Type Extension Trees: a Unified Framework for Relational Feature Construction

Manfred Jaeger

Institut for Datalogi, Aalborg Universitet, Fredrik Bajers Vej 7E, DK-9220 Aalborg Ø
jaeger@cs.aau.dk

Abstract. We introduce type extension trees as a formal representation language for complex combinatorial features of relational data. Based on a very simple syntax this language provides a unified framework for expressing features as diverse as embedded subgraphs on the one hand, and marginal counts of attribute values on the other. We show by various examples how many existing relational data mining techniques can be expressed as the problem of constructing a type extension tree and a discriminant function.

1 Introduction

A key component of relational data mining methods is the construction of relevant features. Whereas in conventional ("propositional") learning settings the set of possible features is usually given by the available attributes, one has in relational learning the ability to construct new features by considering the relational neighborhood of an entity. Taking into consideration related entities and their attributes, one obtains a basically unlimited supply of potential features. The space of features actually explored by specific relational learning methods is often not very clearly defined and more or less only implicit in the learning algorithm and its candidate feature generation mechanisms.

In this paper we study relational features in their own right. We propose type extension trees (TETs) as a very simple, yet general and powerful representation language for relational features. We will illustrate the expressiveness and flexibility of type extension trees by showing how they can represent a great variety of different types of features used by different kinds of methods in graph mining, relational learning, and standard propositional learning.

This paper is mostly conceptual. Though intended to be eventually used in an implemented learning system, we introduce type extension trees in this paper mostly as a basis for an integrated view of existing data mining models and techniques. TETs provide a unified view along several dimensions: they accommodate relational features as considered in a variety of different disciplines, ranging from graph mining to statistical relational learning and probabilistic inductive logic programming (ILP); they provide a unified view of features of single entities, tuples of entities, or whole datasets; they support all levels of separation or integration of the feature construction and model induction components in a relational learning procedure.

Type extension trees, thus, provide a common ground from which fundamental aspects of different data mining techniques can be more clearly elucidated. A unifying conceptual framework is all but indispensable for a theoretical analysis of differences and similarities between different data mining techniques, and can help to translate results and techniques across different areas.

2 Type Extension Trees

In this section we introduce syntax and semantics of type extension trees. Our definitions are partly motivated by *type extension axioms*, which play a pivotal role in finite model theory [4]. Type extension axioms and various generalizations have been used to characterize the combinatorial structure of large random structures [5, 13, 8, 1]. In the finite model theory setting the generating distribution is given, and one is interested in the asymptotic properties of random structures. In relational learning the situation is reversed: one observes one randomly generated structure (the data), and would like to infer a model for the generating process. A language that has been found to express the characteristic features of data sampled from known distributions is also a good candidate for specifying those features of data that are important for reconstructing an unknown generating model.

We now turn to defining type extension trees, starting with definitions relating to relational structures, which serve as a general, abstract model for structured data.

R denotes a *relational signature* (or vocabulary), i.e. a set of relation symbols (of any arities). We typically use $r, s, t, \ldots$ to denote symbols from R, and $|r|$ to denote the arity of r. Throughout, we denote with u, v, w logical variables, and with $a, b, c, \ldots$ constants (representing specific entities in a domain). Tuples of variables and constants are denoted in bold font: $\boldsymbol{v}, \boldsymbol{a}$, etc. The length of a tuple $\boldsymbol{v}$ is denoted $|\boldsymbol{v}|$. An expression of the form $r(u, v), r(a, v), etc.$ is called an *atom*. If all arguments are constants, the atom is *ground*. A *literal* is an atom or a negated atom. A *type* is a conjunction of literals. Types are denoted $\tau, \sigma, \ldots$. We use $\top$ to denote an empty conjunction, which is to be interpreted as a tautology.

Definition 1. *A* relational R-structure $\mathcal{M}$ *consists of a domain M and an* interpretation function $I^{\mathcal{M}}$ *that assigns truth values to ground R-atoms, i.e.* $I^{\mathcal{M}}(r(\boldsymbol{a})) \in \{\text{true}, \text{false}\}$ *for all $r \in R$ and $\boldsymbol{a} \in M^{|r|}$. The* size *of $\mathcal{M}$ is the cardinality of M (in this paper always assumed to be finite).*

In logic programming terminology, an R-structure is just an interpretation. An R-structure with only unary and binary relations can be seen as a graph with colored nodes and edges. From a database perspective, an R-structure is a relational database with only boolean attributes. Using a boolean encoding of categorical attributes, one can represent databases with only categorical attributes as relational structures in the sense of definition 1.

