Nada Lavrač, Vid Podpečan,
Marko Robnik-Šikonja

Representation Learning

Propositionalization and Embeddings

November 24, 2020

Springer

Contents

Chapter 1
Propositionalization of Multi-Relational Data

Inductive Logic Programming, Relational Learning and Relational Data Mining address the task of learning models or patterns from multi-relational data. Unlike the direct learning approaches to relational data developed in the Inductive Logic Programming (e.g., learning in first order logic), this chapter addresses the propositionalization, i.e. the approach of first transforming a relational database into a single-table representation, followed by a model/pattern construction step using a standard learning algorithm. The chapter outlines the propositionalization algorithms for transforming relational data into propositional tabular data format. The chapter is structured as follows. In Section 1.1, we present the problem of relational learning, including its definition, the main approaches, and an illustrative toy example which we expand in further sections. Section 1.3 describes the propositionalization, and briefly discusses feature construction approaches used in standard propositionalization algorithms. In Section 1.4 we present alternatives to the relational feature construction, used in propositionalization, and in Section 1.5 we describe wordification, efficient and practically important propositionalization method, which we exploit in the following chapters. Inspired by text mining, in wordification each original instance is transformed into a-kind-of 'document' represented as a BOW vector of weights of simple features, which can be interpreted as 'words'. The 'words' constructed by wordification can be subsequently weighted by their TF-IDF value. Section 1.6 presents an approach to training deep neural networks on propositionalized relational data, named Deep Relational Machines (DRM). We end the chapter with Section 1.7 with illustration of selected approaches.

1.1 Relational learning

Standard machine learning and data mining algorithms induce models learned from a given data table, where each example corresponds to a single row, i.e. a single fixed-length attribute-value tuple. These learners, which can be referred to as *propositional learners*, thus use as input a propositional representation. However, the re-

striction to a single table poses a challenge for data that are naturally represented in a relational representation, referred to as *relational data* or *multi-relational data*. *Relational learning* problems cannot be directly represented with a tabular representation without loss of information. These problems, which include the analysis of complex structured data or complex networks, are naturally represented using multiple relations.

As mentioned in **??**, learning from relational data can be approached in two main ways:

Multi-relational learning and inductive logic programming. These approaches, referred to as relational learning (RL) [Quinlan, 1990], Inductive Logic Programming (ILP) [Muggleton, 1992, Lavrač and Džeroski, 1994, Srinivasan, 2007], Relational Data Mining (RDM) [Džeroski and Lavrač, 2001], Statistical Relational Learning (SRL) [Getoor, 2007], learn a relational model or a set of relational patterns *directly on the multi-relational data*.

Propositionalization. Propositionalization techniques transform a relational representation into a propositional single-table representation by constructing complex features [Lavrač et al., 1991, Kramer et al., 2000, Krogel et al., 2003, Železný and Lavrač, 2006, Kuželka and Železný, 2011], and then use a propositional learner on the transformed data table.

The former strategy requires the design of dedicated algorithms for analyzing data in a relational representation, while the latter strategy involves a data preprocessing step, named *propositionalization*, enabling the user to transform the data into a tabular data format and subsequently employ a full range of standard propositional learning algorithms on the transformed tabular data representation.

1.2 Relational data representation

After presenting some notational conventions used for describing rules in Section 1.2.1, this section uses an illustrative example shown in Section 1.2.2 to present two data description formats of particular interest: the Prolog format in Section 1.2.3 and the relational database format in Section 1.2.4, respectively.

1.2.1 Notational conventions for describing rules

Depending on the context, one can use three different notations for rules:

Explicit. This notation explicitly marks the condition part (`IF`), the connection of the features in the conjunctive condition (via `AND`), and the conclusion (`THEN`). It is the most readable form and will be mostly used in application-oriented parts, where it is important to understand the semantics of the rule. A rule described in this formalism is illustrated below.

```
IF   Shape = triangle
 AND Color = red
 AND Size  = big
THEN Class = Positive
```

Formal. This notation is based on formal (propositional) logic. The implication sign ($\leftarrow$) is typically written from right to left (as in Prolog, see below), but it may also appear in the other direction. It will be used if theoretical properties of rule learning algorithms are discussed. A rule described in this formalism is shown below.

$Class = Positive \leftarrow Shape = triangle \wedge Color = red \wedge Size = big.$

Prolog. This notation uses the Prolog syntax [Bratko, 1990] of first-order logic used in ILP . It will mostly be used on examples involving relational learning. A rule described in this formalism is shown below.

```
positive(X) :- shape(X,triangle),
               color(X,red),
               size(X,big).
```

1.2.2 Illustrative example

The illustrative example used in this chapter is Michalski's East-West trains challenge [Michie et al., 1994b], illustrated in Figure 1.1, where the goal of the learned model is to classify the direction of an unseen train.

The training data set consists of ten examples, i.e. ten trains `t1`, ..., `t10`, where the predicates `eastbound` and `westbound` indicate the class, i.e. whether the train is eastbound or westbound.

eastbound: `eastbound(t1)`, `eastbound(t2)`, ..., `eastbound(t5)`
westbound: `westbound(t6)`, `westbound(t7)`, ..., `westbound(t10)`

We use this example to present the Prolog and the relational database representations in Section 1.2.3 and Section 1.2.4, respectively.

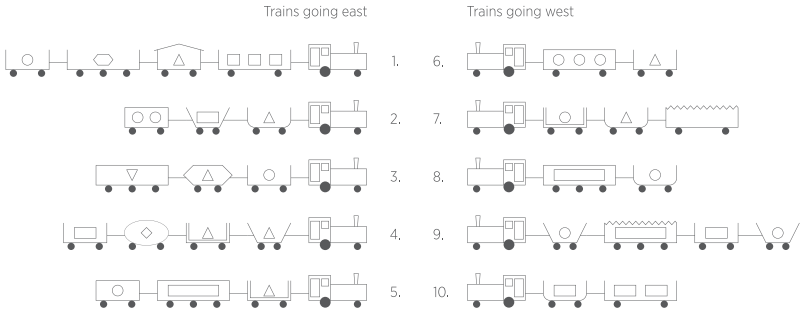

Fig. 1.1 The ten-train East-West trains challenge.

