Relational Learning with Polynomials

Ondřej Kuželka, Andrea Szabóová, Filip Železný
Department of Cybernetics
Czech Technical University in Prague
Prague, Czech Republic
Email: {kuzelon2, szaboand, zelezny}@fel.cvut.cz

***Abstract*—We describe a conceptually simple framework for transformation-based learning in hybrid relational domains. The proposed approach is related to hybrid Markov logic and to Gaussian logic framework. We evaluate the approach in three domains and show that it can achieve state-of-the-art performance while using only limited amount of information.**

I. Introduction

Modelling relational domains which contain substantial part of information in the form of real-valued variables is an important problem with applications in areas as different as bioinformatics or finance. So far there have not been many relational learning systems introduced in the literature that would be able to model multi-relational domains with numerical data efficiently. Examples of frameworks able to work in such domains are hybrid Markov logic networks [1] and Gaussian logic [2]. Another type of systems that do not directly model the probabilities but are able to learn in domains rich in numerical data are systems based on relational aggregation [3], [4]. In this paper we describe a relatively simple transformation-based framework for learning in rich relational domains containing numerical data.

The new framework exploits multi-variate polynomial aggregation functions, which is something that, surprisingly, has not been studied in the relational-learning literature yet. One of the reasons why multivariate polynomial aggregation features are interesting for predictive applications is that they can capture interactions among numerical variables. Let us suppose that we would like to predict solubility of chemicals in water from their chemical structure and from the partial charges of atoms. Solubility in water is typically correlated with polarity of molecules and polarity is a property of pairs of atoms. It would be hard to capture polarity using relational features with single-variable aggregation but it is relatively easy to capture it when the relational features are allowed to compute aggregated values of functions of multiple variables, for example, when they are able to compute averages of products of charges of pairs of atoms which are connected by a bond. The framework presented in this paper is able to discover and exploit dependencies among numerical variables in relational data.

This paper is organized as follows. We first extend the theoretical framework of Gaussian logic [2] and define so-called *polynomial features*. We then study their properties and derive efficient algorithms for *tree-like polynomial features* and *polynomial features* with bounded tree-width, in general. We also present generalization of the feature-construction algorithm RelF [5] which is able to construct large numbers of *polynomial relational features* very quickly while retaining a completeness guarantee. Finally, we evaluate the approach in predictive-classification experiments with three relational learning datasets. The algorithms presented in this paper have been implemented into the open-source software package TreeLiker available at `http://ida.felk.cvut.cz/treeliker/`.

II. Preliminaries and Notation

Let $n \in \mathbf{N}$. If $\vec{v} \in \mathbf{R}^n$ then v_i $(1 \leq i \leq n)$ denotes the i-th component of $\vec{v}$. If $I \subseteq [1; n]$ then $\vec{v}_I = (v_{i_1}, v_{i_2}, \dots v_{i_{|I|}})$ where $i_j \in I$ $(1 \leq j \leq |I|)$. To describe training examples as well as learned models, we use a conventional first-order logic language $\mathcal{L}$ whose alphabet contains a distinguished set of constants $\{\mathsf{r}_1, \mathsf{r}_2, \dots \mathsf{r}_n\}$ and variables $\{R_1, R_2, \dots R_m\}$ $(n, m \in \mathbf{N})$. An *r-substitution* ϑ is any substitution as long as it maps variables R_i only to terms r_j. For the largest k such that $\{R_1/\mathsf{r}_{i_1}, R_2/\mathsf{r}_{i_2}, \dots,\ R_k/\mathsf{r}_{i_k}\} \subseteq \vartheta$ we denote $I(\vartheta) = (i_1, i_2, \dots i_k)$. A (Herbrand) *interpretation* is a set of ground atoms of $\mathcal{L}$. $I(H)$ $(I(\varphi))$ denotes the naturally ordered set of indexes of all constants r_i found in an interpretation H ($\mathcal{L}$-formula φ).

The training examples have both *structure* and *real parameters*. A training example may e.g. describe a measurement of the expression of several genes; here the structure would describe functional relations between the genes and the parameters would describe their measured expressions. The structure will be described by an interpretation, in which the constants r_i represent uninstantiated real parameters. The parameter values will be determined by a real vector. Formally, an example is a pair (H, θ) where H is an interpretation, $\theta \in \Omega_H$, and $\Omega_H \subseteq \mathbf{R}^{|I(H)|}$ (here, $\mathbf{R}$ denotes the set of real numbers).

III. Polynomial Relational Features

Features are first-order logic expressions. For a given example (H, θ), a feature φ *extracts* some components of θ into a set of vectors. Given an example $e = (H, \theta)$ and a feature φ, the *sample set* of φ and e is the multi-set $\mathcal{S}(\varphi, e) = \{\theta_{I(\vartheta)} | H \models \varphi\vartheta\}$ where ϑ are r-substitutions grounding all free variables in φ, and $H \models \varphi\vartheta$ denotes that $\varphi\vartheta$ is true under H.

