Feature Discovery with Type Extension Trees

Paolo Frasconi[2] and Manfred Jaeger[1] and Andrea Passerini[2]

[1] Department for Computer Science, Aalborg University, Denmark
[2] Dipartimento di Sistemi e Informatica, Universitá degli Studi di Firenze, Italy

Abstract. We are interested in learning complex combinatorial features from relational data. We rely on an expressive and general representation language whose semantics allows us to express many features that have been used in different statistical relational learning settings. To avoid expensive exhaustive search over the space of relational features, we introduce a heuristic search algorithm guided by a generalized relational notion of information gain and a discriminant function. The algorithm succesfully finds interesting and interpretable features on artificial and real-world relational learning problems.

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 large supply of potential features.

To illustrate the situation, consider Figure 1, which gives a graphical representation of an imaginary fragment of a movie database. Unfilled circles represent movies, shaded circles represent actors, arrows represent the binary relation *cast*, and stars identify movies that have won the best picture award. Suppose, now, that for movies m_1, m_2 it is not yet known whether they will win best picture award, and we want to predict this. For this prediction, awards won by movies with the same actors as m_1, m_2 may provide relevant information.

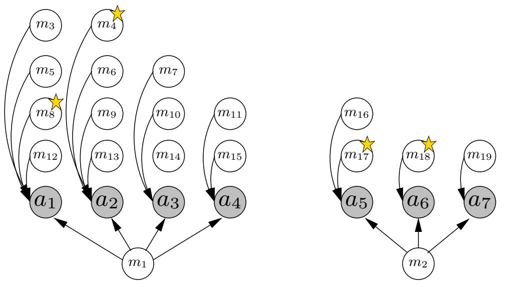


Fig. 1. Movie data fragment

To express features in terms of properties of related objects, many different frameworks have been proposed. Some approaches are based on Boolean features corresponding to logical conditions like "there exists an actor in the cast who has also played in a movie that has won the best picture award." Formal languages for expressing such features can be based on predicate logic (as in inductive logic programming), or use graphical representations [9].

Movies m_1, m_2 in Figure 1 can not be distinguished with such simple qualitative features, since both satisfy the same basic logical properties. Nevertheless, considering certain quantitative aspects of the relational neighborhoods of m_1, m_2, one might perhaps expect that m_2 has a higher chance of receiving an award than m_1: the two awards associated with m_1 come from cast members who have appeared in a relatively large number of movies, indicating that these are perhaps actors playing minor roles in many different movies, some of which happened to be successful. The awards associated with m_2, on the other hand, come from actors appearing in fewer movies, indicating that these could be major actors having leading roles in m_2.

One well established approach to express quantitative features is the use of *slot chains* and *aggregation*, which translate into database queries [5, 13, 14]. An example is the feature that counts the number of award-winning movies played by all cast members. This feature evaluates to 2 for both m_1 and m_2, and thus does not express the distinction described above. This distinction could be expressed by using proportion instead of count as a form of aggregation, yielding the feature that measures the proportion of award-winning movies among those played by cast members. It evaluates to 2/13 and 2/4 for m_1 and m_2, respectively, and thus makes the desired distinction. However, it is doubtful that this numeric feature captures exactly the most relevant quantitative information for making a prediction about m_1, m_2. Perhaps the count of actors with a proportion of $\geq 50\%$ award winning movies among the movies they participated in is the more relevant feature (here evaluating to 0 and 2, respectively, assuming that the movie to be classified is not considered in the count). Nested aggregates like this have been used in conjunction with first-order decision trees [1].

This very simple example illustrates three interesting aspects of relational features. First, the search space of possible features constructible by different combinations of aggregation, even for a single given slot chain, is quite big; second, each feature that we can construct in this way will probably only provide an approximation to the most relevant quantitative information for the prediction of the class label, and third, all these features are really summary statistics of a single, underlying more complex and fundamental feature: the complete specification of how many actors in the movie to be classified acted in how many other award winning and non award winning movies. The values of this *count of count* feature for m_1, m_2 are given by:

$$m_1 : \left\{ \begin{array}{l} \left\{ \begin{array}{r} a : 1 \\ \neg a : 3 \end{array} \right\} : 2 \\ \left\{ \begin{array}{r} a : 0 \\ \neg a : 3 \end{array} \right\} : 1 \\ \left\{ \begin{array}{r} a : 0 \\ \neg a : 2 \end{array} \right\} : 1 \end{array} \right\} \quad m_2 : \left\{ \begin{array}{l} \left\{ \begin{array}{r} a : 1 \\ \neg a : 1 \end{array} \right\} : 1 \\ \left\{ \begin{array}{r} a : 1 \\ \neg a : 0 \end{array} \right\} : 1 \\ \left\{ \begin{array}{r} a : 0 \\ \neg a : 1 \end{array} \right\} : 1 \end{array} \right\} \tag{1}$$

The value for m_1 shows that in the cast of m_1 there are two actors that acted in 1 award winning, and 3 non award winning movies. The formatting of the feature value emphasizes the hierarchical, combinatorial structure of these count of count values (which can also be nested deeper into count of count of count values, etc.). Note that count of count features in our sense are quite different from nested aggregates in the sense of [1]: unlike the latter, our counts of counts do not aggregate a multiset of values into a single number at each level of nesting.

In this paper, we study relational learning on the basis of complex count of count features. To this end we first review syntax and semantics of *Type extension trees (TETs)*, which are a representation language for such features first introduced in [7]. It has been shown in [7] that the TET language is very powerful, and many classes of features that have been used in relational learning can be derived from TET features.

However, the use of TET features in actual learning tasks has not been studied so far. In this paper, we introduce an algorithm for learning the structure of a TET from data in the supervised learning setting. A key component of this algorithm is a special kind of discriminant function that maps TET values into real numbers, and can be used as a simple but computationally efficient classifier. The TET learning algorithm uses heuristic search based on two types of evaluation heuristics: a generalized information gain measure for relational data, and a wrapper evaluation score based on empirical error of the discriminant function. Our experimental evaluation shows that we can learn TETs representing relevant and interpretable features.

2 TET Basics

In this section, we review the basic syntax and semantics definitions of TETs [7]. To simplify the definitions, we will assume fully Boolean domains, i.e. all attributes and relations are Boolean. All basic concepts can be quite easily generalized to multi-valued attributes and relations. In a Boolean domain, data can be viewed as a model in the sense of the following definition.

Definition 1. *Let R be a* relational signature *i.e. a set of relation symbols of different arities. A (finite) model for R, $\mathcal{M} = (M, I)$ consists of a finite domain M, and an interpretation function $I : r(\boldsymbol{a}) \rightarrow \{\text{true}, \text{false}\}$ defined for all* ground atoms $r(\boldsymbol{a})$ *constructible from relations $r \in R$ and arguments $\boldsymbol{a} \in M^{\text{arity}(r)}$.*

In logic programming terminology, $\mathcal{M}$ is a Herbrand interpretation for the vocabulary consisting of R and constant symbols for the elements of M. For convenience we may

$$\begin{tikzcd}
 & & & \substack{\textit{award}(u)\\(0.5)} \\
\substack{\textit{movie}(v)\\(0.5)} \arrow[r, "w"] & \substack{\textit{cast}(v,w)\\(0.375)} \arrow[r, "u"] & \substack{\textit{cast}(u,w), u \neq v\\(0.222)} \arrow[ur] \arrow[dr] & \\
 & & & \substack{\neg\textit{award}(u)\\(0.181)}
\end{tikzcd}$$

Fig. 2. A simple TET

assume that the domain M is partitioned into objects of different *types*, that arguments of relations are typed, and that I is only defined for ground atoms with arguments of appropriate types. We now proceed to define the formal syntax for expressing count of count features. The following definition is illustrated by Figure 2, which shows the TET defining the movie feature whose values for m_1, m_2 are shown in Eq. 1. The numbers in parantheses included in Figure 2 will be explained in Example 2 below.