1.2.3 Example using a logical representation

In a logical representation of the problem, each train is described by a set of ground facts which describe the properties of the train and its cars. In the logic programming notation used in programming language Prolog, the set of logical facts describing the first train is given in Table 1.1.

In this example, each train consists of 2–4 cars. The cars have properties like shape (`rectangular`, `oval`, `u-shaped`), length (`long`, `short`), number of wheels (`2`, `3`), type of roof (`no_roof`, `peaked`, `jagged`), shape of load (`circle`, `triangle`, `rectangle`), and number of loads (`1`–`3`).

For this small data set consisting of ten instances, a relational learner could induce the following Prolog rule distinguishing between eastbound and westbound trains, which states that "Train T is eastbound if it contains a short closed car".

```
eastbound(T) :-
   hasCar(T,C), clength(C,short),
   not croof(C,no_roof).
```

Table 1.1 Prolog representation of the first train in the East-West trains challenge data set. The predicate `cnumber` contains the number of cars for a given train, `hasCar` connects a car to a given train, `cshape`, `clength`, croof, and `cwheels` define properties of a given car, `hasLoad` associates a given load with the given car, and `lshape` and `lnumber` describe properties of a given load.

```
eastbound(t1).

cnumber(t1,4).

hasCar(t1,c11).            hasCar(t1,c12).
cshape(c11,rectangular).   cshape(c12,rectangular).
clength(c11,long).         clength(c12,short).
croof(c11,no_roof).        croof(c12,peak).
cwheels(c11,2).            cwheels(c12,2).
hasLoad(c11,l11).          hasLoad(c12,l12).
lshape(l11,rectangular).   lshape(l12,triangular).
lnumber(l11,3).            lnumber(l12,1).

hasCar(t1,c13).            hasCar(t1,c14).
cshape(c13,rectangualar).  cshape(c14,rectangular).
clength(c13,long).         clength(c14,short).
croof(c13,no_roof).        croof(c14,no_roof).
cwheels(c13,3).            cwheels(c14,2).
hasLoad(c13,l13).          hasLoad(c14,l14).
lshape(l13,hexagonal).     lshape(l14,circular).
lnumber(l13,1).            lnumber(l14,1).
```

1.2.4 Example using a relational database representation

In the considered East-West trains challenge, illustrated in Figure 1.1, instead of data representation in the form of Prolog facts illustrated in Table 1.1, the data can be stored in a relational database, consisting of three relational tables presented in Table 1.2.

In a database terminology, a *primary key* of a table is a unique identifier of each record. For example, `trainID` is the primary key for records in the `TRAIN` table, while `carID` and `loadID` are primary keys in the `CAR` and `LOAD` tables, respectively. A *foreign key* is a key used to link two tables together. A foreign key is a column or a combination of columns whose values match a primary key in a different table. For example, the column `train` in the `CAR` table is its foreign key, linking records in this table (i.e. cars) to their train, described in the `TRAIN` table. The same is true for the `car` column in the `LOAD` table, where `car` plays a role of the foreign key, linking loads to their cars.

In a relational database (RDB), the problem is given as a set of relations $\{R_1,\ldots,R_n\}$ and a set of foreign-key connections between the relations denoted by $R_i \rightarrow R_j$, where R_i has a foreign-key pointing to relation R_j. The foreign-key connections correspond to the relationships in an entity-relationship (ER) diagram.

Take the ER diagram, illustrated in Figure 1.2, which represents the data model describing the structure of the data. It shows three relations appearing in the East-West train challenge: the `Train`, `Car` and the `Load` relational tables. The boxes in the ER diagram indicate *entities*, which are individuals or parts of individuals: the `Train` entity is the individual, each `Car` is part of a train, and each `Load` is part of a car. The ovals denote attributes of entities. The diamonds indicate *relationships* between entities. There is a *one-to-many relationship* from `Train` to `Car`, indicat-

Table 1.2 A relational database representation of the East-West trains challenge data set.

TRAIN

trainID	eastbound
t1	true
⋮	⋮
t5	true
t6	false
⋮	⋮

LOAD

loadID	lshape	lnumber	car
l11	rectangular	3	c14
l12	triangular	1	c13
l13	hexagonal	1	c12
l14	circular	1	c11
⋮	⋮	⋮	⋮

CAR

carID	cshape	clength	croof	cwheels	train
c11	rectangular	long	no_roof	2	t1
c12	rectangular	short	peak	2	t1
c13	rectangular	long	no_roof	3	t1
c14	rectangular	short	no_roof	2	t1
⋮	⋮	⋮	⋮	⋮	⋮

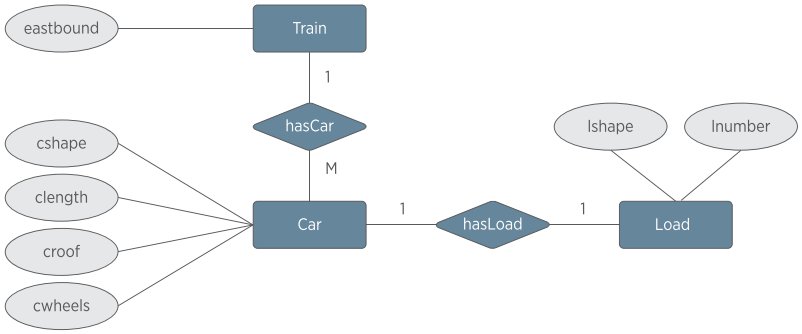

Fig. 1.2 Entity-relationship diagram for the East-West trains challenge, including a *one-to-many relationship* `hasCar` from `Train` to `Car` (corresponding to the `hasCar(T,C)` predicate), and a *one-to-one relationship* `hasLoad` between `Car` and `Load` (corresponding to the `hasLoad(C,L)` predicate).

ing that each train can have an arbitrary number of cars but each car is contained in exactly one train; and a *one-to-one relationship* between `Car` and `Load`, indicating that each car has exactly one load and each load is part of exactly one car.

