artificial Intelligence*, 70:375–392, 1994.

[8] L. De Raedt and W. Van Laer. Inductive constraint logic. In *Proceedings of the 5th Workshop on Algorithmic Learning Theory*, Lecture Notes in Artificial Intelligence. Springer-Verlag, 1995.

[9] J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In A. Prieditis and S. Russell, editors, *Proc. Twelfth International Conference on Machine Learning.* Morgan Kaufmann, 1995.

[10] U.M. Fayyad and K.B. Irani. Multi-interval discretization of continuous-valued attributes for classification learning. In *Proceedings of the 13th International Joint Conference on Artificial Intelligence*, pages 1022–1027, San Mateo, CA, 1993. Morgan Kaufmann.

[11] M. Genesereth and N. Nilsson. *Logical foundations of artificial intelligence.* Morgan Kaufmann, 1987.

[12] J.W. Lloyd. *Foundations of logic programming.* Springer-Verlag, 2nd edition, 1987.

[13] R.J. Mooney. Encouraging experimental results on learning cnf. *Machine Learning*, 19:79–92, 1995.

[14] L. Valiant. A theory of the learnable. *Communications of the ACM*, 27:1134–1142, 1984.