Example 1. *Let us have a feature*

$$\varphi = g(X, R_1) \wedge activates(X, Y) \wedge g(Y, R_2)$$

and a training example

$$e \approx g(a,1), activates(a,b), g(b,2), activates(b,c), g(c,3), activates(a,c)$$

(formally: $e = (\{g(a, r_1),\ activates(a, b),\ g(b, r_2),\ activates(b, c),\ g(c, r_3),\ activates(a, c)\}, (1, 2, 3)^T)$). The sample-set of feature φ w.r.t. example e is:

$$\mathcal{S}(\varphi, e) = \{(1, 2)^T, (2, 3)^T, (1, 3)^T\}.$$

Informally, when each example is viewed as an isolated relational database and when features are treated as database queries then the sample sets can be viewed as result-sets of these database queries.

A *monomial relational feature* M is a pair $(\varphi, (d_1, \dots, d_k))$ where φ is a feature with k distinguished variables and $d_1, \dots, d_k \in \mathbf{N}$. *Degree* of M is $deg(M) = \sum_{i=1}^k d_i$. Given a non-empty sample set $\mathcal{S}(\varphi, e)$, we define the *value* of a monomial feature $M = (\varphi, (d_1, \dots, d_k))$ w.r.t. example e as

$$M(e) = \frac{1}{|\mathcal{S}(\varphi, e)|} \sum_{\theta \in \mathcal{S}(\varphi, e)} \theta_1^{d_1} \cdot \theta_2^{d_2} \cdot \dots \cdot \theta_k^{d_k}$$

where θ_i is the i-th component of vector θ.

A *polynomial relational feature* is an expression of the form $P = \alpha_1 M_1 + \alpha_2 M_2 + \dots + \alpha_k M_k$ where $M_1, \dots, M_k$ are monomial features and $\alpha_1, \dots, \alpha_k \in R$ (formally expressed as a pair of two ordered sets - one of monomials and one of the respective coefficients). *Value* of a polynomial feature $P = \alpha_1 M_1 + \dots + \alpha_k M_k$ w.r.t to an example e is defined as $P(e) = \alpha_1 M_1(e) + \alpha_2 M_2(e) + \dots + \alpha_k M_k(e)$. *Degree* of a polynomial relational feature P is maximum among the degrees of its monomials. Polynomial relational features can be used to capture higher-order moments of distributions – generalizing the Gaussian logic framework [2].

IV. Properties of Polynomial Relational Features

A convenient property of polynomial relational features is their decomposability. We describe two *types* of decomposability. The first one is described in the next proposition.

Proposition 1. *Every polynomial relational feature of degree d can be expressed using monomial features containing at most d distinguished variables.*

Proof: Clearly, in order to compute a value of any monomial M of order d we need at most d distinguished variables because at most d of them can have non-zero exponent in the monomial. If we replace the distinguished variables with zero d_i in M by ordinary variables (i.e. those that do not extract any numerical values) then the new sample-set which we obtain for an example e and the new feature with fewer distinguished variables will differ from the original sample-set only in that every vector in it will miss the entries associated to the removed distinguished variables. These variables are not used in the computation of values of M anyway. ■

The decomposability of polynomial features is practical for feature generation because once we set a maximum degree of the polynomial features that we are interested in, we also know that we can use this limit for the number of distinguished variables in the generated features.

There is also a second form of decomposability for polynomial features and that is decomposability of *disconnected* features. We say that a formula F is disconnected if it can be rewritten as $F = (G) \wedge (H)$ such that G and H do not have any variable in common (note that the parentheses ensure that also the logical quantifiers are applied to the formulas G and H separately).

Proposition 2. *Let $\varphi = (\psi) \wedge (\gamma)$ be a disconnected feature such that ψ and γ do not share any variable. Let $M = (\varphi, (d_1, \dots, d_k))$ be a monomial. Then it is possible to find monomial features M_ψ and M_γ such that $M(e) = M_\psi(e) \cdot M_\gamma(e)$ for all examples e.*

Proof: It holds $\mathcal{S}(\varphi, e) = \mathcal{S}(\psi, e) \times \mathcal{S}(\gamma, e)$ where $\times$ denotes Cartesian product. We can therefore construct the monomial features as follows. First, we split the set of distinguished variable of F to two (necessarily disjoint) sets $\mathcal{R}_\psi$, $\mathcal{R}_\gamma$ according to the formula in which they appear. We split also the respective exponents to ordered sets $(v_1, \dots, v_{k_\psi})$ and $(w_1, \dots, w_{k_\gamma})$.

$$M_\psi = (\psi, (d_1^\psi, \dots, d_{k_\psi}^\psi))$$
$$M_\gamma = (\gamma, (d_1^\gamma, \dots, d_{k_\gamma}^\gamma)).$$