Entity-relationship diagrams can be used to choose a proper logical representation for the data. If we store the data in a relational database, the most obvious representation is to have a separate table for each entity in the domain, with relationships being expressed by foreign keys. This is not the only possibility: for instance, since the relationship between `Car` and `Load` is one-to-one, both entities could be combined in a single table, while entities linked by a one-to-many relationship cannot be combined without either introducing significant redundancy or significant loss of information, e.g., introduced through aggregate attributes. Note that one-to-many relationships distinguish relational learning and Inductive Logic Programming from propositional learning.

In propositionalization, we use the entity-relationship diagram to define types of objects in the domain, where each entity corresponds to a distinct type. The data model constitutes a *language bias* that can be used to restrict the hypothesis space (i.e. the space of possible models) and guide the search for good models. In most problems, only individuals and their parts exist as entities, which means that the entity-relationship model has a tree-structure with the individual entity at the root and only *one-to-one* or *one-to-many* relations in the downward direction (i.e. not containing any *many-to-many* relations). Representations with this restriction are called *individual-centered representations* [Flach and Lachiche, 1999a]. This restriction determines the language bias, constraining the relational database input to propositionalization.

1.3 Propositionalization

Propositionalization (a transformation of multi-relational data into a single table format) cannot always be done without loss of information. It is a suitable approach to relational learning when the problem at hand is *individual-centered*, i.e. when learning occurs only at the level of the individual. Take as example a problem of classifying authors into research fields given a citation network—in this case the author is the individual and learning is performed at the author level (e.g., assigning class labels to authors).

Accepting some loss of information, the propositionalization can be achieved by using aggregation queries on the relational database in order to compress the data into a single table format. Feature construction through aggregation queries has been addressed in early work on propositionalization of Kramer et al. [2001] and Krogel et al. [2003], which includes an extensive overviews of different feature construction approaches. Alternative propositionalization approaches perform relational feature construction.

Relational learners that directly use the multi-relational representations intertwine feature construction and model construction. In propositionalization, these two steps are separated. The workload of finding good relational features is performed by the propositionalization algorithm, while the work of combining these features to produce a good model is offloaded to the propositional learner having its own hypothesis language bias, e.g., decision trees, classification rules, SVM, or more recently deep neural networks as we show in **??**.

Propositionalization can be performed with many ML or DM tasks in mind: classification, association discovery, clustering, etc. In this chapter, we focus on the classification task, as this is its most frequent application.

1.3.1 Relational feature construction

Most of the propositionalization algorithms, use ILP notation for features and rules. Individual features are described by *literals* or *conjunctions of literals*. For example, in the toy example of Section 1.2, the features describing the cars are either literals (e.g., `clength(C,short)`), or conjunctions of literals (e.g., `clength(C,short)`, `not croof(C,no_roof)`).

Given the training examples and the background knowledge, an ILP learner can induce the following two Prolog rules describing eastbound trains:

```
eastbound(T) :-
   hasCar(T,C), clength(C,short),
   not croof(C,no_roof).
eastbound(T) :-
   hasCar(T,C1), clength(C1,short),
   hasCar(T,C2), not croof(C2,no_roof).
```

While the first rule (the same as in example in Section 1.2.3) expresses that trains which have a short closed car are going East, the second rule states that trains which have a short car and a (possibly different) closed car are going East. This rule is more general than the first, covering all the instances covered by the first rule, and in addition covering the instances where the short car is different from the closed car. In the example, `T` is a global variable denoting any train, while `C`, `C1` and `C2` are existentially quantified local variables introduced in the rule body, denoting the existence of a car with a certain property.

Note that in the terminology used in this chapter, we say that the body of the first rule consists of a single *relational feature*, while the body of the second rule contains two distinct relational features. Formally, a feature is defined as a minimal set of literals such that no local (i.e. existential) variable occurs both inside and outside that set. The main point of relational features is that they localize variable sharing: the only variable which is shared among features is the global variable occurring in the rule head. This can be made explicit by naming the features:

```
hasShortCar(T)  :-
   hasCar(T,C), clength(C,short).
hasClosedCar(T) :-
   hasCar(T,C), not croof(C,no_roof).
```

Using these named features as background predicates, the second rule above can be translated into a rule without local variables:

```
eastbound(T) :- hasShortCar(T), hasClosedCar(T).
```

This rule only refers to properties of trains, and hence could be expressed extensionally by a single table describing trains in terms of these properties. In the following, we will see how we can automatically construct such relational features and the corresponding propositional table.

As manual feature construction is complex and/or unfeasible, the task of propositionalization is to automate construction of relevant features which can act as queries about each individual. The queries will be evaluated as *true/false* on the original data when constructing a transformed data table, and form truth values of constructed relational features. The actual goal of propositionalization is thus to automatically generate a number of relevant features about an individual that the learner can use to construct a model. In the machine learning literature, automated feature construction is known as *constructive induction*. Propositionalization is thus a form of constructive induction, since it involves changing the representation for learning.

Formally, a *relational feature* expresses a property of an individual by a conjunction of predicates and properties, composed as follows:

1. there is exactly one free variable which will play the role of the global variable in rules (e.g., for the trains example, the global variable `T` represents any train);

2. each predicate introduces a new (existentially quantified) local variable, and uses either the global variable or one of the local variables introduced by other predicates (e.g., the predicate `hasCar(T,C)` introduces a new local variable `C` which stands for any car of train `T`);
3. properties do not introduce new variables (e.g., `lshape(L,triangular)` stands for a triangular shape of a load, provided that the load variable `L` has been already instantiated);
4. all variables are 'consumed', i.e. used either by a predicate or a property.

Actual relational feature construction can be restricted by parameters that define the maximum number of literals constituting a feature, number of variables, and number of occurrences of individual predicates. The following relational feature, denoting the property of a train having a car with a triangular load, can be constructed for the trains example, in the language bias allowing for up to 4 literals and 3 variables:

```
trainFeature42(T) :-
   hasCar(T,C), hasLoad(C,L), lshape(L,triangular).
```

This example shows that a typical feature contains a chain of predicates, closed off by one or more properties. Properties can also establish relations between parts of the individual, e.g., the following feature expresses the property of 'having a car whose shape is the same as the shape of its load':

```
trainFeature978(T) :-
   hasCar(T,C),cshape(C,CShape),
   hasLoad(C,L),lshape(L,LShape),
   CShape = LShape.
```

1.3.2 Data transformation in propositionalization

In the trains classification task, a complete propositional representation of the problem would define a set of features, acting as queries $q_i \in Q$ that return *true* (value 1) or *false* (value 0) for a given train.