Definition 2. *A* type extension tree (TET) *over R is a tree whose nodes are labeled with R-types, and whose edges are labeled with (possibly empty) sets of variables.*

A type extension tree is syntactically closely related to predicate logic formulas, with subtrees corresponding to sub-formulas, and edge labels corresponding to variables that are quantified over. In analogy to standard predicate logic definitions one defines the *free variables* of a TET, and the *substitution* of constants $\boldsymbol{a}$ for free variables $\boldsymbol{v}$.

TETs can be represented graphically as in figure 4 or equation (1). We also write $[\tau, [\boldsymbol{w}_1, T_1], \ldots, [\boldsymbol{w}_m, T_m]]$ for a TET whose root is labeled with τ, and which has m subtrees $T_1, \ldots, T_m$ reached by edges with labels $\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m$. We write $T(\boldsymbol{v})$ for a TET containing free variables $\boldsymbol{v}$, and $T(\boldsymbol{a})$ for the result of substituting $\boldsymbol{a}$ for $\boldsymbol{v}$ in T.

The semantics of TETs differs from semantics of most formal languages in that interpretations of TETs do not just return a truth value, but a complex combinatorial structure. To motivate the following definitions, consider the first-order sentence $\tau(\boldsymbol{a}) \rightarrow \exists w \sigma(\boldsymbol{a}, w)$. Interpreted over a structure $\mathcal{M}$ with $\boldsymbol{a} \subseteq M$ this sentence is either true or false. Furthermore, one can replace the existential quantifier $\exists$ with a counting quantifier like $\exists^{\geq 10}$ stipulating the existence of at least ten w with $\sigma(\boldsymbol{a}, w)$. The interpretation of the new sentence would still be either true or false. However, instead of specifying exact truth conditions for the quantifier in front of w, we can also define the interpretation of the formula as the number of $b \in M$ such that $\sigma(\boldsymbol{a}, b)$ is true. This is basically how we will interpret the TET $\tau(\boldsymbol{a}) \overset{w}{-} \sigma(\boldsymbol{a}, w)$. For more complex TETs the interpretations are not simply numbers, but more complex combinatorial specifications, which are introduced by the following two definitions.

Definition 3. *For a finite set D and $k \in \mathbb{N}$ define the set of all k-multisets over D as*

$$\text{multisets}(D, k) := \{f : D \rightarrow \{0, \ldots, k\} \mid \sum_{d \in D} f(d) = k\}$$

Concrete multisets are written in the notation $[d_1 \mapsto k_1, \ldots, d_q \mapsto k_q]$ (only listing the $d_i \in D$ with $f(d_i) = k_i > 0$).

Definition 4. *The* value space $\mathcal{V}_n(T)$ *of a TET T for structures of size n is inductively defined as follows:*

- *If T consists of a single node, then $\mathcal{V}_n(T) = \{\text{true}, \text{false}\}$.*
- *If $T = [\tau, [\boldsymbol{w}_1, T_1], \ldots, [\boldsymbol{w}_m, T_m]]$, then*

$$\mathcal{V}_n(T) = \{\text{true}, \text{false}\} \times \times_{i=1}^{m} \text{multisets}(\mathcal{V}_n(T_i), n^{|\boldsymbol{w}_i|})$$

Example 1. The value space of the TET $T(\boldsymbol{v}) = \tau(\boldsymbol{v}) \overset{w}{-} \sigma(\boldsymbol{v}, w)$ over a structure $\mathcal{M}$ of size n is $\mathcal{V}_n(T(\boldsymbol{v})) = \{true, false\} \times multisets(\{true, false\}, n)$. A tuple $\boldsymbol{a} \in \mathcal{M}^{|\boldsymbol{v}|}$ defines a value $V(T(\boldsymbol{a})) = (I, f) \in \mathcal{V}_n(T(\boldsymbol{v}))$, where $I \in \{true, false\}$ is the truth value of $\tau(\boldsymbol{a})$ in $\mathcal{M}$, and $f \in multisets(\{true, false\}, n)$ gives the counts

