A Toolbox for Learning from Relational Data with Propositional and Multi-Instance Learners

Peter Reutemann[1,2], Bernhard Pfahringer[2], and Eibe Frank[2]

[1] Department of Computer Science, University of Freiburg, Freiburg, Germany
[2] Department of Computer Science, University of Waikato, Hamilton, New Zealand

Abstract. Most databases employ the relational model for data storage. To use this data in a propositional learner, a propositionalization step has to take place. Similarly, the data has to be transformed to be amenable to a multi-instance learner. The Proper Toolbox contains an extended version of RELAGGS, the Multi-Instance Learning Kit MILK, and can also combine the multi-instance data with aggregated data from RELAGGS. RELAGGS was extended to handle arbitrarily nested relations and to work with both primary keys and indices. For MILK the relational model is flattened into a single table and this data is fed into a multi-instance learner. REMILK finally combines the aggregated data produced by RELAGGS and the multi-instance data, flattened for MILK, into a single table that is once again the input for a multi-instance learner. Several well-known datasets are used for experiments which highlight the strengths and weaknesses of the different approaches.

1 Introduction

This paper describes the Proper Toolbox [4], a general framework for database-oriented propositionalization algorithms that can also create multi-instance data from relational data.[1] The paper is organized as follows: first we discuss the RELAGGS propositionalization system, which is a major component of Proper, and then the other components of Proper. After that we report on results obtained from a suite of experiments that apply Proper to some relational benchmark datasets. The final section summarizes the paper.

2 The Proper Toolbox

In this section we discuss the various components of Proper, starting with its most important building block, RELAGGS.

2.1 RELAGGS: The propositionalization engine

RELAGGS is a database-oriented approach based on aggregations that are performed on the tables adjacent to the table that contains the target attribute. For

[1] Proper is freely available from http://www.cs.waikato.ac.nz/ml/proper/.

each row in the target table the following SQL group functions are executed for all numeric columns in the adjacent tables: average, minimum, maximum, sum. Additionally standard deviation, quartile, and range are computed. For nominal columns the number of occurrences of each nominal value is counted and represented as a new attribute. RELAGGS also computes aggregations based on pairs of attributes with one nominal attribute. This nominal attribute serves as an additional `GROUP BY` condition for the aggregation process [2]. RELAGGS uses the names of primary keys to determine the relationships between the various tables in the database. Proper uses the version of RELAGGS from [3].

We modified RELAGGS to relax some of the constraints it imposes on its input. First, RELAGGS expects an integer as the primary key of a table. In some domains the primary key of the table is an alpha-numeric string. In such cases Proper generates an additional table containing the original identifiers and newly generated integer keys, which replace the original alpha-numeric keys in all other tables. Second, determining the relationship between two tables solely using primary keys proved to be problematic when the relationship between different tables is based on compound IDs. Compounds may have more than one instance and this clearly rules out the compound ID as a primary key. Therefore, instead of primary keys, indices are used to identify relationships between tables. Third, the use of indices instead of primary keys unfortunately has further consequences: joins may work differently, and care has to be taken to avoid loss of information. When importing datasets into Proper, either an additional unique index (based on table name and row-ID) is generated automatically, or some key can be specified to be the unique index. Fourth, due to the possibility of importing Prolog data, and the closed-world assumption used in Prolog-based representations, tables do not necessarily include explicit information about the absence of feature values. Hence, to prevent against potential loss of instances in joins, Proper uses the `LEFT OUTER JOIN` instead of the `NATURAL JOIN` (which is used in the original version of RELAGGS). Fifth, since the above version of RELAGGS only aggregates tables adjacent to the target table, Proper pre-flattens arbitrarily deep nested structures into temporary tables.

2.2 The other components of Proper

In the following we describe the Proper framework, which is depicted in Figure 1. We will explain the individual steps with suitable examples. The first step is the import of data from a file or database.

Import Currently Proper supports two different formats for importing data into databases: Prolog (only extensional knowledge, but including ground facts with functors) and CSV-files (with or without identifiers for the columns). For both formats the types of the columns in the table are determined automatically. Supported types are `Integer`, `Double`, `Date` and `String`. CSV import is pretty straightforward, since the data is already in a column-like representation. If the file contains a header row with the names of the columns, then these are used.