Transformed data table. We demonstrate data transformation on the East-West trains challenge, using a relational feature construction algorithm RSD [Železný and Lavrač, 2006]. We allow construction of features that use at most two predicates that introduce local variables, and at most two predicates that describe properties. There are 190 such features, as illustrated in the data table in Table 1.3. The direction attribute is not preprocessed; it is only appended to the transformed feature vectors.

Output of propositional rule learning. In model construction, the learner exploits the features describing individual instances to construct a classification model, e.g., a decision tree. Each node in the tree then contains a feature and has

Table 1.3 An outline of the propositional form of the East-West trains challenge.

T	f1(T)	f2(T)	f3(T)	f4(T)	...	f190(T)	eastbound(T)
t1	1	0	0	1	...	0	true
t2	1	1	0	1	...	1	true
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮
t6	0	0	1	1	...	1	false
t7	1	0	0	0	...	0	false
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

two branches (*true* and *false*). To classify unseen individuals, the classifier evaluates the features that are found in the decision tree nodes and follow the branches according to their answers to arrive at a classification in a leaf of the tree.
If we use a propositional rule learning algorithm, its output is a set of rules S, consisting of individual rules R. Rules are built from binary features $f \in F$, where F is the set of possible features. Taking Table 1.3 as input (using column `eastbound(T)` as the class attribute), an attribute-value rule learner such as CN2 [Clark and Niblett, 1989] can be used to induce if-then rules such as:

$$\texttt{eastbound(T)=true} \leftarrow \texttt{f1(T)=1} \wedge \texttt{f4(T)=1},$$

where `f1(T)` and `f4(T)` are two of the automatically generated features, and $\wedge$ denotes the logical conjunction operator.

1.4 Alternatives to relational feature construction

In propositionalization, relational feature construction is the most common approach to data transformation. LINUS [Lavrač et al., 1991] was one of the pioneering propositionalization approaches using automated relational feature construction. LINUS was restricted to generation of features that do not allow recursion and existential local variables, which means that the target relation cannot be many-to-many and self-referencing. The second limitation is more serious: the queries cannot contain joins (conjunctions of literals). SINUS [Lavrač and Flach, 2001], a descendant of LINUS, incorporates more advanced feature construction techniques. The LINUS approach had many followers, including relational subgroup discovery system RSD [Železný and Lavrač, 2006].

Aggregation approaches to propositional feature construction are a popular alternative to relational feature construction. An *aggregate* can be viewed as a function that maps from a set of records in a relational database to a single value. This adds a constructed attribute to one of the tables in the relational database. If an aggregation function considers values from only one table of a relational database, then aggregation is equivalent to feature construction. For instance, in the database representation of trains example in Table 1.2, one could add a binary aggregate column to the `CAR` table by testing the predicate '#cwheels is odd'. Aggregation becomes

more powerful if it involves more tables in the relational database. An example of a first-order aggregate would be a column added to the TRAIN table which computes the total number of wheels on all cars of the given train; this requires information from the CAR table. A second-order aggregate (encompassing all tables) could add a column to the TRAIN table representing its total number of wheels on cars containing rectangular-shaped loads.

Below we outline a selection of propositionalization approaches, while an interested reader can find extensive overviews of different feature construction approaches in the work of Kramer et al. [2001] and Krogel et al. [2003].

Relaggs [Krogel and Wrobel, 2001] stands for *rel*ational *agg*regation. It is a propositionalization approach that takes the input relational database schema as a basis for a declarative bias, using optimization techniques usually used in relational databases (e.g., indexes). The approach employs aggregation functions in order to summarize non-target relations with respect to the individuals in the target table.

1BC [Flach and Lachiche, 1999b] uses the propositional naive Bayes classifier to handle relational data. It first generates a set of first-order conditions and then uses them as attributes in the naive Bayes classifier. The transformation, however, is done in a dynamic manner, as opposed to standard propositionalization, which is performed as a static step of data preprocessing. This approach is extended by 1BC2 [Lachiche and Flach, 2003], which allows distributions over sets, tuples, and multisets, thus enabling the naive Bayes classifier to consider also structured individuals.

Tertius [Flach and Lachiche, 2001] is a top-down rule discovery system, incorporating first-order clausal logic. The main idea is that no particular prediction target is specified beforehand, hence Tertius can be seen as an ILP system that learns rules in an unsupervised manner. Its relevance for this survey lies in the fact that Tertius encompasses 1BC, i.e. relational data is handled through 1BC transformation.

RSD [Železný and Lavrač, 2006] is a relational subgroup discovery algorithm composed of two main steps: the propositionalization step and the (optional) subgroup discovery step. The output of the propositionalization step can be used also as input to other propositional learners. Using different biases, RSD efficiently produces an exhaustive list of first-order features that comply with the user-defined mode constraints, similar to those of Progol [Muggleton, 1995] and Aleph [Srinivasan, 2007]. RSD features satisfy the connectivity requirement, which imposes that no feature can be decomposed into a conjunction of two or more features.

HiFi [Kuželka and Železný, 2008] is a propositionalization approach that constructs first-order features with hierarchical structure. Due to this feature property, the algorithm performs the transformation in polynomial time of the maximum feature length. The resulting features are the shortest in their semantic equivalence class. The algorithm performs much faster than RSD for longer features.

RelF [Kuželka and Železný, 2011] constructs a set of tree-like relational features by combining smaller conjunctive blocks. The algorithm scales better than other state-of-the-art propositionalization algorithms.

Cardinalization [Ahmed et al., 2015] is designed to handle not only categorical but also numerical attributes in propositionalization. It handles a threshold on numeric attribute values and a threshold on the number of objects simultaneously satisfying the condition on the attribute. Cardinalization can be seen as an implicit form of discretization.