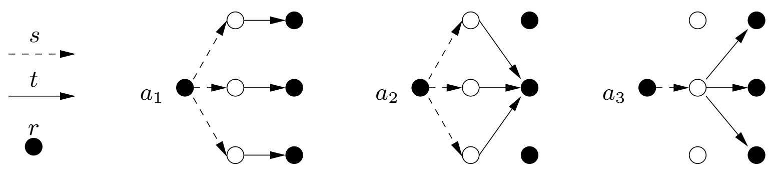


Fig. 1. Example 2

of elements $b \in M$ for which $\sigma(\boldsymbol{a}, b)$ is true, respectively false. The general definition of the value of a tuple $\boldsymbol{a}$ in a structure $\mathcal{M}$ is given in the following definition.

Definition 5. *Let $T(\boldsymbol{v})$ be a TET, $\mathcal{M}$ an R-structure of size n, and $\boldsymbol{a} \in M^{|\boldsymbol{v}|}$. The value of the ground TET $T(\boldsymbol{a})$ is defined as follows:*

– $V([\tau(\boldsymbol{a})]) = I^{\mathcal{M}}(\tau(\boldsymbol{a}))$
– $V([\tau(\boldsymbol{a}), [\boldsymbol{w}_1, T_1(\boldsymbol{a}, \boldsymbol{w}_1)], \ldots, [\boldsymbol{w}_m, T_m(\boldsymbol{a}, \boldsymbol{w}_m)]]) =$
$(I^{\mathcal{M}}(\tau(\boldsymbol{a})), f(\boldsymbol{a}, \boldsymbol{w}_1, T_1), \ldots, f(\boldsymbol{a}, \boldsymbol{w}_m, T_m))$, *where*

$$f(\boldsymbol{a}, \boldsymbol{w}_i, T_i) : \gamma \mapsto |\{\boldsymbol{b} \in M^{|\boldsymbol{w}_i|} \mid V(T_i(\boldsymbol{a}, \boldsymbol{b})) = \gamma\}| \qquad (\gamma \in \mathcal{V}_n(T_i))$$

Example 2. Figure 1 shows three different structures for a vocabulary containing two binary relations s, t, and one unary relation r. The relational neighborhoods of the entities a_1, a_2, a_3 in the three structures have somewhat similar, yet unique, properties. Consider the following three TETs:

$$T_1(v) : \top(v) \overset{u,w}{-} s(v,u), t(u,w), r(w)$$
$$T_2(v) : \top(v) \overset{u}{-} s(v,u) \overset{w}{-} t(u,w), r(w)$$
$$T_3(v) : \top(v) \overset{w}{-} r(w) \overset{u}{-} s(v,u)t(u,w)$$

The value space $\mathcal{V}_7(T_1(v))$ is $\{t, f\} \times$ *multisets*$(\{t, f\}, 7^2)$. If M is a domain of seven entities, and $a \in M$, then $V(T_1(a)) = (t, [t \mapsto l, f \mapsto 7^2 - l])$, where l is the number of distinct tuples $(b, c) \in M^2$ for which $s(a, b) \wedge t(b, c) \wedge r(c)$ is true. Thus, $T_1(a)$ just gives the counts of paths leading from a via one $s()$ and one $t()$ relation to an entity with attribute $r()$. This count is 3 for a_1, a_2, a_3, so that $V(T_1(a_1)) = V(T_1(a_2)) = V(T_1(a_3))$.

The value space $\mathcal{V}_n(T_2)$ is more complex. Values here not only encode the number of $s - t$-paths to an entity with attribute r, but also differentiate with regard to the number of different intermediate nodes on these paths. The precise values are (for legibility, multisets here written as column vectors):

$V(T_2(a_1)) = V(T_2(a_2)) =$ $\qquad\qquad$ $V(T_2(a_3)) =$

$$\left(t, \begin{bmatrix} \left(t, \begin{bmatrix} t \mapsto 1 \\ f \mapsto 6 \end{bmatrix}\right) \mapsto 3 \\ (f, [f \mapsto 7]) \mapsto 4 \end{bmatrix}\right) \qquad \left(t, \begin{bmatrix} \left(t, \begin{bmatrix} t \mapsto 3 \\ f \mapsto 4 \end{bmatrix}\right) \mapsto 1 \\ (f, [f \mapsto 7]) \mapsto 6 \end{bmatrix}\right)$$

Thus, T_2 distinguishes a_1 from a_3, but not from a_2. Another variation is provided by $T_3(v)$, which counts the number of distinct endpoints c (but not the number of distinct mid-points b): $V(T_3(a_1)) = V(T_3(a_3)) \neq V(T_3(a_2))$.

