Stochastic propositionalization of relational data using aggregates

Valentin Gjorgjioski and Sašo Dzeroski

Jožef Stefan Institute

Abstract. The fact that data is already stored in relational databases causes many problems in the practice of data mining. To deal with this problem, one either constructs a single table by hand, or one uses a Multi-Relational Data Mining algorithm. In this paper, we propose an approach in which the single table is constructed automatically using aggregate functions, which repeatedly summarize information from different tables over associations in the relational database. Following the construction of the single table, we recommend applying traditional data mining algorithms. Next to an in-depth discussion of our approach, the paper discusses testing of our algorithm on two well-known data sets.

1 Introduction

An important practical problem in data mining is that we often want to find models and patterns over data that resides in multiple tables. This is solved by either constructing a single table by hand (deriving attributes from the other tables) or by using a Multi-Relational Data Mining or Inductive Logic Programming(ILP) approach. In this paper we discuss aggregation approach, automatic construction of the single mining table using aggregates [3]. The difficult case in constructing a single table is when there are one-to-many relationships between tables. The traditional way to summarize such relationships in Statistics and On Line Analytical Processing (OLAP) is through aggregates, such as count, sum, min, max, and avg. We provide an open system where aggregates can be easily added. Also, note that aggregates can be applied recursively over a collection of relationships. Because of that, the aggregation approach easily leads to very huge number of attributes, and a lot of time is needed such propositionalization algorithm to finish. We use stochastic optimization to solve this problem and to tune the parameters to make trade-off between time, space and quality of the output.

The idea of propositionalization (the construction of one table) is not new [5]. Several relatively successful algorithms have been proposed in the context of ILP [1,4,7]. A common aspect of these algorithms is that the derived table consists solely of binary features, each corresponding to a (promising) clause discovered by an ILP-algorithm. Especially for numerical attributes, our approach leads to a markedly different search space.

The rest of paper is organized as follows. First we discuss propositionalization and aggregates in more detail. Further on, we introduce stochastic optimization, used in our algorithm, and in next section we are describing our algorithm. At the end we discuss results of the experiments and our approach.

2 Propositionalization

In this section we describe the basic concepts involved in propositionalization, and provide some definitions. In this paper, we define propositionalization as the process of transforming a multi-relational dataset, containing structured examples, into a propositional dataset with derived attribute-value features, describing the structural properties of the examples. The process can thus be thought of as summarizing data stored in multiple tables in a single table (the target table) containing one record per example. The aim of this process, of course, is to preprocess multi-relational data for subsequent analysis by attribute-value learners. We use this definition in the broadest sense. We don't make any assumptions about the datatype of the derived attribute (binary, nominal, numeric,etc.).

With a growing availability of algorithms from the fields of ILP and Multi-Relational Data Mining (MRDM), one might wonder why such a cumbersome preprocessing step is desirable in the first place, instead of applying one of these algorithms to the multi-relational data directly. The following is a (possibly incomplete) list of reasons:

– Pragmatic choice for specific propositional techniques. People may wish to apply their favorite attribute-value learner, or only have access to commercial Data Mining tools.
– Greater speed of propositional algorithms. This advantage of course only holds if the preceding work for propositionalization was limited, or performed only once and then reused during multiple attribute-value learning sessions.
– Advantages related to multiple consecutive learning steps. Because we are applying two learning steps, we are effectively combining two search strategies. The first step essentially transforms a multi-relational search space into a propositional one. The second step then uses these complex patterns to search deeper than either step could achieve when applied in isolation.

3 Aggregates

In the previous section we observed that an essential element of propositionalization is the ability to summarize the information distributed over several tables in the target table. We require functions that can reduce pieces of substructure to a single value, which describes some aspects of this substructure. Such functions are called aggregates. Having a set of well-chosen aggregates will allow us to describe the essence of the structural information over a wide variety of structures. Some of aggregates that are used in propositionalization are:

– count, count with condition,
– count distinct,

- min, max, sum, avg,
- exists, exists with condition,
- select record (oldest son, first contract),
- predominant value.

4 Summarization

We will be viewing the propositionalization process as a series of steps in which information in one table is projected onto records in another table successively. Each association in the data model gives rise to one such step. The specifics of such a step, which we will refer to as summarization, are the subject of this section. Let us consider two tables P and Q, neither of which needs to be the target table, that are joined by an association A. By summarizing over A, information can be added to P about the structural properties of A, as well as the data within Q. To summarize Q, a set of aggregates are needed. Our choice of aggregates depends on the multiplicity of A. In particular if we summarize Q over A only the multiplicity on the side of Q is relevant. This is because an association in general describes two relationships between the records in both tables, one for each direction. The basic types of connectivity for relations are:

- **one(or many)-to-one** For every record in P there is but a single record in Q. This is basically a look-up over a foreign key relation and no aggregates are required. A simple join will add all non-key attributes of Q to P. Special case is if a look-up fails because a record in P may not have a corresponding record in Q. In such case an outer join is necessary, which fills in NULL values for missing records.
- **one-to-many** For every record in P, there is at least one record in Q. Aggregates are required in order to capture the information in the group of records belonging to a single record in P. Special case is if some corresponding group in Q is an empty set. In that case, values of certain aggregates may be undefined due to empty groups. Special care needs to be taken (by RDBMS or by the user) to deal with the resulting NULL values.