Definition 2. *An* R-literal *is a (negated) atom* $r(\boldsymbol{v})$ *(*$r \in R \cup \{=\}$*,* $\boldsymbol{v}$ *a tuple of (possibly typed) variable symbols). An* R*-literal may not contain any constant symbols for elements* $a \in M$ *as arguments. We also allow the special literal* $\top(\boldsymbol{v})$*, which always evaluates to* true*. A* type *is a conjunction of* R*-literals.*

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.*

Note that the type of an object corresponds to a literal consisting of a single unary relation (e.g. *movie*(v)). Definition 2 generalizes this to types of tuples of objects, and to types which are expressed via a conjunction of literals. Labeled edges in a TET are related to quantifiers in predicate logic: like a quantifier, a labeled edge *binds* all occurrences of the variables associated with the edge in the subtree rooted at this edge. The *free variables* of a TET are all variables not bound by an edge label. The TET of Figure 2 has the single free variable v. We write $T(\boldsymbol{v})$ to denote a TET whose free variables are among the variables $\boldsymbol{v}$ (but does not necessarily contain all of them). We write

$$T(\boldsymbol{v}) = [\tau(\boldsymbol{v}), (\boldsymbol{w}_1, T_1(\boldsymbol{v}, \boldsymbol{w}_1)), \ldots, (\boldsymbol{w}_m, T_m(\boldsymbol{v}, \boldsymbol{w}_m))]$$

to denote a TET with a root labeled with $\tau(\boldsymbol{v})$, and m sub-trees $T_1(\boldsymbol{v}, \boldsymbol{w}_i)$ reached by edges labeled with variables $\boldsymbol{w}_i$ (possibly empty).

A TET $T(\boldsymbol{v})$ with free variables $\boldsymbol{v} = v_1, \ldots, v_k$ defines a *feature* for k-tuples of domain elements: for any model $\mathcal{M}$, and any $\boldsymbol{a} \in M^k$ the TET defines a *feature value* $V(T(\boldsymbol{a}))$. Eq 1 shows $V(T(m_1))$ and $V(T(m_2))$ for the TET T in Figure 2. We give the general definition of TET semantics in two steps: first we define the value space of nested counts associated with a given TET, and then the actual mapping $\boldsymbol{a} \mapsto V(T(\boldsymbol{a}))$.

Definition 3. *For any set* A *we denote with* multisets(A) *the set of all multisets over* A*. We denote with* $\{a_1 : k_1, \ldots, a_n : k_n\}$ *a multiset that contains* k_i *copies of* a_i*. The* pruned value space $\mathcal{V}(T)$ *of a TET* T *is inductively defined as follows:*

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

$$\mathcal{V}(T) = \{\text{false}\} \cup \times_{i=1}^{m} \text{multisets}(\mathcal{V}(T_i)))$$

We call the $\mathcal{V}(T)$ of Definition 3 the pruned value space, because with these values we will only count objects that satisfy the conditions in the nodes. Full TET values, as originally defined in [7], treat *true* and *false* on an equal basis, and count both satisfying and non-satisfying objects.

Definition 4. *Let* $\mathcal{M} = (M, I)$ *be a model,* $T(v_1, \ldots, v_k)$ *a TET, and* $\boldsymbol{a} \in M^k$*. The value* $V(T(\boldsymbol{a})) \in \mathcal{V}(T)$ *is defined as follows:*

(i) If $T(\boldsymbol{v}) = [\tau(\boldsymbol{v})]$ consists of a single node, then $V(T(\boldsymbol{a})) := I(\tau(\boldsymbol{a}))$.
(ii) Otherwise:
(a) If $I(\tau(\boldsymbol{a})) = \text{false}$ then $V(T(\boldsymbol{a})) = \text{false}$.
(b) If $I(\tau(\boldsymbol{a})) = \text{true}$ then

$$V(T(\boldsymbol{a})) = (\mu(\boldsymbol{a}, \boldsymbol{w}_1, T_1), \ldots, \mu(\boldsymbol{a}, \boldsymbol{w}_m, T_m)),$$

with $\mu(\boldsymbol{a}, \boldsymbol{w}_i, T_i) \in \text{multisets}(\mathcal{V}(T_i))$ given by

$$\{\gamma :| \{\boldsymbol{b} \in M \mid V(T_i(\boldsymbol{a}, \boldsymbol{b})) = \gamma\} \mid \mid \gamma \in \mathcal{V}(T_i)\}.$$

Example 1. Eq. 1 shows $V(T(m_1))$ and $V(T(m_2))$. To explain the compuations of $V(T(m_1))$, we introduce abbreviations for sub-TETs of T:

$$\begin{aligned} T^{(1)}(u, w) &:= [\textit{cast}(u, w), (\emptyset, [\textit{award}(u)]), (\emptyset, [\neg \textit{award}(u)])] \\ T^{(2)}(v, w) &:= [\textit{cast}(v, w), (u, T^{(1)}(u, w))] \end{aligned}$$

Since $I(\textit{movie}(m_1)) = \textit{true}$, the computation of $V(T(m_1))$ follows case ii.b of Definition 4. That means we have to count for each $\gamma \in \mathcal{V}(T^{(2)})$ the number of actors a for which $V(T^{(2)}(m_1, a)) = \gamma$. We do this by computing for each of the four actors $a_1, \ldots, a_4$ the value $V(T^{(2)}(m_1, a_i))$. Consider a_1. Since $I(\textit{cast}(m_1, a_1)) = \textit{true}$, we again reach case ii.b, and now have to compute for each of the 13 movies $m' \neq m_1$ the value $V^{(1)}(m', a_1)$. For 9 out of the 13 movies this is just the *false* value (ii.a), because $I(\textit{cast}(m', a_1)) = \textit{false}$. For 3 movies (the ones without award), we obtain the value $(\{\textit{false} : 1\}, \{\textit{true} : 1\})$ (the pair of values $(V([\textit{award}(m')]), V([\neg \textit{award}(m')])))$, and for the award winning movie the value $(\{\textit{true} : 1\}, \{\textit{false} : 1\})$. Collecting these counts, we get

$$V(T^{(2)}(m_1, a_1)) = \left\{ \begin{array}{r} \textit{false} : 9 \\ (\{\textit{false} : 1\}, \{\textit{true} : 1\}) : 3 \\ (\{\textit{true} : 1\}, \{\textit{false} : 1\}) : 1 \end{array} \right\}$$

This has been simplified to $\{a : 1, \neg a : 3\}$ in Equation 1. The same value is obtained for $V(T^{(2)}(m_1, a_2))$, giving a count of 2 for this value in $V(T(m_1))$.