In the preceding example we found that type extension trees T_2, T_3 provided somewhat more information than T_1. This is formally captured in the following definition, which provides a refinement concept related to those used in ILP-based learning methods.

Definition 6. *Let $T(\boldsymbol{v}), T'(\boldsymbol{v})$ be type extension trees. T' is called a* refinement *of T if there exist functions*

$$\Gamma_n : \; \mathcal{V}_n(T') \rightarrow \mathcal{V}_n(T) \qquad (n \in \mathbb{N}),$$

such that for all structures $\mathcal{M}$ of size n, and all $\boldsymbol{a} \in M^{|\boldsymbol{v}|}$:

$$V(T'(\boldsymbol{a})) = \gamma \rightarrow V(T(\boldsymbol{a})) = \Gamma_n(\gamma).$$

Here we have defined refinement on a semantic basis. One can easily define several syntactic operations on TETs that produce refinements (e.g. adding a new subtree to an existing TET).

Values $V(T(\boldsymbol{a}))$ provide a very detailed quantitative picture of the "relational neighborhood" of tuple $\boldsymbol{a}$ in structure $\mathcal{M}$. For all but the simplest TETs, the full value will be too complex to handle by a model induction algorithm. The complex feature value $V(T(\boldsymbol{a}))$, therefore, may need to be reduced or summarized.

Definition 7. *A* discriminant function *for a TET T is a function*

$$d : \cup_{n \geq 1} \mathcal{V}_n(T) \rightarrow \mathbb{R}.$$

Discriminant functions can be employed in different ways. For example, a 0,1-valued discriminant function turns T into a boolean feature. A collection of TETs, each equipped with a boolean discriminant function, then defines a set of boolean features that can be used by standard propositional model induction algorithms. However, discriminant functions can also represent the final model itself. For example, a discriminant function on $T(v)$ with values in $\{1, 2, 3\}$ could represent a predictive model for a three-state class variable $c(v)$.

3 Examples

In this section we show how several quite distinct relational data mining tasks can be represented as the problem of finding a TET and a discriminant function.

3.1 Frequent subgraphs

A key task in graph-based data mining is finding frequent subgraphs (see e.g. [19]). Consider, for example, a relational alphabet $R = \{r(\cdot), t(\cdot, \cdot)\}$. Figure 2 shows a two node graph G over this vocabulary.

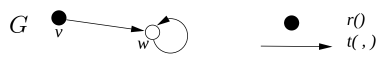


Fig. 2. A two node subgraph

Now consider the TET

$$T_G(v,w) = r(v) \overset{\emptyset}{-} \neg r(w) \overset{\emptyset}{-} \neg t(v,v) \overset{\emptyset}{-} t(w,w) \overset{\emptyset}{-} t(v,w) \overset{\emptyset}{-} \neg t(w,v) \qquad (1)$$

The value space $\mathcal{V}_n(T_G(v,w))$ is equivalent to the 6-fold product of $\{\mathit{true}, \mathit{false}\}$ (independent of n). For any two entities a, b, the value $V(T_G(a,b))$ determines the truth value of all the ground atoms $r(a), r(b), \ldots, t(b,a)$, and thus represents the subgraph induced by a, b.

Now let d_e be the discriminant function on $\mathcal{V}(T_G)$ that counts the number of *false* components in $V(T_G)$. For any a, b, then $d_e(V(T_G(a,b)))$ is the edit distance between the subgraph induced by a, b and G.