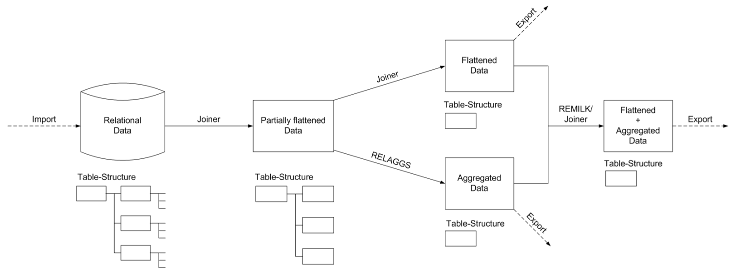

Fig. 1. Proper's program structure.

Otherwise a name is constructed automatically out of both the file name and the position of each column.

```
train(east,
      [c(1,rectangle,short,not_double,none,2,l(circle,1)),
       c(2,rectangle,long,not_double,none,3,l(hexagon,1)),
       c(3,rectangle,short,not_double,peaked,2,l(triangle,1)),
       c(4,rectangle,long,not_double,none,2,l(rectangle,3))]).
```

Fig. 2. East-West-Challenge Example.

Prolog (or closely related formats like Progol or Golem) can be imported into databases in such a way that each functor represents a separate table. Consider the example of the East-West-Challenge in Figure 2. Since this dataset is a relational Prolog database we do not need to specify the relations between the functors explicitly. Otherwise we would have to do this by indicating which argument index functions as a key, e.g. in the well-known Alzheimer datasets the argument that contains the compound ID.

The structure of this example can easily be translated into the table structure shown in Figure 3. The `train_list` table would not actually be necessary to represent the `1..n` relationship between `train` and `car`, but this is Proper's generic approach of storing each functor in its own table. In the case of uniform lists (i.e. all lists are of same length) Proper can also turn a list directly into a table with an equal number of columns. This built-in optimization gets rid of one table thus reducing the complexity of the generated database.

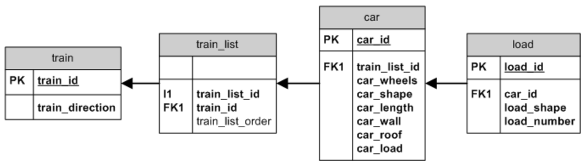

Fig. 3. East-West-Challenge as a relational database.

Proper also includes a few more advanced features for importing Prolog. First, if the relations cannot be determined from the Prolog database itself, it is possible to define them explicitly via *foreign key relations*. Then during import, functors will be rearranged to fit the proposed relational model. Second, for problems that are represented as flat, ground Prolog facts one also has to specify which columns are to be used for joins, as this is not necessarily obvious from the plain data. Third, depending on the representation of the data there might be more than one argument containing a key, e.g. in the Alzheimer datasets, where there are functors that define a relation between the two arguments: `less_toxic(a1, b1)`. For a symmetric relation `equally_toxic`, the instance `equally_toxic(a1, b1)` is split into two instances `equally_toxic(a1, 1)` and `equally_toxic(b1, 1)`, where the second argument is the so called `split_id` that links both instances together. To properly represent asymmetric relationships, new distinct functors have to be defined for each argument position and `less_toxic(a1, b1).` becomes `less_toxic(less_toxic0(a1), less_toxic1(b1)).`

Joiner The central processing algorithm in Proper is the Joiner. As can be seen in Figure 1 the Joiner performs the flattening of arbitrarily nested structure of relations into appropriate structures for RELAGGS (maximum depth of 1), MILK (one flat table of depth 0, suitable for the multi-instance learning kit MILK) and REMILK (also one flat table). In multi-instance learning each example consists of several instances, and is also called a *bag* of instances. The data for REMILK (*RE*lational aggregation enrichment for MILK) is produced by joining the tables that have been generated for RELAGGS and MILK.