CARAF [Charnay et al., 2015] aggregates base features into complex compounds, similarly to Relaggs. While Relaggs tackles overfitting problem by restricting itself to relatively simple aggregates, CARAF incorporates more complex aggregates into a random forest, which ameliorates the overfitting effect.

Aleph [Srinivasan, 2007] is actually an ILP toolkit with many modes of functionality: learning of theories, feature construction, incremental learning, etc. Aleph uses mode declarations to define the syntactic bias. Input relations are Prolog clauses, defined either extensionally or intensionally. Aleph's feature construction functionality also means that it is a propositionalization approach.

Wordification [Perovšek et al., 2013, Perovšek et al., 2015] is a propositionalization method inspired by text mining that can be viewed as a transformation of a relational database into a corpus of text documents. The distinguishing property of Wordification is its efficiency when used on large relational data sets and the potential for using text mining approaches on the transformed propositional data. This approach is described in more detail in Section 1.5 below.

1.5 Wordification: Unfolding relational data into BoW vectors

Inspired by the text mining, this section presents a propositionalization approach to relational data mining, called *wordification*. Most other propositionalization techniques first construct complex relational features that act as attributes in the resulting tabular data representation. Contrary to that, the wordification generates much simpler features with the aim of achieving greater scalability.

1.5.1 Outline of the wordification approach

This section provides an informal description of the wordification approach, illustrated in Figure 1.3. The input to the wordification is a relational database, and the output is a set of feature vectors, which can be viewed as a corpus of text documents represented in the BoW vector format. Each text document represents an individual entry of the main data table. A document is described by a set of words (or features), where a word is constructed as a combination of the table name, name of the attribute and its discrete (or discretized) value (see line 4 in Algorithm 2):

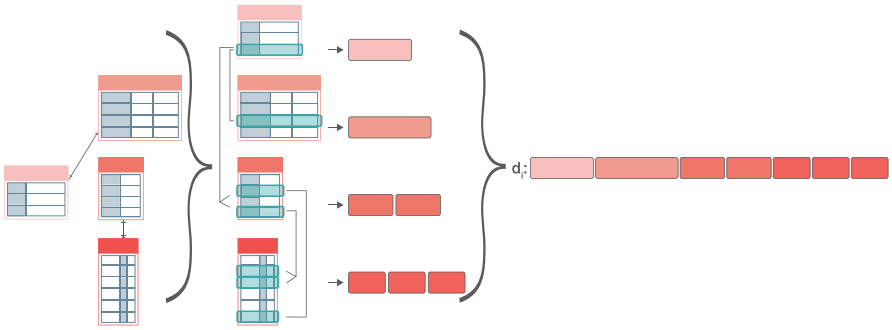

Fig. 1.3 The transformation from a relational database representation into the BoW feature vector representation. For each individual entry of the main table, one BoW vector d_i containing weights of 'words' is constructed. Here 'words' correspond to the features (attribute values) of the main table and the related tables.

$$[table\ name]\_[attribute\ name]\_[value]. \tag{1.1}$$

Such constructs are called *word-items* or simply *words* in the rest of the chapter. Note that values of every non-discrete attribute need to be discretized beforehand in order to be able to represent them as word-items. For each individual, the word-items are first generated for the main table and then for each entry from the related tables, and finally joined together according to the relational schema of the database (see line 10 in Algorithm 2).

In the described transformation there is some loss of information as a consequence of building the document for each instance (each individual row in the main table) by concatenating all word-items from multiple instances (rows) of the connected tables into a single document. To overcome this loss, n-grams of word-items, constructed as conjunctions of several word-items, can also be concatenated with the document. These concatenations of elementary word-items represent conjunctions of features cooccurring in individual instances (rows of joined tables). Technically, the n-gram construction takes every combination of length k of word-items from the set of all word-items corresponding to the given individual. The n-grams are concatenated as follows:

$$[word_1]\_\_[word_2]\_\_ \ldots \_\_[word_k], \tag{1.2}$$

where $1 \leq k \leq n$ and each word-item is a combination of the table name, name of the attribute and its discrete value. The instances are concatenated in a predetermined order, each using the "__" concatenation symbol.

In the rest of this section, we refer to individuals as documents, to features as words, and to the resulting representation as the BoW representation. For a given word w_i in document d_j from corpus D, the TF-IDF measure is computed using Equation **??** on page **??**, where a word with a high TF-IDF value will be considered important for the given document provided that it is frequent within this document and not frequent in the entire document corpus. Consequently, the weight of a word

provides an indication of how relevant is this feature for the given individual. The TF-IDF weights can then be used either for filtering out words with low importance or using them directly by a propositional learner.

We present further details of the wordification methodology by first presenting an illustrative example, followed by the presentation of the actual Wordification algorithm.

1.5.2 Illustrative example

The wordification approach is illustrated on a modified and substantially simplified version of the East-West trains challenge shown in Table 1.4, where we have only one East-bound and one West-bound train, each with just two cars with certain properties.

Table 1.4 Simplified East-West trains challenge to illustrate wordification.

TRAIN

trainID	eastbound
t1	eastbound
t5	westbound

CAR

carID	shape	roof	wheels	train
c11	rectangle	none	2	t1
c12	rectangle	peaked	3	t1
c51	rectangle	none	2	t5
c52	hexagon	flat	2	t5

In this simplified problem, the `TRAIN` table is the main table and the trains are the instances. We want to learn a classifier to determine the direction of an unseen train. For this purpose, the direction attribute is not pre-processed and is only appended to the resulting feature vector (list of words).

The underlying idea of wordification is to consider an individual (a row in the table) as a 'document' (or an item set) and individual attribute values as 'words' (or items) in this document (item set). In addition to simple 'words', more complex 'words' can be constructed by concatenating two/three 'words' into a bigram/trigram. In the example, for two trains `t1` and `t5`, the corresponding documents (one for each train) are generated, as shown in Table 1.5. After this, the documents are transformed into the BoW representation by first computing the attribute values shown in Table 1.6, and then calculating the TF-IDF values for each word of each document with the class attribute column appended to the transformed BoW table, as shown in Table 1.7. For simplicity, only unigrams and bigrams are shown in this example. After this textually inspired transformation, traditional machine learning methods can be employed on the transformed data set shown in Table 1.7.