Suppose, now, we want to determine whether our data $\mathcal{M}$ contains at least k occurrences of 2-node subgraphs whose edit distance to G is at most l. This becomes an evaluation of a TET-discriminant function by defining on the TET

$$T'_G = \top \overset{v,w}{-} T_G(v,w) \qquad (2)$$

the discriminant function $d_{k,l}$ defined by

$$d_{k,l}(V(\top), f((v,w), T_G(v,w)) = \begin{cases} 1 \text{ if} \displaystyle\sum_{\gamma \in \mathcal{V}(T_G(v,w)) : d_e(\gamma) \leq l} f((v,w), T_G(v,w))(\gamma) & > k \\ 0 \text{ else} \end{cases}$$

Finally, the problem of finding all subgraphs G' that are "approximately frequent" (in the sense that graphs with edit distance to G' at most l occur at least k times) now becomes the problem of finding all TETs $T_{G'}$ of the structure $\top \overset{\boldsymbol{v}}{-} T_{G'}(\boldsymbol{v})$ with $d_{k,l}(V(T_{G'})) = 1$.

This example shows how TETs and discriminant functions capture operations at all stages of feature construction and model induction: a TET of the form (1) together with the discriminant function d_e is just a boolean feature that could be used by any model induction algorithm. However, full model induction tasks can also be expressed as a search for TETs and discriminant functions.

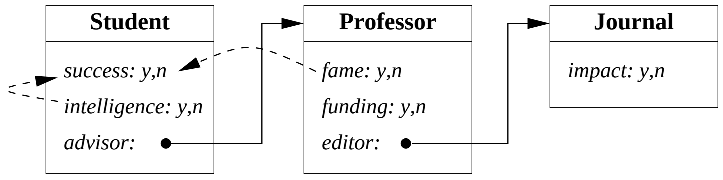


Fig. 3. A simple PRM

3.2 Probabilistic Relational Models

In this section we consider probabilistic relational models (PRMs) in the sense of [17, 6]. PRMs in their simplest form (there exist various extensions) are a probabilistic model for attribute values in a relational database. Figure 3 illustrates a simple example PRM based on [17]. It consists of a database schema consisting of three tables 'student', 'professor' and 'journal'. All tables have one or two boolean attributes. The student table also has a *reference attribute advisor*, which points to entries in the professor table, and the professor table has a reference attribute *editor* pointing to the journal table. The dashed arrows in figure 3 indicates that the PRM provides a probabilistic model for the *success* attribute in the student table, and that according to this model the probability for student.*success* values depends on the values of the *intelligence* attribute for the given student, and on the value of the *fame* attribute of the student's advisor. The complete specification of the PRM furthermore will contain the quantitative specification of the conditional distribution for student.*success*, which is not shown here.

Generally, the probabilities for attributes can be defined conditional on attribute values at any other table entries that can be reached by following a sequence of references. References can also be followed in inverse order: for example, by first following the *advisor* reference from a student entry s, one finds an entry p in the professor table; by then following the inverse of the *advisor* reference from p, one finds all entries in the student table that have the same advisor as s. Similarly, the student.*success* attribute could also be defined conditional on the *impact* of journals for which the student's advisor is an editor. Friedman et al. [6] call any such sequence of references a *slot chain*.

Slot chains essentially define the features used in a probabilistic relational model. When the slot chain also contains one-to-many relationships, then aggregation operators will also be needed to define a feature. Features definable by slot chains can be represented as linear TETs:

$$T_1(v) : \top \overset{w}{-} \mathit{advisor}(v, w) \overset{\emptyset}{-} \mathit{fame}(w) \tag{3}$$

$$T_2(v) : \top \overset{w}{-} \mathit{advisor}(v, w) \overset{u}{-} \mathit{advisor}(u, w), u \neq w \overset{\emptyset}{-} \mathit{success}(u) \tag{4}$$

$T_1(v)$ counts how many famous and how many non-famous advisors entity v has. $T_2(v)$ counts how many successful and non-successful entities u exist that have the same advisor as v.