A TET $T(\boldsymbol{v})$ represents a feature of object tuples $\boldsymbol{a}$. Several features might be expressed by TETs $T_1(\boldsymbol{v}), \ldots, T_m(\boldsymbol{v})$. Such a collection of features can always be combined into the single TET $[\top(\boldsymbol{v}), (\emptyset, T_1(\boldsymbol{v})), \ldots (\emptyset, T_m(\boldsymbol{v}))]$, which exactly provides the combined information of the T_i. In the following we will therefore focus on the scenario where a single TET represents all relevant features.

3 TET Discriminant Function

A TET alone only defines a feature of objects in a relational domain. TET-defined features can be incorporated in many ways into existing types of predictive or descriptive models. For example, one can define distance or kernel functions on TET value spaces $\mathcal{V}(T)$, thereby making TET features useable for standard clustering techniques, or SVM classifiers. In this section we briefly describe how to build a predictive model on a TET feature using simple *discriminant functions* on TET values, i.e. functions of the form

$$d : \mathcal{V}(T) \rightarrow \mathbb{R}.$$

To obtain a binary classification model (throughout we assume that class labels are $\{+,-\}$), we learn two discriminant functions d_+, d_-, and a threshold value t, and assign class label $+$ to a tuple $\boldsymbol{a}$ iff

$$d_+(V(T(\boldsymbol{a}))/d_-(V(T(\boldsymbol{a})) > t. \tag{2}$$

We now introduce one simple type of TET-discriminant function. The motivation for the particular form of discriminant function we propose is twofold: first, for a given TET, our functions are efficient to learn and evaluate. Since we use the discriminant function in TET learning within a wrapper evaluation routine for candidate TETs, efficiency is an important issue. Second, in spite of their simplicity, one can show that "propositional TETs" (TETs with only unary relations and no edge labels) together with these discriminant functions provide a uniform generalization of the classic decision tree and Naive Bayes models. Due to space limitations we have to defer a full discussion of the relationship between our discriminant functions and other predictive models to a full paper.

Definition 5. *Let T be a TET. A* weight assignment $\boldsymbol{\beta}$ *for T assigns a nonnegative real number to all nodes of T. A weight assignment can be written as $(\beta^r, \boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_m)$, where β^r is the weight assigned to the root, and $\boldsymbol{\beta}_i$ is the weight assignment to the ith sub-tree.*

For a TET T with node weights $\boldsymbol{\beta}$ we define the discriminant function $d^{\boldsymbol{\beta}}$ as follows. Let $\gamma \in \mathcal{V}(T)$:

– *If $\gamma =$ false define $d^{\boldsymbol{\beta}}(\gamma) := 0$.*
– *If $\gamma =$ true then $T = [\tau(\boldsymbol{v})]$ consists of a single node, and $\boldsymbol{\beta} = (\beta^r)$. Define $d^{\boldsymbol{\beta}}(\gamma) := \beta^r$.*
– *If $\gamma = (\mu_1, \ldots, \mu_m)$, $\mu_i \in$ multisets$(\mathcal{V}(T_i))$, define*

$$d^{\boldsymbol{\beta}}(\gamma) := \beta^r \cdot \prod_{i=1}^{m} \prod_{\gamma' \in \mu_i, \gamma' \neq \text{false}} \frac{1}{\beta^r} d^{\boldsymbol{\beta}_i}(\gamma').$$

Example 2. The numbers shown in Figure 2 are a weight assignment. These are the actual weights defining the discriminant function d_+ we would learn using the method described in Section 4 for this TET, assuming that m_1 and m_2 of Figure 1 are given as a negative and positive example, respectively.

Now consider the value $\gamma_1 = V(T(m_1))$ as shown on the left in Eq. 1 and explained in Example 1. The (simplified) inner values $\{a : j, \neg a : k\} \in \mathcal{V}(T^{(2)})$ have the discriminant value

$$\begin{aligned} d(a : j, \neg a : k) &= 0.375 \cdot \left(\tfrac{0.222}{0.375}\tfrac{0.5}{0.222}\right)^j \cdot \left(\tfrac{0.222}{0.375}\tfrac{0.181}{0.222}\right)^k \\ &= 0.375 \cdot \left(\tfrac{0.5}{0.375}\right)^j \cdot \left(\tfrac{0.181}{0.375}\right)^k . \end{aligned}$$

This leads to

$$\begin{aligned} d(\gamma_1) &= 0.5 \cdot \left(\tfrac{d(a:1,\neg a:3))}{0.5}\right)^2 \left(\tfrac{d(a:0,\neg a:3))}{0.5}\right) \left(\tfrac{d((a:0,\neg a:2))}{0.5}\right) \\ &= 0.5 \cdot \left(\tfrac{0.375}{0.5}\right)^4 \cdot \left(\tfrac{0.5}{0.375}\right)^2 \cdot \left(\tfrac{0.181}{0.375}\right)^{11} . \end{aligned}$$

Similarly for $\gamma_2 = V(T(m_2))$:

$$d(\gamma_2) = 0.5 \cdot \left(\tfrac{0.375}{0.5}\right)^3 \cdot \left(\tfrac{0.5}{0.375}\right)^2 \cdot \left(\tfrac{0.181}{0.375}\right)^2 .$$

Intuitively, we can interpret this discriminant function as follows: because in the training data the positive example had a smaller cast than the negative example, the discriminant function multiplies the marginal probability of the positive class (0.5) for each actor in the cast with a penalty term $\frac{0.375}{0.5} < 1$. Furthermore, since in the negative example the actors in the cast appeared in a larger number of movies than in the positive example, another penalty factor $\frac{0.222}{0.375} < 1$ is added for every associated movie of any actor in the cast. Finally, a factor of $\frac{0.5}{0.222} > 1$ is added for every associated award movie, and a factor $\frac{0.181}{0.222} < 1$ for every non-award movie (cancelling out the 0.222 terms).

4 TET Learning

We first describe in more detail the learning problem we want to solve. Our data consists of a model $\mathcal{M}$ in the sense of Definition 1. In our implementation, $\mathcal{M}$ is given as a relational database containing one table for each $r \in R$, where the table for r contains all tupels $\boldsymbol{a} \in M^{\mathit{arity}(r)}$ for which $I(r(\boldsymbol{a})) = \mathit{true}$. Furthermore, we are given an initial *target table*, i.e. a table consisting of a set of examples with $+/-$ class labels. For example, a learning problem given by the data depicted in Figure 1, and m_1, m_2 as a negative and positive example, respectively, would be given by the 4 leftmost tables in Table 4. Columns in the data tables are headed by synthetic identifiers *Arg*$_i$. Columns in the target table (other than the class label column) are headed by variable names, which will then become the names of the free variables in the TET we construct. Thus, given the input we will want to construct a TET $T(v)$ over the signature $R = \{\mathit{movie}, \mathit{cast}, \mathit{award}\}$ that captures features of v that are predictive for the class label given in *target*.