5 Stochastic optimization

Because of time and space complexity of the propositionalization step, we introduce stochastic optimization in our propositionalization system PRORED (PROpositionalization of RElational Data). We implement stochastic optimization at two levels. First, we use heuristic (1) for choosing the attributes from a table, then we apply the same heuristic for choosing aggregate functions. We use the same heuristic formula given bellow, but parameter *depthImportance* is set by the user, separately for aggregate functions and for attributes.

$$h = (10 - log((depthImportance \cdot currentDepth + 1))) \cdot 10 \qquad (1)$$

In this formula, *depthImportance* is a parameter which is used to determine how much the depth will influence when we choose columns or aggregates. The user is required to enter two parameter values into the optimization system. The first one is how much the depth will influence on the selection of attributes (columns) from tables, and the other one is to determine the influence of the depth on the selection of aggregate functions that will be used. Probability p which denotes the probability of choosing an attribute (or an aggregate function) is given as $p = \frac{h}{100}$ if $h > 0$, and $p = 0$, otherwise. Note that if $depthImportance = 0$, then $h = 100$ and $p = 1$, which means that all attributes (and aggregates) at all levels will be chosen during propositionalization. Our optimization algorithm first calculates which aggregation functions can be applied to the certain data type, and then with calculated probability p, it chooses a subset from the set of aggregate functions. Similar calculations are made when it chooses columns from the tables.

6 The algorithm

With the basic operations presented in the previous sections we can now define the basic propositionalization algorithm. The algorithm will traverse the data model graph and repeatedly use the summarization operation to project data from one table onto another, until all information has been aggregated at the target table. Although this repeated summarization can be done in several ways, we will describe our algorithm. The algorithm performs a depth-first search (DFS) through the data model, up to a specified depth. Whenever the recursive algorithm reaches its maximum depth or a leaf in the graph, it will summarize the relevant table by summarizing it on the parent in the DFS tree. Internal nodes in the tree will again be summarized after all their children have been summarized. This means that attributes considered deep in the tree may be aggregated multiple times.

The process continues until all tables are summarized on the target table. The following pseudo code describes the algorithm more formally:

```
summarize (Table T, Datamodel M, int d)
    V := T;
    if d > 0
        for all associations A from T in M
            W := summarize(T.getTable(A), M, d-1);
            S := collect(W, A);
        V.add(S);
    return V;
```

The effect of summarize is that each attribute appearing in a table other than the target table will appear several times in the aggregated form in the resulting view. This multiple occurrence happens for two reasons. The first reason is that tables may occur multiple times in the DFS tree because they can be reached through multiple paths in the data model. Each path will produce a different aggregation of the available attributes. The second reason is related to the choices of aggregate class at each summarization along a path in the data model. This choice and the fact that aggregates may be combined in longer paths produces multiple occurrences of an attribute per path.

7 Experiments

In order to acquire empirical knowledge about the effectiveness of our approach, we have tested our algorithm on two well-known multi-relational datasets. These datasets were chosen because they show a variety of data models that occur frequently in many multi-relational problems. They are Financial[9] and Musk[2]. For experiments we use 10-fold cross-validation averaged accuracy and also for Musk dataset we have computed the average accuracy by leave-one-out cross-validation.

Financial: Financial database is taken from the Discovery Challenge organized at PKDD 99 and PKDD 2000 [9]. The database is based on data from a Czech bank. It describes the operations of 5369 clients holding 4500 accounts. The data is stored in 8 tables, 4 of which describe the usage of products, such as credit cards and loans. Three tables describe client and account information, and the remaining table contains demographic information about 77 Czech districts. We have chosen the *account* table as our target table. Although we thus have 4500 examples, the dataset contains a total of 1079680 records. Our aim was to determine the loan-quality of an account, that is the existence of a loan with status 'finished good loan' or 'running good loan'. We present results obtained using *account* as a target table. Setting *account* table as target table, we set $depthImportance = 0, 20, 200$. The results are presented in Table 1. A near perfect score of 99.9% was achieved using $depthImportance = 0$. Classification accuracy in the table is obtained using J48 data mining algorithm with binary splits and 10-fold cross-validation. Experiments using $depthImportance > 0$ were repeated ten times and the scores are averaged, because of their stochastic nature.