The Joiner works on tree structures. To build up such a tree structure the Joiner can either use user-specified relationships between tables or discover such relationships automatically. A GUI frontend supports specifying these tree structures. Auto-discovery of relationships determines the possible relation between tables based on column names. In order to keep the IO operations to a minimum, the joins are ordered such that smaller tables are joined first.

Left outer joins are performed in order not to lose any instances of the target table. Since classifiers normally handle missing values, the created NULL values can be interpreted as missing values. The columns over which the join is performed (i.e. the columns that are tested in the `WHERE` clause of the generated join-query) are determined by the intersection of the indices of the first table with all the columns of the second one. The user can specify replacement values for automatically generated NULLs on a column-basis (e.g. replacing them by "0") if they should not be treated as missing values. Such columns are updated after a join-operation.

In cases where there are additional duplicate columns beside the join columns, the duplicate columns' names are prefixed with `mX_`, where `X` is a unique number used for all columns in the current join. Without that precaution potentially essential information could be lost. A common case for this situation to arise is the handling of asymmetric relationships, where the (initially identically named) properties of both arguments have to be included in the final table.

Export The is the last step before the classifiers can be built and evaluated. Tables generated by Proper are transformed into appropriate ARFF files for the WEKA workbench. If certain columns contain implicit knowledge like identifiers of tables (and their aggregates), it is possible to exclude them from export. In the case of multi-instance data, a bag identifier can be specified explicitly or one can be determined automatically. NULL values that were present in the data or were introduced during left outer joins are exported as missing values. If the ARFF file is too large it is possible to export only a stratified sample by specifying a sampling percentage. Finally, WEKA filters can be applied to the data before it is written to an ARFF file, e.g. nominal attributes can be turned into binary indicator attributes.

3 Experiments

We used 18 datasets in our experiments with Proper.[2] Table 1 shows the results. Note that the `alzheimer_*`, `dd_*`, and `proteins` datasets only have one instance per bag in the MILK and REMILK versions, so they are not "true" multi-instance datasets.

Dataset	RELAGGS	MILK	REMILK
alzheimer_amine_uptake	87.59 ± 4.31	73.35 ± 5.42	87.26 ± 4.52
alzheimer_choline	89.18 ± 2.73	79.17 ± 3.68	89.45 ± 2.68
alzheimer_scopolamine	87.84 ± 4.23	74.16 ± 5.30	87.78 ± 4.33
alzheimer_toxic	92.93 ± 2.74	88.77 ± 3.48	92.39 ± 3.00
dd_pyrimidines	92.47 ± 2.02	92.46 ± 1.94	92.46 ± 1.94
dd_triazines	74.76 ± 0.85	74.78 ± 0.85	74.78 ± 0.85
eastwest	80.00 ± 41.03	55.00 ± 51.04	75.00 ± 44.42
genes_growth	31.70 ± 1.60	34.00 ± 1.01	33.36 ± 1.25
genes_growth_bin	84.14 ± 0.42	84.33 ± 0.22	84.34 ± 0.40
genes_nucleus	76.06 ± 2.15	57.22 ± 2.19	62.55 ± 2.03
genes_nucleus_bin	87.28 ± 1.39	74.13 ± 1.67	78.49 ± 1.63
musk1_rel	80.13 ± 15.21	81.64 ± 14.68	79.50 ± 14.58
musk2_rel	72.83 ± 13.44	76.82 ± 12.61	74.40 ± 13.73
mutagenesis3_atoms	79.58 ± 8.90	81.86 ± 8.23	80.15 ± 9.24
mutagenesis3_bonds	85.38 ± 7.85	83.62 ± 7.87	86.68 ± 7.46
mutagenesis3_chains	85.31 ± 7.83	84.54 ± 7.00	84.85 ± 8.32
proteins	59.52 ± 4.08	59.12 ± 5.08	59.92 ± 3.38
suramin	45.45 ± 52.22	63.63 ± 50.45	45.45 ± 52.22

Table 1. Accuracy and standard deviation.

For MILK and REMILK we used the multi-instance learner MIWrapper[3], which can be wrapped around any standard propositional learner as described in [1]. The MI-Wrapper approach assigns each instance of the n instances in a bag a weight of $1/n$. Therefore all the bags have the same total weight regardless of the number of instances they contain. For predicting a bag label a class probability is obtained from the propositional model for every instance of the bag. These probabilities are simply averaged to determine the resulting class label for the bag.