Note that in this representation of the learning task it is not visible that the class we want to predict is actually the *award* relation. Such a representation is appropriate, for example, when we have temporally stratified data, and the task is to predict for current movies m_1, m_2 whether they will receive an award – a prediction for which we may use the award label of older movies $m_3, \ldots, m_{19}$ (this is the scenario we have in our IMDB

Table 1. Data and Target Tables

Arg$_1$
m_1
m_2
$\vdots$
m_{19}
movie

Arg$_1$	Arg$_2$
m_1	a_1
m_1	a_2
$\vdots$	$\vdots$
m_{19}	a_7
cast	

Arg$_1$
m_4
m_8
m_{17}
m_{18}
award

v	Label
m_1	$-$
m_2	$+$
target	

Input data

v	w	Label
m_1	a_1	$-$
$\vdots$	$\vdots$	$\vdots$
m_1	a_4	$-$
m_2	a_5	$+$
m_2	a_6	$+$
m_2	a_7	$+$
target		

Local target

experiment, Section 5). For prediction problems without such a stratification, notably when for the prediction of any object's class label one may use the class label of all other objects in the domain, the target table will essentially duplicate the class table (in our example, the award table would also contain m_2, and the target table would contain all movies, correctly labeled + or $-$). In such a case suitable syntactic constraints can be imposed in the TET construction that prevent the construction of "illegal" features (*award*(v) as a feature for predicting the class of v).

Our general approach to TET learning is a recursive top-down construction that associates with each node a local discrimination task represented by a local target table. This makes our approach somewhat related to decision tree learning. In a fundamental difference to decision tree learning, however, it will generally not be the case that the target tables of child nodes partition the target table of their parent. In our running example, if starting with the input data in Table 4 we initially constructed the extension *movie*$(v) \overset{w}{\text{—}}$ *cast*(v, w), then we would associate with the node *cast*(v, w) the local target table shown on the right of Table 4. Rather than a reduction of the discrimination task to a smaller set of examples, as in decision tree learning, the construction of this new target table amounts to a problem transformation: the problem of predicting the label of a movie is transformed into predicting the label of movie/actor pairs in the new target table, which may be effected by taking into consideration attributes of both movies and actors, as well as additional relations between actors and movies (if any such exist in the data).

The exact specification of the construction of local target tables is as follows. Let n be a TET node associated with a local target table $tt_n(\boldsymbol{v}, L)$ with columns for variables $\boldsymbol{v}$ and label L. Let n' be a child of n labeled with type $\sigma(\boldsymbol{v}, \boldsymbol{w})$, and reached by an edge labeled with variables $\boldsymbol{w}$ (we include the possibility that $\boldsymbol{w}$ is the empty tuple, i.e. the edge is really unlabeled). Then n' is associated with a target table $tt_{n'}$ with columns for $\boldsymbol{v}, \boldsymbol{w}$ and L, defined as:

$$tt_{n'}(\boldsymbol{v}, \boldsymbol{w}, L) = \{(\boldsymbol{a}, \boldsymbol{b}, l) \in M^{|\boldsymbol{v}|+|\boldsymbol{w}|} \times \{+, -\} : (\boldsymbol{a}, l) \in tt_n; \mathcal{M} \models \sigma(\boldsymbol{a}, \boldsymbol{b})\}. \quad (3)$$

When building the TET we score candidate extensions based on a *generalized information gain (gig)* measure, which is a function of the old and new target table. The *gig* measure plays a central role in our TET learning method, and is more fully discussed in Section 4.1 below. In addition to *gig* scoring of candidate extensions (which is a local score for one node and a child), our learning method is driven by a global evaluation of the current full TET structure based on the accuracy of this TET in conjunction with the chosen classification model. In our implementation this is the discriminant function model. Using a different type of TET-based classifier (e.g. an SVM based on a TET-kernel) could obviously lead to different results in the TET learning. Thus, the TET learner does not perform "pure" feature discovery; rather, it constructs features that are useful in combination with a discriminant function classifier. However, our experimental results (Section 5) indicate that the features we obtain are not highly biased towards this particular classification model.

Table 2 shows the general structure of our TET learning method. The method is essentially implemented as a node constructor, that receives as arguments the data, a target table tt to be classified, a pointer r to the root of the TET in which this node is

Table 2. TET learning

```
$TET\_node(Data\,\mathcal{M}, Labeled\ table\ tt, TET\_node\ r, Type\ \tau(\boldsymbol{v}))$
1. this.type = $\tau(\boldsymbol{v})$
2. this.weight = positive_class_frequency(tt)
3. TET_node root
4. if ($r$=null) root=this else root=$r$
5. $current\_score = predictive\_score(root)$
6. EXT:=possible_extensions(root,this)
7. for all $\sigma(\boldsymbol{v},\boldsymbol{w}) \in EXT$ compute $gig(tt, \sigma(\boldsymbol{v},\boldsymbol{w}))$
8. CAND:= candidate_extensions(EXT,gig-values)
9. for all $\sigma(\boldsymbol{v},\boldsymbol{w}) \in CAND$
10.     $tt' = construct\_tt(\mathcal{M}, tt, \sigma(\boldsymbol{v},\boldsymbol{w}))$
11.     add as child $c = TET\_node(\mathcal{M}, tt', root, \sigma(\boldsymbol{v},\boldsymbol{w}))$
12.     $new\_score = predictive\_score(root)$
13.     if $new\_score - current\_score < threshold$
14.       delete $c$
15.     else current_score=new_score
```

constructed, and the type $\tau(\boldsymbol{v})$ with which the node is to be labeled. A TET is learned by the call

$$root\_of\_learned\_TET = TET\_node(\mathcal{M}, tt, null, \top(\boldsymbol{v}))$$

(because of initialization issues, all our TETs have the vacuous type $\top(\boldsymbol{v})$ in the root).

The construction works as follows: Line 1 labels the node under construction with its type. Line 2 sets the weight for the discriminant function d_+ for this node. It is just the relative frequency of positive examples in the target table associated with this node (the weight for d_- being one minus this weight). Thus, the discriminant function here is learned (at little extra cost) in parallel with the TET construction. If a different classification model was used for the TET construction, then line 2 would be omitted. Lines 3-4 set a local *root* variable pointing to the root of the TET under construction.

The function *predictive_score* called in lines 5 and 12 performs the global evaluation of the current TET based on its predictive performance in conjunction with the chosen classification model. If a model other than the discriminant function here is used, then calls to *predictive_score* may require computationally expensive model training for the current TET.

Lines 6-8 are crucial: here a subset of all the possible extensions of the current node is constructed for further exploration. This operation is analogous to refinement operators in ILP. Our construction is in two steps: in the first step the set of possible extensions for the current node is constructed by the function *possible_extensions*. This function can implement various constraints and a language bias. In our implementation, we restrict possible extensions to contain only one literal, and to introduce at most one new variable. The function can also take TILDE-style user-defined *rmode* declarations, that can force certain arguments of the new literal to be filled with variables already present in the parent node (input variable), or with a new variable introduced by this extension (output variable). In all our experiments we use rmode declarations that force the argument of any unary relation to be an input variable. Furthermore, *possible_extensions* is used to implement a termination condition: if the depth of the current node in the TET has reached a (user specified) maximum depth, then *possible_extensions* will return an empty set. In a second step, the generalized information gain is computed for all possible extensions; the function *candidate_extensions* then performs a selection based on *gig* values. Our current implementation of *candidate_extensions* selects all extensions whose *gig* value exceeds user defined thresholds (we use different thresholds depending on whether a candidate extension introduces a new variable; the *gig* values for these two types of extensions are incomparable, see Section 4.1).