depthImportance	num. of attributes	accuracy
0	753	99.9%
20	193	94.6%
200	98	85.3%

Table 1. Results on Financial dataset

Musk: The Musk database describes molecules occurring in different conformations. Each molecule is either musk or non-musk and one of the conformations determines this property. Such a problem is known as a multiple-instance problem, and will be modeled by two tables molecule and conformation, joined by a one-to-many association. Conformation contains a molecule identifier plus 166 continuous features. Molecule just contains the identifier and the class. Table 2 shows the results of our algorithm using different *depthImportance* parameter. In Table 3 we give comparison to other already published results, and we can see that our algorithm performs better then Tilde, Back-propagation, C4.5. It outperforms also one of the axis-parallel rectangles algorithms but it performs worse than the other.

depthImportance	num. of attributes	10-fold CV	Leave-one-out
0	1162	81.37%	80.40%
20	574	80.72%	79.41%
200	232	78.43%	82.02%

Table 2. Results on Musk dataset

Algorithm	Accuracy
iterated-discrim APR	89.20%
Our exhaustive	**81.37%**
GFS elim kde APR	80.40%
Our Average	**80.17%**
Tilde	79.40%
Back-propagation	67.70%
C4.5	58.80%

Table 3. Comparison of results with other systems on Musk dataset using ten fold cross-validation

From the results, we conclude that stochastic optimization leads to smaller number of attributes compared to previously proposed algorithms, but classification accuracy stays high enough. Note that previously proposed aggregation methods [3,6] are without any optimization, which means exhaustive methods are used. Also note that our algorithm with $depthImportance = 0$ is also exhaustive search algorithm. Roughly speaking, our algorithm allows to the user better flexibility and tuning during the propositionalization. For Financial dataset, we found most interesting the results obtained by setting $depthImportance = 20$ for attribute and aggregate selection and unlimited depth. Setting such parameters we get classification accuracy of near 95% which is good enough. But increasing this parameter leads to worse results. Classification accuracy of 95% is very often considered as excellent accuracy of a model. Our algorithm performs very good on Musk dataset as well, and we belive that applying better algorithm than J48 after propositionalization better results can be obtained. We conclude that lowering accuracy with increasing *depthImportance* parameter seems to be trade-off between time consumption, number of attributes and information that they provide to the data mining algorithm.

8 Discussion and further work

The experimental testing and results demonstrate that our approach is good in feature generation, and easily tunable with parameters. Source code of our system can be found on the site of authors [8]. It is open source and written in Java. The system is easy to use, and even more important characteristic, it is open system which allows upgrades and adding features very fast. Currently, our system works with postgresql database using standard JDBC connection, but it can be easily upgraded to be used with other known relational database management systems. We believe, that good algorithm for feature generation as we propose here, and good algorithm for feature selection applied to our results will lead to very good results. For further work, making of one algorithm for both techniques(propositionalization and feature selection) is also considered. Main directions are building a data mining model after each summarization and then using stochastic optimization to choose which attribute to be kept.

References

1. L. Dehaspe and H. Toivonen. Discovery of frequent datalog patterns. *Data Mining and Knowledge Discovery*, 3(1):7–36, 1999.
2. T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez. Solving the multiple-instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.
3. A.J. Knobbe, M. de Haas, and A. Siebes. Propositionalisation and aggregates. In L. De Raedt and A. Siebes, editors, *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, volume 2168 of *Lecture Notes in Artificial Intelligence*, pages 277–288. Springer, 2001.
4. S. Kramer. *Relational Learning vs. Propositionalization: Investigations in Inductive Logic Programming and Propositional Machine Learning.* PhD thesis, Vienna University of Technology, Vienna, Austria, 1999.
5. S. Kramer, N. Lavrač, and P. Flach. Propositionalization approaches to relational data mining. In S. Džeroski and N. Lavrač, editors, *Relational Data Mining*, pages 262–291. Springer-Verlag, 2001.
6. M.-A. Krogel and S. Wrobel. Facets of aggregation approaches to propositionalization. In T. Horváth and A. Yamamoto, editors, *Proceedings of the Work-in-Progress Track at the 13th International Conference on Inductive Logic Programming*, pages 30–39, 2003.
7. A. Srinivasan, R.D. King, and D.W. Bristol. An assessment of ILP-assisted models for toxicology and the PTE-3 experiment. In *Proceedings of the Ninth International Workshop on Inductive Logic Programming*, volume 1634 of *Lecture Notes in Artificial Intelligence*, pages 291–302. Springer-Verlag, 1999.
8. Prored system. Source code can be found at `http://kt.ijs.si/ValentinGjorgjioski/prored`.
9. *Workshop notes on Discovery Challenge.* PKDD, 1999.