3.3 Relational Bayesian Networks

Relational Bayesian networks (RBNs) [7, 9] are related both to PRMs and to probabilistic modeling languages based on logic programming (e.g. [18, 11]). RBNs also specify probability distributions over probabilistic relations on a given domain. The representation language is quite different, however: an RBN representation is given by the specification of one *probability formula* for each probabilistic relation. A simple RBN encoding for the domain described in the previous section could be given by assigning to the probabilistic relation $\mathit{success}(v)$ the probability formula

$$F(v) = \mathit{noisy\text{-}or}\{(\mathit{fame}(w) : 0.8, 0.5) \mid w : \mathit{advisor}(v, w)\}.$$

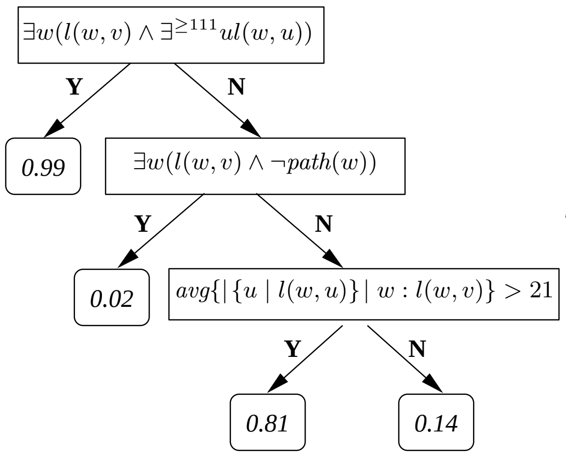
This formula can be read as a procedural computation rule for computing the probability of a ground atom $\mathit{success}(a)$ as follows: determine all objects w for which $\mathit{advisor}(a, w)$ is true (which may or may not be exactly one); each such w contributes a value of 0.8 if $\mathit{fame}(w)$ is true, and a value of 0.5 if $\mathit{fame}(w)$ is false to a multiset of probabilities; finally, compute a single probability value by combining all the probabilities in the multiset with a *noisy-or*.

The reader is referred to [7, 9] for details on syntax and semantics of probability formulas. The preceding small example should be sufficient, however, to illustrate the connection between RBNs and TETs: for the evaluation of the given probability formula for $\mathit{success}(a)$ it is sufficient to know the value of $T_1(a)$ with T_1 as in (3). More refined probability formulas could be constructed that define the probability of $\mathit{success}(a)$ also dependent on the feature $T_2(a)$. Generally, one can show that for each probability formula $F(\boldsymbol{v})$ one can (automatically) construct a TET $T(\boldsymbol{v})$, such that the evaluation of the probability formula for some objects $\boldsymbol{a}$ only depends on the value $V(T(\boldsymbol{a}))$. On the other hand, the probability formula computes a probability value for each possible value of the TET. Thus, a probability formula is both an (implicit) definition of a TET, and a specification of a discriminant function on that TET.

3.4 Relational Decision Trees

Several approaches exist for adapting decision trees to relational settings [2, 12, 15]. Internal nodes in such a decision tree are labeled by relational features, which can be expressed e.g. in a logical [2, 15] or graphical [12] notation.

Figure 4 shows a relational decision tree described in [15]. This is a decision tree for a domain of web-pages with attributes like *isStudent*,*isFaculty*,*isCourse* describing the nature of a web page (student, faculty, course homepage etc.). Another attribute, *path*, describes whether or not the URL of a page contains a directory path (i.e. pages for which *path*=*false* are top-level pages in a domain).



$\exists w(l(w,v) \wedge \exists^{\geq 111} u l(w,u))$
Y
N
0.99
$\exists w(l(w,v) \wedge \neg path(w))$
Y
N
0.02
$avg\{|\{u \mid l(w,u)\}| \mid w : l(w,v)\} > 21$
Y
N
0.81
0.14

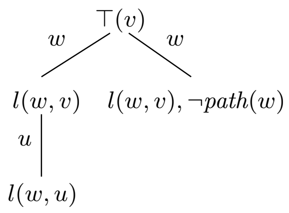


Fig. 4. Relational Decision Tree and Feature TET

The decision tree of figure 4 estimates the probability that a web-page v belongs to the *isStudent* class. For this it tests (at most) three different relational features: at the root node it is tested whether the page v is linked from another page w which has at least 111 outgoing links (indicating, perhaps, a student directory). The second node tests the feature whether v is linked from a page without a directory path in the URL. The third node tests whether the average number of outgoing links from pages w that also contain a link to v is at least 21.