For each candidate extension then a child node is created. The function *construct_tt* constructs the local target table for the child according to (3). After line 11 is executed, a whole new subtree is rooted at the new node c. Lines 12-15 then evaluate the extension of the old TET with this new subtree, and either accept or reject the subtree based on a user defined threshold for the required global score improvement.

4.1 Generalized Information Gain

To select promising candidate extensions in lines 7-8 of the TET learner, we use a *generalized information gain* measure that we now introduce. Given is a TET node labeled with a target table $tt(\boldsymbol{v}, L)$. We distinguish extensions with unlabeled and labeled edges, which we also call *condition introductions* and *object introductions*, respectively.

In a condition introduction we add a child node labeled with a type $\sigma(\boldsymbol{v})$ that puts additional constraints on the tuples in $tt(\boldsymbol{v}, L)$, and the new target table obtained from tt and $\sigma(\boldsymbol{v})$ via (3) is just a subset of tt. We denote this sub-table $tt_{\sigma(\boldsymbol{v})}$. Let $H(tt)$ denote the entropy of the class label distribution in tt. Standard information gain

$$ig(\sigma(\boldsymbol{v})) := H(tt) - \frac{|tt_{\sigma(\boldsymbol{v})}|}{|tt|}H(tt_{\sigma(\boldsymbol{v})}) - \frac{|tt \setminus tt_{\sigma(\boldsymbol{v})}|}{|tt|}H(tt \setminus tt_{\sigma(\boldsymbol{v})}) \qquad (4)$$

can therefore be used to measure the informativeness of a candidate condition introduction.

When evaluating candidate object introductions, we have to consider two different aspects: first, extending a node in the current TET by an object introduction refines the feature represented by the TET and modifies the discriminant function, which may directly lead to improved predictive performance. In this case we say that the object introduction is *directly informative*. Secondly, an object introduction can be *potentially informative* when the problem transformation represented by the new target table enables additional strategies for discriminating between positive and negative examples.

To illustrate these two aspects, consider the candidate extension $\xrightarrow{w}$ *cast*(v, w) of the root $\top(v)$ for the data in Table 4. This extension introduces the number of actors in a movie as a feature. Since in the data movies in the positive class have fewer actors than movies in the negative class (i.e. there is *degree disparity* [8]), this extension is directly informative. Note that due to the fact that every movie has at least one actor, *cast*(v, w) is a *non-discriminating relation* [4] for standard ILP systems without aggregation functions. The extension also is potentially informative, because, as the new local target table indicates, movies in the positive class have different actors in their cast than movies in the negative class, and therefore the classes can also be distinguished if we succeed in separating the two sets of actors by subsequent extensions of the new *cast*(v, w) node.

Our goal is to find an evaluation measure for both the direct and potential informativeness of a candidate object introduction $\xrightarrow{\boldsymbol{w}} \sigma(\boldsymbol{v}, \boldsymbol{w})$. Measuring the potential informativeness serves similar goals as *lookahead* [2,4] in ILP systems, but differs in that it tries to achieve this goal by considering only one-step extensions (i.e. single literals in the context of our learner), rather than the combinatorial search space of possible multi-step refinements.

Consider a specific tuple of objects $\boldsymbol{a}$ that can be substituted for the variables $\boldsymbol{w}$ in $\sigma(\boldsymbol{v}, \boldsymbol{w})$. Then we can consider $\sigma(\boldsymbol{v}, \boldsymbol{a})$ as a condition introduction (it imposes on the tuples in $tt(\boldsymbol{v}, L)$ the condition of being related to the fixed objects $\boldsymbol{a}$ in the way specified by $\sigma(\boldsymbol{v}, \boldsymbol{a})$), and we can measure $ig(\sigma(\boldsymbol{v}, \boldsymbol{a}))$ according to (4). If objects $\boldsymbol{a}$ are either mostly associated with $\boldsymbol{v}$'s from the positive class, or with $\boldsymbol{v}$'s from the negative class, then $ig(\sigma(\boldsymbol{v}, \boldsymbol{a}))$ will tend to be larger than when no such association exists. If this association exists for a significant proportion of tuples $\boldsymbol{a}$, then this will be reflected in the generalized information gain defined as

$$gig(\sigma(\boldsymbol{v}, \boldsymbol{w})) := \frac{1}{|\,bindings(\boldsymbol{w})\,|} \sum_{\boldsymbol{a} \in bindings(\boldsymbol{w})} ig(\sigma(\boldsymbol{v}, \boldsymbol{a})),$$

where $bindings(\boldsymbol{w})$ is the set of tuples of objects that can be substituted for $\boldsymbol{w}$. The values one obtains for $gig(\sigma(\boldsymbol{v}, \boldsymbol{w}))$ must be interpreted with some care: in most situations $|\,tt_{\sigma(\boldsymbol{v},\boldsymbol{a})}\,|/|\,tt\,|$ will be small for all $\boldsymbol{a}$, and therefore also $gig(\sigma(\boldsymbol{v}, \boldsymbol{w}))$ will be a small number. It therefore does not make sense to compare $gig(\sigma(\boldsymbol{v}, \boldsymbol{w}))$ against the information gain $ig(\sigma(\boldsymbol{v}))$ of a simple condition introduction. However, $gig(\sigma(\boldsymbol{v}, \boldsymbol{w}))$ provides a meaningful measure to compare several candidate object introductions against each other. [3]

So far, we have motivated *gig* by its ability to measure the potential informativeness of an object introduction. However, direct informativeness from degree disparity, too, will increase *gig*: when $\boldsymbol{v}$'s in the positive class, on average, have more associated $\boldsymbol{w}$'s than $\boldsymbol{v}$'s in the negative class, then, conversely, for many $\boldsymbol{a}$'s in $bindings(\boldsymbol{w})$ the distribution of $\boldsymbol{v}$'s within $tt_{\sigma(\boldsymbol{v},\boldsymbol{a})}$ must show a higher proportion of the positive class than in the base table tt. This leads to higher values of $ig(\sigma(\boldsymbol{v}, \boldsymbol{a}))$, for many $\boldsymbol{a}$'s, and thus a higher value of $gig(\sigma(\boldsymbol{v}, \boldsymbol{w}))$.

5 Experiments

Our experiments focus on the capability of TET learning as a feature discovery method. We are interested in whether the TET learner will produce known relevant and/or new useful and interpretable features. We also investigate the predictive performance of the learned TETs in conjunction with the discriminant function classifier. We compare

[3] *gig* in Table 2 refers to either *gig* when object introductions are scored, or standard *ig* for scoring condition introductions.

the TET learner with a range of other relational learning systems: *relational probability trees* [11] as implemented in the *Proximity* system (kdl.cs.umass.edu/proximity), TILDE [3] (a relational decision tree learner driven by traditional info-gain), and the *Alchemy* system (alchemy.cs.washington.edu) for Markov Logic Networks [15].