Table 1.5 The data set from Table 1.4 in the symbolic BoW document representation.

```
t1: [car_roof_none, car_shape_rectangle, car_wheels_2,
    car_roof_none__car_shape_rectangle, car_roof_none__car_wheels_2,
    car_shape_rectangle__car_wheels_2, car_roof_peaked, car_shape_rectangle,
    car_wheels_3, car_roof_peaked__car_shape_rectangle,
    car_roof_peaked__car_wheels_3, car_shape_rectangle__car_wheels_3],
    eastbound
    ...
t5: [car_roof_none, car_shape_rectangle, car_wheels_2,
    car_roof_none__car_shape_rectangle, car_roof_none__car_wheels_2,
    car_shape_rectangle__car_wheels_2, car_roof_flat, car_shape_hexagon,
    car_wheels_2, car_roof_flat__car_shape_hexagon,
    car_roof_flat__car_wheels_2, car_shape_hexagon__car_wheels_2],
    westbound
    ...
```

Table 1.6 The basic BoW representation of database from Table 1.4 and its symbolic representation from Table 1.5 using the wordification approach. It consists the counts of 'words' appearing in the 'documents'. This is an intermediate step and the values are further weighted with TF-IDF scheme as shown in Table 1.7.

ID	car_shape_rectangle	car_roof_peaked	car_wheels_3	car_roof_peaked__car_shape_rectangle	car_shape_rectangle__car_wheels_3	...	Class
t1	2	1	1	1	1	...	eastbound
t5	1	0	0	0	0	...	westbound

Table 1.7 The transformed database from Table 1.4 using the wordification approach. It consists of TF-IDF values, which are zero if the 'word' appears in all the 'documents'. This final output can be given as an input to a propositional classifier.

ID	car_shape_rectangle	car_roof_peaked	car_wheels_3	car_roof_peaked__car_shape_rectangle	car_shape_rectangle__car_wheels_3	...	Class
t1	0.000	0.693	0.693	0.693	0.693	...	eastbound
t5	0.000	0.000	0.000	0.000	0.000	...	westbound

1.5.3 Wordification algorithm

The overall Wordification algorithm consists of two main transformation steps, presented in Algorithm 1 and Algorithm 2.

The algorithm starts recursive document construction on the instances of the main table (lines 4–9 in Algorithm 1). It first creates word-items for the attributes of the target table (lines 3–7 in Algorithm 2), followed by concatenations of the word-items and results of the recursive search through examples of the connecting tables (lines 9–17 in Algorithm 2). As this document construction step is done independently for each example of the main table, this allows simultaneous search along the tree of connected tables. Lines 4–9 in Algorithm 1 can be run in parallel, which allows for a significant speed-up. A common obstacle in parallel computing is memory synchronization between different subtasks, which is not problematic here as concurrent processes only need to share a cached list of subtrees. This list

```
wordification (T, p, k)
  Input  : target table T, pruning percentage p, maximal number of word-items per
           word k
  Output : Propositionalized table R with TF-IDF values, corpus of documents D
D ← []
W ← ∅                                                        // vocabulary set
for ex ∈ T do
    // entries of the target table
    d ← wordify(T, ex, k)                                    // construct the document
    D ← D + [d]                                              // append document to the corpus
    W ← W ∪ keys(d)                                          // extend the vocabulary
end
W ← prune(W, p)                                              // optional step
return [ calculateTFIDFs(D, W), D]
```

Algorithm 1: Pseudocode of wordification algorithm.

```
wordify (T, ex, k)
  Input  : table T, example ex from table T, maximal number of word-items per
           word k
  Output : document collection d
d ← {}                                  // hash table with the default size of 0
for i ← 1 to k do                                  // for all word-item lengths
    for comb ∈ attrCombs(ex, k) do // attr. combinations of length k
        d[word(comb)] ← d[word(comb)] + 1
    end
end
// for every connected table through an example
for secTable ∈ connectedTables(T) do
    for secEx ∈ secTable do
        if primaryKeyValue(ex) = foreignKeyValue(secEx) then
            for (word, count) ∈ wordify(secTable, secEx, k) do
                d[word] ← d[word] + count
            end
        end
    end
end
return d
```

Algorithm 2: Psudo code of creation of one document in wordification.

stores the results of subtree word concatenations in order to visit every subtree only once.

As wordification can produce a large number of features (words), especially when the maximal number of n-grams per word-items is large, we perform pruning of words that occur in less than a predefined percentage (5% on default) of documents. This reduces the size of trees by removing sections of the tree that is expected to provide little power for instance classification.

The constructed features are simple, and as we do not explicitly use existential variables in the new features (words), we instead rely on the TF-IDF measure to implicitly capture the importance of a word for a given individual. In the context of text mining, TF-IDF value reflects how representative is a certain feature (word) for a given individual (document).

1.5.4 Improved efficiency of Wordification

This section presents an implementation of the Wordification algorithm, developed with the aim to improve its *scalability*, achieved by the following design decisions [Lavrač et al., 2020]:

1. The input no longer needs to be hosted in a relational database. The algorithm supports SQL conventions, as commonly used in the ILP community[1]. This modification renders the method completely local, enabling offline execution without additional overhead. Such setting also offers easier parallelism across computing clusters.
2. The algorithm is implemented in Python 3 with minimum dependencies for computationally more intense parts, such as the Scikit-learn [Pedregosa et al., 2011], Pandas, and Numpy libraries [Van Der Walt et al., 2011]. All database operations are implemented as array queries, filters or similar.
3. As shown by Perovšek et al. [2015], Wordification's caveat is extensive sampling of (all) tables. We relax this constraint to close (up to second order) foreign key neighborhood, notably speeding up the relational item sampling part, but with some loss in terms of relational item diversity. For larger databases, minimum relational item frequency can be specified, constraining potentially noisy parts of the feature space.

One of the original Wordification's most apparent problems is its spatial complexity. This issue is addressed as follows:

1. Relational items are hashed for minimal spatial overhead during sampling.
2. During construction of the final representation, a sparse matrix is filled based on relational item occurrence.
3. The matrix is serialized directly into list-like structures.
4. Only the final representation is stored as a low-dimensional (e.g., 32) dense matrix.