The TET shown on the right of figure 4 represents the combinatorial properties that determine the truth value of the three features tested in the decision tree: for each web page v, the truth value of a feature is a function of $V(T(v))$ (where, in fact, for the truth values of the first and third feature only the value of the left branch is relevant, and for the truth value of the second feature only the right branch is relevant). Furthermore, the probability estimates computed by the decision tree can be defined by a discriminant function on T.

3.5 From Features to Statistics

In standard "propositional" learning settings there is a clear distinction between a *data item* and a *data set*. The latter is a collection of the former, and, moreover, the data items are typically assumed to be independent samples from some common underlying distribution (iid). The distinction between data item and data set gives rise to the distinction between a feature and a *statistic*: the former describes a property of an individual data item, the latter a property of a data set as a whole.

It is well-known that this "iid" model of data is often inappropriate in the context of relational data (see e.g. [10]). Specifically, one can not regard the entities in a relational structure as independent data items.

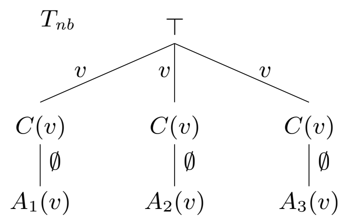


Fig. 5. Naive Bayes Sufficient Statistics

Seeing that in relational data there is no clear dividing line between data items and data sets, there can also not be a clear distinction between a feature and a statistic. Type extension trees provide a clear perspective on both the common nature and the remaining distinction between feature and statistic: a statistic is basically a feature expressed by a TET without free variables. Such a TET represents a global property of a relational structure, without reference to any particular entities in the structure

The following example illustrates the use of TETs to represent global statistics, and how standard propositional data can be treated as a special case of relational structures.

Example 3. Assume a relational signature R containing four unary relation symbols C, A_1, A_2, A_3. A relational structure for R just consists of a collection of entities, and the relations can be read as attributes in the conventional way, i.e. an R-structure can also be seen as a standard propositional dataset.

Consider the TET T_{nb} of figure 5. This is a TET without free variables. Its values represent the 2-way marginal counts of attribute pairs C, A_i $(i = 1, 2, 3)$. Thus, the value of T_{nb} for a given R-structure is just the minimal sufficient statistic for learning a naive Bayes classifier for C given the A_i. Compare this to

$$T_{sat} = \top \overset{v}{-} C(v) \overset{\emptyset}{-} A_1(v) \overset{\emptyset}{-} A_2(v) \overset{\emptyset}{-} A_3(v)$$

Values of T_{sat} are the full joint counts of $C, A_1, \ldots, A_3$, and, thus, are sufficient for learning a saturated model, i.e. without any constraints on the joint distribution of the $C, A_1, \ldots, A_3$.

4 Discussion and Related Work

In the previous sections we have introduced type extension trees as a language for specifying relational features. We have seen that the language is very expressive, and can represent different types of features that have been used within a wide spectrum of different relational learning tasks.

One should compare TETs to two different classes of alternatives. First, we may consider logics, database query or programming languages, which are not specifically designed to specify relational features, but can be employed for that purpose. The advantage of TETs over existing predicate logics is the former's ability to represent quantitative, combinatorial properties, whereas the latter only define Boolean features. Database query languages like SQL can also retrieve quite complex combinatorial information about the relational neighborhood of an entity. There are several advantages of the TET language over SQL, however: most importantly, the syntax of SQL is much more complex than the syntax of TETs, and does not possess a similarly clearly defined semantics. For example, it is not clear how to distinguish SQL queries that refer to properties of single entities, from queries that refer to entity pairs, entity triples, etc. In TETs this distinction is clearly made by the number of free variables.

Second, TETs need to be compared to specific feature specification languages used in existing learning frameworks. Such specification languages have been defined in terms of e.g. fragments of first-order logic [3], graph fragments [14], and aggregation operators [16]. Most similar in spirit to TETs are perhaps the selection graphs of [12]. However, the latter are much more limited in only being able to express boolean features of single entities. A key advantage of TETs over selection graphs and many other feature languages is their clear parsimonious syntax, coupled with a precise formal semantics.

Apart from being an alternative to existing feature languages, TETs also provide a solid basis for theoretical analyses of their characteristic properties. For example, identifying different feature languages with specific sub-classes of TETs provides a basis for analyzing more clearly what kind of information the different languages are able to extract.