Slotchain data (synthetic) An important type of relational features consists of attribute values of other objects that are reachable via a certain sequence of relations, also called *slotchain*. The challenge posed by slotchain features lies in the fact that they consist of a conjunction of literals, none of which is informative individually. Thus, slotchain features are useful prototype features for testing the effectiveness of tools like lookahead or our generalized information gain.

We generated data representing slotchain dependencies as follows: the domain consists of 300 objects each of 3 different types $\mathit{type}_0, \ldots, \mathit{type}_2$. Between objects of type i and $i + 1$ ($i = 0, 1$) 2 different binary relations $\mathit{rel}_{i,k}$ ($k = 0, 1$) are generated by randomly selecting for each object o of type i exactly 2 distinct objects as $\mathit{rel}_{i,k}$-successors. Objects of type type_2 have a boolean attribute *att*. Objects of type type_0 have a class label *class*. The probability that $o \in \mathit{type}_0$ is in the positive class is 0.8 if o is connected via the chain of relations $\mathit{rel}_{0,0}, \mathit{rel}_{1,0}$ to an object $o' \in \mathit{type}_2$ with $\mathit{att}(o) = \mathit{true}$; and 0.1 otherwise. The *att* relation is true for $o' \in \mathit{type}_2$ with probability 0.2. The relevant feature for predicting the class label therefore is

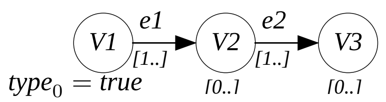
$$\top(v) \overset{w_0}{\text{—}} \mathit{rel}_{0,0}(v, w_0) \overset{w_1}{\text{—}} \mathit{rel}_{1,0}(w_0, w_1) \text{—} \mathit{att}(w_1)$$

The TET learner returned the weighted TET

$$\begin{tikzcd}
 & & \underset{(0.6)}{\mathit{rel}_{0,0}(w_2, w_1), w_2 \neq v} & \\
\underset{(0.57)}{\top(v)} \arrow[r, dash, "w_1"] & \underset{(0.57)}{\mathit{rel}_{0,0}(v, w_1)} \arrow[ur, dash, "w_2"] \arrow[dr, dash, "w_3"] & & \\
 & & \underset{(0.57)}{\mathit{rel}_{1,0}(w_1, w_3)} \arrow[r, dash] & \underset{(0.74)}{\mathit{att}(w_3)}
\end{tikzcd}$$

While this TET contains a spurious branch apart from the correct feature, this branch does not contribute much to the discriminant function, because the weight of its root is very close to the weight of the root's parent. This weighted TET achieves a predictive accuracy of 0.86 and AUC(ROC) value of 0.87 on a test set.

A comparison with the *Proximity* implementation of relational probability trees (Prox-RPTs) is made difficult by the fact that Proximity does not operate directly on the relational database, but on set of examples consisting of small subgraphs of the full data graph. These subgraphs have to be constructed by a user-defined query, the formulation of which already amounts to a large part of the feature discovery process. Queries are formulated in a language of graph patterns. To retrieve suitable subgraphs for the slotchain learning task, we used the query



This query retrieves for every object $o \in \mathit{type}_0$ the subgraph consisting of all nodes and edges that are at most two steps away from o. The feature language employed by Prox-RPTs uses count and aggregation operators for attributes of node and edge elements in the query graph. The RPT we obtained had in its root the feature $\mathit{prop}(\mathit{V3.att} = \mathit{true}) \geq 0.875$, and similar features referring to *V3.att* in its other nodes. Thus, while the Prox-RPT learner determined the relevance of the *att* attribute in nodes two steps away, it did not refine this in terms of the exact slotchain by which these nodes are reached. Accuracy (0.65) and AUC(ROC) (0.633), therefore, were much lower than in the TET-based model.

TILDE without lookahead, as expected, was unable to learn the slotchain (it returns the empty clause). Only when the correct slotchain is given as a user-defined lookahead clause, can this feature be learned.

Alchemy structure learning did not terminate on the slotchain data within 6 hours (whereas the TET learner took about 30 seconds, and Proximity less than 10 seconds). However, Alchemy tries to solve the much harder task of learning a full generative model for all relations in the data, rather than a predictive model for the *class* attribute.

IMDB Our second experiment uses the real world Internet Movie Database (IMDb). In our experimental setup we follow [10]. The problem is to predict whether a movie will earn more than \$2 million in its opening weekend (expressed by a *receipts* class label). Available data includes attributes of movies, e.g. a *genre* attribute, binary relations between movies and actors, movies and producing studios, and movies and directors, as well as an *award* attribute of actors. Movie data from 5 successive years 1996-2000 was used, and the 4 separate prediction problems for the class labels of movies from the years 1997,...,2000 were considered. When classifying movies from a given year, then the *receipts* label of movies from the preceding year is available. In all 4 versions of the experiment we obtained quite similar TETs, a representative one being

$$\begin{tikzcd}
 & & \top(v) \arrow[dll] \arrow[dl] \arrow[d, "s"] \arrow[dr, "a"] & \\
\mathit{thriller}(v) \arrow[d, "s"'] & \neg\mathit{thriller}(v) \arrow[d] & \mathit{studio}(s,v) \arrow[d, "w"'] & \mathit{cast}(a,v) \\
\mathit{studio}(s,v) \arrow[d, "w"'] & T_1(v) & \mathit{studio}(s,w) \arrow[dl] \arrow[dr] & \\
\mathit{studio}(s,w) & \mathit{receipts}(w) & & \neg\mathit{receipts}(w)
\end{tikzcd}$$

Because of space limitations, the subtree $T_1(v)$ here is not shown. It consists of further condition introduction nodes (10 in total) testing different genre attributes of movies. The most interesting feature encoded by this TET (which is present in all 4 versions of the experiment) is the third branch from the left, which counts the *receipt* attribute values among the movies produced (in previous years) by the same studio as the movie to be classified. Interestingly, this is essentially the underlying count statistic of the feature that Neville and Jensen [10] obtain at the root of a relational probability tree they construct for the same classification problem:

$$\text{Studio Movie } \mathit{Prop}(\mathit{receipts} = y) > 0.6$$

The TETs we construct, together with the simple discriminant function as the classification model, yields accuracy results (84.4, 78.8, 80.7, and 86.4%, respectively, for the 4 different years) which are competitive with the results reported by Neville and Jensen.

CORA Our third experiment uses the CORA citation data. We consider the entity resolution problem as investigated by Singla and Domingos [15]; we use the dataset available at alchemy.cs.washington.edu. The predictive tasks are to decide whether two strings found in the *author*, respectively *venue* field in two different bibliographic entries refer to the same entity, and whether two different bibliographic entries refer to the same paper. Types of objects in the database include *author*, *venue*, *bib*, *word*,... Available relations include the binary relations $\mathit{hasWordVenue}(v, w)$ representing that venue v contains the word w, $\mathit{bibToVenue}(b, v)$ representing that bibliographic item b contains v in the venue field, and similar relations for the fields 'author' and 'title'.