This implementation of wordification is used in two pipelines unifying propositionalization and embeddings, presented in **??**, where in **??** we first present entity embeddings, a more general methodology capable of supervised, as well as unsupervised embeddings of many entities, including texts and knowledge graphs. Based on that, we present in **??** two unifying methods, PropDRM and PropStar, which combine propositionalization and embeddings and benefit from the advantages of both,

[1] https://relational.fit.cvut.cz/

i.e. they capture relational information through propositionalization and apply deep neural networks to obtain its dense embeddings. Both of these two pipelines use the version of the Wordification algorithm presented in this section.

1.6 Deep Relational Machines

Deep neural networks are effective learners in numeric space, capable of constructing intermediate knowledge concepts and thereby improving the semantics of baseline input representations. Training deep neural networks on propositionalized relational data wes explored by Srinivasan et al. [2019], following the work of Lodhi [2013], where Deep Relational Machines (DRMs) were first introduced. In Lodhi's work, the DRMs used bodies of the prolog sentences (first order Horn clauses) as inputs to the restricted Boltzmann machines. For example, consider the following propositional representation of five instances (rows), where complex features are comprised of conjuncts of features f_i, as illustrated in Figure 1.4.

Instance	$f_1 \wedge f_2$	$f_3 \wedge f_2$	$f_1 \wedge f_3$	$f_5 \wedge f_2$	$f_4 \wedge f_1 \wedge f_5$	Class
1	[1	1	1	1	0]	+
2	[0	1	0	0	1]	+
3	[0	0	1	0	0]	-
4	[0	1	0	0	1]	-
5	[1	0	0	0	1]	-

Fig. 1.4 An example input to a deep relational machine that operates on the instance level.

The propositionalized data set P is usually a sparse matrix, which can represent additional challenge for neural networks. The DRMs proposed by Lodhi [2013] were used for prediction of protein folding properties, as well as mutagenicity assessment of small molecules. This approach used feature selection with information theoretic measures such as information gain as the sparse matrix resulting from the propositionalization was not suitable as an input to the neural network. The initial studies regarding DRMs explored how deep neural networks could be used as an extension of relational learning.

Recently, promising results were demonstrated in the domain of molecule classification [Dash et al., 2018] using the ILP learner Aleph in its propositionalization mode for feature construction. After obtaining propositional representation of data, the obtained data table was fed into a neural network to predict the target class (e.g., a molecule's activity). Again, sparsity and size of the propositionalized representation was a problem for deep neural networks.

Concerning the interpretability of DRMs, the work of Srinivasan et al. [2019] proposes a logical approximation of the LIME explanation method [Ribeiro et al., 2016] and shows how it can be efficiently computed.

In summary, DRMs address the following issues at the intersection of deep learning and relational learning:

- DRMs demonstrated that deep learning on propositionalized relational structures is a sensible approach to relational learning.
- Their input is comprised of logical conjuncts, offering the opportunity to obtain human-understandable explanations.
- DRMs were successfully employed for classification and regression.
- Ideas from the area of representation learning have only recently been explored in the relational context [Dumančić et al., 2018], indicating there are possible improvements both in terms of execution speed, as well as more informative feature construction on the symbolic level.

Development of DRMs that are efficient with respect to both space and time is an ongoing research effort. Building on the ideas of DRMs, a variant of this approach, called PropDRM [Lavrač et al., 2020] is capable of learning directly from large, sparse matrices that are returned from Wordification of a given relational database, rather than using feature selection or the output of Aleph's feature construction approach. An efficient implementation of PropDRM is presented in **??**.

1.7 Implementation and reuse

This section demonstrates the wordification algorithm and introduces the *pythonrdm* Python package which simplifies relational data mining by integrating wrappers for several RDM algorithms.

1.7.1 Wordification

The wordification approach is demonstrated on the East-West trains challenge [Michie et al., 1994a]. The data is provided in the CSV (comma separated values) format, where files contain additional headers defining relations between tables (i.e. their primary and foreign keys). We demonstrate different settings of the the wordification algorithm. The resulting features are ranked, selected and used with a a decision tree learner to build a classifier. The Jupyter notebook with code is available in the book repository: https://github.com/vpodpecan/representation_learning/blob/master/Chapter4/wordification.ipynb.

1.7.2 The python-rdm package

While several relational data mining (RDM) tools are open source, they are not easily accessible for machine learning users working in this area. The *python-rdm*[2] package is a collection of wrappers for RDM algorithm implementations that aims to remedy this problem. As the input data, the package supports several relational databases, such as MySQL, PostgreSQL, and SQLite, as well as plain CSV text files. Its complete documentation is available online[3]. Some properties of the algorithms supported by *python-rdm* are compared in Table 1.8.

Table 1.8 A brief comparison of propositionalization approaches, implemented in the *python-rdm* package.

Approach	*Property*			
	Numerical	*Complete*	*Non-redundant*	*Recursive*
RSD	-	X	X	-
RelF	-	X	X	-
HiFi	-	-	X	-
LINUS	-	-	-	-
DINUS	-	-	-	-
SINUS	-	-	-	-
RELAGGS	X	-	-	-
Wordification	-	X	X	-
Stochastic	X	-	-	-
Aleph	-	-	-	X
Progol	-	-	-	X
Safarii	X	-	-	-

Our demonstration invokes the RSD relational data mining system to obtain tabular representation of the East-West trains challenge [Michie et al., 1994b] where data is stored in a remote MySQL database. Once the data is converted into the appropriate format, the code invokes RSD and induces features. The rest of the notebook is similar to the Wordification example in Section 1.7.1. The Jupyter notebook with code is available in the book repository: https://github.com/vpodpecan/representation_learning/blob/master/Chapter4/python-rdm.ipynb.

References

Chowdhury Farhan Ahmed, Nicolas Lachiche, Clément Charnay, Soufiane El Jelali, and Agnès Braud. Flexible propositionalization of continuous attributes in relational data mining. *Expert Systems with Applications*, 42(21):7698–7709, 2015.