5 Conclusion

We have introduced TETs as an expressive and principled framework for describing features in relational domains. TETs provide a uniform language for specifying features of single entities, tuples of entities, or a relational dataset as a whole. By combining TETs with the notion of discriminant functions one obtains a coherent, integrated view of feature construction and model induction that can capture diverse data mining tasks such as graph mining and statistical relational learning.

So far, we have introduced TETs mostly as a conceptual tool. One prerequisite for using TETs in practice is to complement the language of TETs with a formal language for defining discriminant functions. Based on the TET tree structure it is quite easy to inductively define such languages in a way that is similarly parsimonious and semantically clear as the TET language itself. One example of such a language already exists in the language of probability formulas (section 3.3).

Here we have introduced TETs only for boolean data. However, all our basic definitions can be directly generalized to non-binary data by only relaxing the definition of literals to also include equational literals $r(v) = A$ (A a possible value of categorical attribute r), or $s(v) \geq q$ ($q \in \mathbb{R}$ a threshold value for numerical attribute s).

References

1. A. Blass and Y. Gurevich. Choiceless polynomial time computation and the zero-one law. In *Proc. of CSL 2000*, pages 18–40, 2000.
2. H. Blockeel and L. de Raedt. Top-down induction of first-order logical decision trees. *Artificial Intelligence*, (101):285–297, 1998.
3. C. Cumby and D. Roth. Feature extraction languages for propositionalized relational learning. In *Proc. of the IJCAI-2003 workshop on learning statistical models from relational data*, 2003. http://kdl.cs.umass.edu/srl2003.
4. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer Verlag, 1995.
5. Ronald Fagin. Probabilities on finite models. *Journal of Symbolic Logic*, 41(1):50–58, 1976.
6. N. Friedman, Lise Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proceedings of the 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999.
7. M. Jaeger. Relational bayesian networks. In Dan Geiger and Prakash Pundalik Shenoy, editors, *Proceedings of the 13th Conference of Uncertainty in Artificial Intelligence (UAI-13)*, pages 266–273, Providence, USA, 1997. Morgan Kaufmann.
8. M. Jaeger. Convergence results for relational Bayesian networks. In *Proceedings of the 13th Annual IEEE Symposium on Logic in Computer Science (LICS-98)*, pages 44–55, IEEE Computer Society Press.
9. M. Jaeger. Complex probabilistic modeling with recursive relational Bayesian networks. *Annals of Mathematics and Artificial Intelligence*, 32:179–220, 2001.
10. D. Jensen and J. Neville. Linkage and autocorrelation cause feature selection bias in relational learning. In *Proceedings of the 19th International Conference on Machine Learning (ICML–2002)*, pages 259–266, 2002.
11. K. Kersting and L. De Raedt. Towards combining inductive logic programming and bayesian networks. In *Proceedings of the Eleventh International Conference on Inductive Logic Programming (ILP-2001)*, Springer Lecture Notes in AI 2157, 2001.
12. A. J. Knobbe, A. Siebes, and D. van der Wallen. Multi-relational decision tree induction. In *Proceedings of PKDD-99*, pages 378–383, 1999.
13. Ph. G. Kolaitis and M.Y.Vardi. 0-1 laws and decision problems for fragments of second-order logic. *Information and Computation*, 87:302–338, 1990.
14. S. Kramer and L. de Raedt. Feature construction with version spaces for biochemical applications. In *Proc. of ICML-01*, 2001.
15. J. Neville, D. Jensen, L. Friedland, and M. Hay. Learning relational probability trees. In *Proceedings of SIGKDDD'03*, 2003.
16. C. Perlich and F. Provost. Aggregation-based featrue invention and relational concept classes. In *Proc. of SIGKDD'03*, 2003.
17. A. Pfeffer. *Probabilistic Reasoning for Complex Systems.* PhD thesis, Stanford University, 2000.
18. T. Sato. A statistical learning method for logic programs with distribution semantics. In *Proceedings of the 12th International Conference on Logic Programming (ICLP'95)*, pages 715–729, 1995.
19. Takashi Washio and Hiroshi Motoda. State of the art of graph-based data mining. *SIGKDD Explor. Newsl.*, 5(1):59–68, 2003.