This approach enjoys an advantage over aggregation as performed by RELAGGS if the data looks like that in Figure 4, i.e. if interactions between attributes are significant for prediction. Here the aggregates generated by RELAGGS are identical for both classes, making discrimination impossible, but the MIWrapper algorithm would be able to create a useful classifier, for example, using a propositional decision tree learner.

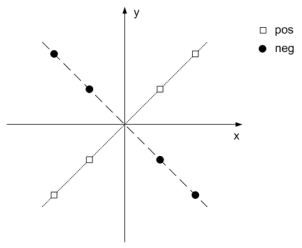

Fig. 4. Artificial dataset.

We used unpruned decisions trees in our experiments with RELAGGS and MILK/REMILK. Only for the `genes_*` datasets we used boosted decision stumps instead because the trees became too large. Both learning schemes are insensitive to the relative scale of the instances' weights and that is why we used them. In all experiments we used 10 runs of stratified 10-fold cross-validation, only in case of `suramin` and `eastwest` we used Leave-One-Out. This was done because of the very small size of these datasets. Also, to imitate RELAGGS's behaviour, we binarized all nominal attributes before passing them to the MIWrapper and replaced missing values in the resulting attributes by 0.

[2] All the data used in our experiments is available from the Proper web page http://www.cs.waikato.ac.nz/ml/proper.

[3] The MIWrapper is part of MILK, the Multi-Instance Learning Kit, which is freely available from http://www.cs.waikato.ac.nz/ml/milk/.

When interpreting the results shown in Table 1, we see that RELAGGS and REMILK perform similarly (the exception being the `gene_nucleus_*` data, where REMILK performs worse—possibly because the RELAGGS attributes follow after the attributes from the multi-instance data in the REMILK version of the data, and the decision tree learner is thus biased towards the latter set of attributes). The results indicate that in practice one might as well run the faster and less memory-demanding RELAGGS approach instead of the combination approach REMILK.

MILK is performing as well as the other approaches on about two thirds of all datasets, but it does worse on the remaining datasets. Currently we do not have a good explanation for this difference, as we were actually expecting the multi-instance approach to enjoy an advantage. But this theoretical advantage (see the example discussed above) does not seem to be relevant in practice. Note that the difference on the single-instance `alzheimer_*` datasets is solely due to the fact that the RELAGGS approach enables the propositional learner to treat `NULL` effectively as a separate value rather than a missing value (because some of the aggregate functions used by RELAGGS return zero if there are no applicable records). This different treatment of missing values may be partially responsible for the differences observed in other cases as well. There are no `NULL` values in the `musk` datasets and here MILK actually has a slight edge.

4 Conclusions and Future Work

This paper presents an attempt to develop a practical database-oriented framework for different propositionalization algorithms. The flexible design allows for the future integration of other propositionalization algorithms in addition to RELAGGS. Proper makes standard propositional and multi-instance learning algorithms available for relational learning. A preliminary empirical investigation has shown the feasibility of this approach.

References

1. E. Frank and X. Xu. *Applying Propositional Learning Algorithms to Multi-instance data.* Working Paper 06/03, Computer Science, University of Waikato, 2003.
2. M.-A. Krogel and S. Wrobel. *Facets of Aggregation Approaches to Propositionalization.* In: T. Horváth and A. Yamamoto (Eds.) Proceedings of the Work-in-Progress Track at the 13th International Conference on Inductive Logic Programming, 2003.
3. M.-A. Krogel, S. Rawles, F. Železný, P. A. Flach, N. Lavrač, and S. Wrobel. *Comparative Evaluation of Approaches to Propositionalization.* In: T. Horváth and A. Yamamoto (Eds.) Proceedings of the 13th International Conference on Inductive Logic Programming. LNCS 2835, Springer-Verlag, 2003.
4. P. Reutemann. *Development of a Propositionalization Toolbox.* MSc Thesis, Computer Science, University of Freiburg, 2004.