[2] https://pypi.org/project/python-rdm/

[3] https://python-rdm.readthedocs.io/en/latest/

Ivan Bratko. *Prolog Programming for Artificial Intelligence*. Addison-Wesley, Wokingham, 2nd edition, 1990.

Clément Charnay, Nicolas Lachiche, and Agnès Braud. CARAF: Complex aggregates within random forests. In *Inductive Logic Programming - 25th International Conference, ILP 2015*, pages 15–29, 2015.

Peter Clark and Tim Niblett. The CN2 induction algorithm. *Machine Learning*, 3 (4):261–283, 1989.

Tirtharaj Dash, Ashwin Srinivasan, Lovekesh Vig, Oghenejokpeme I Orhobor, and Ross D King. Large-scale assessment of Deep Relational Machines. In *Proceedings of the International Conference on Inductive Logic Programming*, pages 22–37, 2018.

Sebastijan Dumančić, Tias Guns, Wannes Meert, and Hendrik Blockleel. Auto-encoding logic programs. In *Proceedings of the International Conference on Machine Learning*, 2018.

Sašo Džeroski and Nada Lavrač, editors. *Relational Data Mining*. Springer, Berlin, 2001.

Peter Flach and Nicholas Lachiche. 1BC: A first-order Bayesian classifier. In *Proceedings of the 9th International Workshop on Inductive Logic Programming (ILP-99)*, pages 92–103. Springer, 1999a.

Peter Flach and Nicholas Lachiche. Confirmation-guided discovery of first-order rules with Tertius. *Machine Learning*, 42(1/2):61–95, 2001.

Peter Flach and Nicolas Lachiche. 1bc: A first-order bayesian classifier. In *International Conference on Inductive Logic Programming*, pages 92–103. Springer, 1999b.

Lise Getoor. *Introduction to Statistical Relational Learning*. The MIT Press, 2007.

Stefan Kramer, Bernhard Pfahringer, and Christoph Helma. Stochastic propositionalization of non-determinate background knowledge. In *Proceedings of the 8th International Conference on Inductive Logic Programming (ILP-2000)*, pages 80–94, 2000.

Stefan Kramer, Nada Lavrač, and Peter Flach. Propositionalization approaches to relational data mining. In *Relational Data Mining*, pages 262–291. Springer-Verlag, 2001.

Mark A Krogel and Stefan Wrobel. Transformation-based learning using multirelational aggregation. In *Proceedings of International Conference on Inductive Logic Programming*, pages 142–155. Springer, 2001.

Mark A. Krogel, Simon Rawles, Filip Železný, Peter Flach, Nada Lavrač, and Stefan Wrobel. Comparative evaluation of approaches to propositionalization. In *Proceedings of the 13th International Conference on Inductive Logic Programming (ILP-2003)*, pages 197–214, 2003.

Ondrej Kuželka and Filip Železný. Block-wise construction of tree-like relational features with monotone reducibility and redundancy. *Machine Learning*, 83(2): 163–192, 2011.

Ondřej Kuželka and Filip Železný. HiFi: Tractable propositionalization through hierarchical feature construction. In *Late Breaking Papers, the 18th International Conference on Inductive Logic Programming*, pages 69–74, 2008.

Nicolas Lachiche and Peter A. Flach. 1BC2: A true first-order Bayesian classifier. In *Proceedings of Inductive Logic Programming*, pages 133–148, 2003.

Nada Lavrač and Sašo Džeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, 1994.

Nada Lavrač and Peter Flach. An extended transformation approach to inductive logic programming. *ACM Transactions on Computational Logic*, 2(4):458–494, 2001.

Nada Lavrač, Sašo Džeroski, and Marko Grobelnik. Learning nonrecursive definitions of relations with LINUS. In *Proceedings of the 5th European Working Session on Learning (EWSL-91)*, pages 265–281, 1991.

Nada Lavrač, Blaž Škrlj, and Marko Robnik-Šikonja. Propositionalization and embeddings: two sides of the same coin. *Machine Learning*, 109:1465–1507, 2020.

Huma Lodhi. Deep Relational Machines. In *Proceedings of the International Conference on Neural Information Processing*, pages 212–219, 2013.

D. Michie, S. Muggleton, D. Page, and A. Srinivasan. To the international computing community: A new East-West challenge. Technical report, Oxford University Computing laboratory, Oxford,UK, 1994a. URL ftp://ftp.comlab.ox.ac.uk/pub/Packages/ILP/trains.tar.Z.

Donald Michie, Stephen Muggleton, David Page, and Ashwin Srinivasan. To the international computing community: A new East-West challenge. Technical report, Oxford University Computing laboratory, Oxford, UK, 1994b.

Stephen Muggleton. Inverse entailment and Progol. *New Generation Computing*, 13(3-4):245–286, 1995.

Stephen H. Muggleton, editor. *Inductive Logic Programming*. Academic Press Ltd., London, 1992.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12(Oct):2825–2830, 2011.

Matic Perovšek, Anže Vavpetič, Bojan Cestnik, and Nada Lavrač. A wordification approach to relational data mining. In *Proceedings of the International Conference on Discovery Science*, pages 141–154. Springer, 2013.

Matic Perovšek, Anze Vavpetič, Janez Kranjc, Bojan Cestnik, and Nada Lavrač. Wordification: Propositionalization by unfolding relational data into bags of words. *Expert Systems with Applications*, 42(17-18):6442–6456, 2015.

J. Ross Quinlan. Learning logical definitions from relations. *Machine Learning*, 5: 239–266, 1990.

Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. Why should I trust you?: Explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 1135–1144. ACM, 2016.

Ashwin Srinivasan. *The Aleph Manual*. University of Oxford, 2007. Online. Accessed 26 October 2020. URL: https://www.cs.ox.ac.uk/activities/programinduction/Aleph/.

Ashwin Srinivasan, Lovekesh Vig, and Michael Bain. Logical explanations for Deep Relational Machines using relevance information. *Journal of Machine Learning Research*, 20(130):1–47, 2019.

Stefan Van Der Walt, S Chris Colbert, and Gael Varoquaux. The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22, 2011.

Filip Železný and Nada Lavrač. Propositionalization-based relational subgroup discovery with RSD. *Machine Learning*, 62:33–63, 2006.