We ran the TET learner on each of the 5 different training sets defined by a 5-fold division of the source data. In all 3 tasks the TETs constructed over the different folds exhibited a quite stable structure, with some branches being present in all folds. A representative TET for the *venue* resolution task is shown at the left in Figure 3.

The left branch (which was constructed in all 5 runs) represents a central feature for entity resolution: whether two entities (i.e. strings) are the same depends on the number of words that appear in both of them. This simple fact is called *reverse predicate equivalence* in [15], and hand-coded into the prediction model via logical axioms of the form

$$\mathit{hWV}(v_1, w) \wedge \mathit{hWV}(v_2, w) => \mathit{sameVenue}(v_1, v_2)$$

The right branch first introduces the bib-entries that have the string v_1 in the venue field). The left sub-branch tests whether b_1 also contains v_2 in the venue field. In effect, this is a test whether v_1 and v_2 are identical strings – an obvious feature to consider, yet one that is not expressible in a more straightforward manner with the given relational vocabulary! The right sub-branch introduces the title of the bib entry b_1, and then counts the number of bib-entries b_2 that also have this title, and that contain venue v_2 in the

$$\begin{tikzcd}
 & \top(v_1, v_2) \arrow[dl, "w"'] \arrow[dr, "b_1"] & & \\
\mathit{hWV}(v_1, w) \arrow[d] & & \mathit{bTV}(b_1, v_1) \arrow[dl] \arrow[dr, "t"] & \\
\mathit{hWV}(v_2, w) & \mathit{bTV}(b_1, v_2) & & \mathit{bTT}(b_1, t) \arrow[d, "b_2"] \\
 & & & \mathit{bTT}(b_2, t) \arrow[d] \\
 & & & \mathit{hTV}(b_2, v_2)
\end{tikzcd}$$

$$\begin{tikzcd}
 & \top(p, r) \arrow[d, "r1"'] & \\
 & \mathit{within\_5}(p, r, r1) \arrow[d] & \\
 & \mathit{consCYS}(p, r1) \arrow[d, "r2"'] & \\
 & \mathit{within\_5}(p, r1, r2) \arrow[dl] \arrow[dr] & \\
\mathit{consXXX}(p, r2) & & \mathit{consYYY}(p, r2)
\end{tikzcd}$$

Fig. 3. TETs for Venue resolution (left) and metal binding sites (right)

venue field. In effect, this whole branch represents a feature that considers how often v_1 and v_2 appear in bib-entries b_1, b_2 that have the same title.

For the three prediction tasks, author resolution, venue resolution, and bib-entry resolution, we achieved with our weighted TETs AUC(PR) values of 0.987, 0.771, 0.938, respectively (micro-averaged over the 5 folds). This is slightly better for author and venue than the values 0.980, 0.743, 0.971 reported in [15] for a Markov Logic Network model that (like our TET) is language independent, i.e. does not contain rules referring to specific strings occurring in the data. The worse performance on Bib can probably be explained by the fact that [15] perform multi-task classification, and that Bib resolution benefits more from the multi-task setting than Author and Venue (there is no direct evidence for two Bibs being the same: they tend to be the same if their authors and venues are the same).

Metal binding sites In our last experiments we predict metal binding residues in proteins, using sequence information only. Metal ions are essential for Life, performing structural, catalytical and regulatory roles in the cell. About one third of the known proteins is believed to bind metal ions in their native conformation. A single metal ion is typically coordinated by a number of residues ranging from one to eight, each metal having a preferred coordination number and a range of possible alternatives. The most common metal ligands are CYS and HIS, followed by ASP and GLU which are however far more abundant in proteins. Metal binding sites typically show many regularities in terms of number, type, and distance of ligands and surrounding residues. Biologists have tried to encode such regularities in motifs identifying specific metal binding sites, as well as other biologically relevant portions of proteins. PROSITE [6] motifs consist of either regular expressions or position-specific profiles with amino acid weights and ggvgap costs. The former are extremely specific but have a very low recall, the latter show a more balanced precision/recall ratio but their performance is still rather unsatisfactory. While providing interpretable explanations, such motifs fail to capture much of the information provided by the sequence, especially when enriched by evolutionary information in the form of multiple alignment profiles, as shown by the performance achieved by a supervised learning architecture [12] fed by such inputs. Our TET learning algorithm discovered relevant and interpretable combinatorial features that outperform the knowledge-based regular expressions defined in PROSITE. Furthermore, counts-of-counts capabilities indeed provide additional discriminative power, as shown by experimental comparisons with Tilde and with TETs based on regular expression-like features only.

We learned two separate TETs for CYS and HIS respectively. Objects in the domain consist of proteins and residues within proteins. Residue attributes are extracted from multiple alignment profiles, and consist of the binarized conservation of either relevant residue types such as CYS, HIS, ASP, GLU or PRO, or relevant residue classes such as *small*, *hydrophobic* or *negative*. Relations have the form *Within_n(p,r1,r2)* and *Plus_n(p,r1,r2)*, and represent pairs of residues *(r1,r2)* in a certain protein *p* whose sequence separation is at most or exactly *n*, respectively. Note that despite their similarity, these two types of relations express quite different features: once one of the two residues is given, *Plus_n(p,r1,r2)* is satisfied by at most one other residue, and will thus

Table 3. Microaveraged area under the ROC curve (%) for CYS and HIS metal bonding state prediction.

Task	TET_{full}	TET_{regexp}	PROSITE_{motifs}	$\text{PROSITE}_{patterns}$	TILDE	SVM-BRNN
CYS	85.6±0.8	81.6±0.9	76.5±0.9	64.4±1.0	83.8±0.8	93.2±0.6
HIS	80.1±1.0	77.7±1.1	67.3±1.2	60.5±1.2	71.1±1.1	88.6±0.8

lead to regular expression-like features similar to PROSITE patterns (but typically less specific). On the other hand, *Within_n(p,r1,r2)* allows to collect all residues in the neighbourhood of a given residue, and naturally generates counts-of-counts types of features. We ran a five-fold cross validation procedure on the same folds employed in [12]. Table 3 reports microaveraged area under the ROC results for TET with (TET_{full}) and without (TET_{regexp}) the *Within_n(p,r1,r2)* relation, PROSITE patterns and motifs and TILDE, as well as the state-of-the-art performance for this task [12]. Figure 4 reports significance of performance difference between learners, computed by a two-tailed Hanley-McNeil test [16] on areas under the ROC curve. TILDE was run with the same declarative bias used for learning TETs (rmode declarations that set one of the residues in *Plus_n* and *Within_n* relations to input, and the other to output). However, it was unable to exploit the *Within_n(p,r1,r2)* relations, which are *non-discriminating relations* [4] as any residue has at least one neighbour, while *Plus_n(p,r1,r2)* relations happen to be slightly discriminant as residues at sequence boundaries do not satisfy the relation and are typically non-binding residues. Forcing TILDE to explore *Within_n(p,r1,r2)* relations by extensive lookahead produced worse results and increased computational costs. Area under the ROC results for TILDE were calculated assigning to each example the probability of binding of the leaf node which classified it.

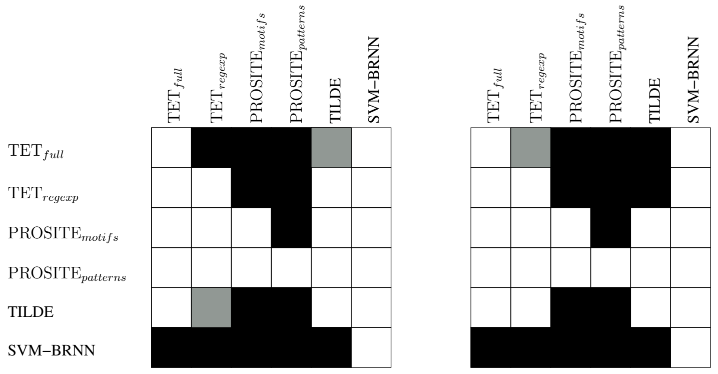


Fig. 4. Significance of performance difference between learners for the MBS data set. A black (grey) box indicates that the learner on the row is significantly better than that on the column at p=0.05 (p=0.1).

The best reported performance for this task (93.2% and 88.6% AUC for CYS and HYS, respectively) was previously obtained with SVM-BRNN, a highly engineered architecture based on support vector machines and bidirectional recurrent neural networks, fed by amino acid windows [12]. However, predictions obtained by such an architecture are not interpretable. While the TET-based classifier does not reach the same levels of accuracy, it significantly outperforms annotations curated by experts and effectively suggests novel and more compact explanations: TETs produced by our learning algorithm had 22 nodes on average, while there are 467 PROSITE patterns and 305 position-specific profiles that match at least one residue in the dataset. Counts-of-counts features do contribute to the overall TETs performance, as shown by the improvements over TILDE and TETs which rely on regular expression-like features only.

Figure 3 (right) shows a TET structure learned for CYS. Inspecting the structure, we note that the first branch learned in all folds encodes counts-of-counts features exploring the surrounding of the candidate residue, where XXX and YYY can be: HIS or *negative* identifying candidate co-ligands (ASP and GLU are both negatively charged); *polar* or *positive* identifying hydrophilic residues, an indication that the region is exposed to the exterior and thus apt to contain a binding site; *small* as small residues often provide the needed flexibility for the 3D conformation of the site. A similar branch is learned most of the time for the left context of the target residue too. Note that by concatenating the relation *within_5* twice, learned TETs manage to explore a wide context of the target residue. Empirical evidence in [12] shows that wide contexts (in a range of up to 15 residues) are useful for accurate prediction. Branches encoding standard regular expressions such as *CXXC* (two CYS separated by any two residues) are added at the latest stages only, i.e. they only play a minor refinement role.

The learning experiments reported here took in the range of 30 seconds (slotchain) and 19 hours (Bib-resolution; per fold) to complete on an Intel Xeon 3.2 GHz platform, with metal binding sites (3 h) and venue resolution (5 h; per fold) in between. By far the most time is spent on data retrieval from the underlying MySQL database.

6 Conclusion

The TET language is a very simple yet highly expressive language for representing features in relational data. Unlike previous feature representation frameworks a TET represents raw count of count statistics, not simple numeric or Boolean features obtained by aggregation of such counts. Our current TET implementation is restricted to Boolean attributes and relations only. However, on the representational, semantic level, there is no problem with extending the TET framework to multi-valued and numeric datatypes. However, numeric data poses additional challenges for transforming the complex TET values into more manageable, condensed features that then can be used in standard predictive models.

We have defined a discriminant function that enables prediction directly on the basis of TET values. Using this discriminant function and a new generalized information gain measure, we have developed a TET learner that in our experiments has been able to discover relevant and interpretable features in a variety of learning settings that differ significantly both with regard to the structure of the data, and the type of learning task.

The discriminant function we used in this paper is motivated by its grounding in classical prediction models, and the fact that it is easy to learn and evaluate, which makes it suitable for use in a wrapper evaluation procedure. In spite of its simplicity, we achieve with the discriminant function in our experiments predictive performance that is competitive with other state-of-the-art models, and sometimes (Cora) outperforms models that have been built using hand-coded domain knowledge. It must be emphasized, though, that the TET feature language is not tied to this discriminant function. Future work, therefore, will be directed towards constructing refined TET-based classification models using tools for collective and multi-task classification, as well as integrating TET features into other existing supervised learning techniques, e.g. kernel methods or relational probability trees. Also, in future work, we will further investigate "relational" information gain measure (our *gig* at this point being a somewhat ad-hoc solution) in order to obtain better theoretical justifications for the current *gig*, or improved versions thereof.

References

1. A. Van Assche, C. Vens, H. Blockeel, and S. Dzeroski. First order random forests: Learning relational classifiers with complex aggregates. *Machine Learning*, 64:149–182, 2006.
2. H. Blockeel and L. De Raedt. Lookahead and discretization in ilp. In *Proc. of the 7th Int. Workshop on ILP*, pages 77–84, 1997.
3. H. Blockeel and L. De Raedt. Top-down induction of first-order logical decision trees. *Artifi cial Intelligence*, (101):285–297, 1998.
4. L. P. Castillo and S. Wrobel. A comparative study on methods for reducing myopia of hill-climbing search in multirelational learning. In *Proc. of ICML-04*, 2004.
5. N. Friedman, Lise Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *Proc. of IJCAI-99*, 1999.
6. N. Hulo, C. J. A. Sigrist, V. Le Saux, P. S. Langendijk-Genevaux, L. Bordoli, A. Gattiker, E. De Castro, P. Bucher, and A. Bairoch. Recent improvements to the prosite database. *Nucleic Acids Research*, 32(Database-Issue):134–137, 2004.
7. M. Jaeger. Type extension trees: a unified framework for relational feature construction. In *Proceedings of Mining and Learning with Graphs (MLG-06)*, 2006.
8. D. Jensen, J. Neville, and M. Hay. Avoiding bias when aggregating relational data with degree disparity. In *Proc. of ICML-03*, 2003.
9. A. J. Knobbe, A. Siebes, and D. van der Wallen. Multi-relational decision tree induction. In *Proceedings of PKDD-99*, pages 378–383, 1999.
10. J. Neville and D. Jensen. Collective classification with relational dependency networks. *Proc. of 2nd Int. Workshop on Multi-Relational Data Mining*, pages 77–91, 2003.
11. J. Neville, D. Jensen, L. Friedland, and M. Hay. Learning relational probability trees. In *Proceedings of SIGKDDD'03*, 2003.
12. A. Passerini, M. Punta, A. Ceroni, B. Rost, and P. Frasconi. Identifying cysteines and histidines in transition-metal-binding sites using support vector machines and neural networks. *Proteins*, 65(2):305–316, 2006.
13. C. Perlich and F. Provost. Aggregation-based featrue invention and relational concept classes. In *Proc. of SIGKDD'03*, 2003.
14. A. Popescul and L. H. Ungar. Feature generation and selection in multi-relational statistical learning. In L. Getoor and B. Taskar, editors, *Statistical Relational Learning*. MIT press, 2007.
15. P. Singla and P. Domingos. Entity resolution with markov logic. In *Proc. of ICDM-06*, 2006.
16. J.A. Hanley and B.J. McNeil. A method of comparing the areas under receiver operating characteristic curves derived from the same cases. *Radiology*, 148(3):839–843, 1983.