The product of these monomial features gives rise to the original feature because

$$\begin{aligned} M_\psi(e) \cdot M_\gamma(e) &= \left(\tfrac{1}{|\mathcal{S}(\psi,e)|} \textstyle\sum_{\theta \in \mathcal{S}(\psi,e)} \theta_1^{v_1} \dots \theta_k^{v_{k_\psi}}\right) \cdot \\ &\cdot \left(\tfrac{1}{|\mathcal{S}(\gamma,e)|} \textstyle\sum_{\theta \in \mathcal{S}(\gamma,e)} \theta_1^{w_1} \dots \theta_{k_\gamma}^{w_{k_\gamma}}\right) = \tfrac{1}{|\mathcal{S}(\psi,e)| \cdot |\mathcal{S}(\gamma,e)|} \cdot \\ &\cdot \left(\textstyle\sum_{\theta \in \mathcal{S}(\psi,e)} \theta_1^{v_1} \dots \theta_{k_\psi}^{v_{k_\psi}}\right) \cdot \left(\textstyle\sum_{\theta \in \mathcal{S}(\gamma,e)} \theta_1^{w_1} \dots \theta_{k_\gamma}^{w_{k_\gamma}}\right) \\ &= \tfrac{1}{|\mathcal{S}(\psi,e) \times \mathcal{S}(\gamma,e)|} \textstyle\sum_{\theta \in \mathcal{S}(\gamma,e) \times \mathcal{S}(\gamma,e)} \theta_1^{d_1} \dots \theta_k^{d_k} = M(e) \end{aligned}$$

■

As a consequence of Proposition 2, we can focus only on constructing connected monomial features which also means that we have to search a much smaller space of features.

V. Complexity of Polynomial Relational Feature Evaluation

When features are general conjunctive queries, the problem of evaluating them is NP-hard (e.g. by reduction from θ-subsumption hardness). Features in the form of conjunctive queries with bounded tree-width can be evaluated in time polynomial in their size, in the number of their distinguished variables and in the size of the training examples, as we show in this section.

We start by describing an algorithm for evaluation of so-called tree-like monomial features. At the end of this section, we outline how it can be extended to deal with general bounded-tree-width monomial features. First, we define tree-like features. A first-order conjunction without quantifications

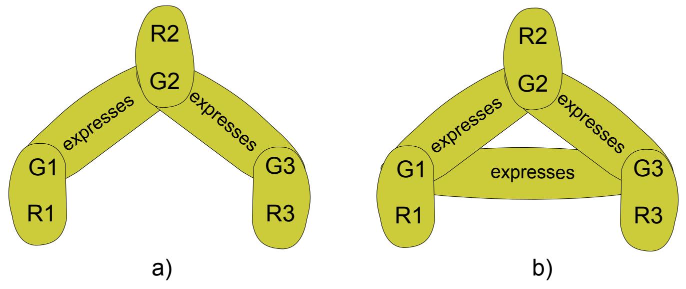

Fig. 1. Left: a hypergraph corresponding to a tree-like feature $g(G_1, R_1)$, $g(G_2, R_2)$, $g(G_3, R_3)$, $expresses(G_1, G_2)$, $expresses(G_2, G_3)$. Right: a hypergraph corresponding to a non-tree-like feature $g(G_1, R_1)$, $g(G_2, R_2)$, $g(G_3, R_3)$, $expresses(G_1, G_2)$, $expresses(G_2, G_3)$, $expresses(G_3, G_1)$.

C is *tree-like* if the iteration of the following rules on C produces the empty conjunction: (i) *Remove an atom which contains fewer than 2 variables.* (ii) *Remove a variable which is contained in at most one atom.* Intuitively, a tree-like conjunction can be imagined as a tree with the exception that whereas trees are graphs, conjunctions correspond in general to hypergraphs. Monomial features based on tree-like formulas are called tree-like. See Fig. 1 for an example of a tree-like and non-tree-like feature.

The basic ideas of the algorithm for computing values of monomial features can be summarized as follows: Let us have a tree-like feature φ with k distinguished variables, a monomial feature $M = (\varphi, (d_1, \ldots, d_k))$ and an example e. For simplicity, we assume that any literal $l \in \varphi$ can contain at most one distinguished variable[1]. Our task is to compute the value $M(e)$. We start by picking a literal $l \in \varphi$ containing distinguished variable R_1 and ground all its variables using a substitution $\vartheta : vars(l) \rightarrow constants(c)$ so that $e \models \varphi\vartheta$. We then create a new auxiliary monomial $M_\vartheta = (\varphi\vartheta, (d_1, \ldots, d_k))$. The problem of computing $M_\vartheta(e)$ can be decomposed as:

$$M_\vartheta(e) = (R_1\vartheta)^{d_1} \cdot \prod_i M_i(e) \tag{1}$$

where $M_1, \ldots, M_m$ are connected sub-features of $\varphi\vartheta$ which arise when we remove literal $l\vartheta$ from $\varphi\vartheta$. This follows from Proposition 2.

The value $M(e)$ can be then computed as

$$M(e) = \frac{1}{\sum_{\vartheta \in \Theta} \alpha_\vartheta} \sum_{\vartheta \in \Theta} \alpha_\vartheta M_\vartheta(e)$$

where $\Theta = \{\vartheta : vars(l) \rightarrow constants(c) | e \models \varphi\vartheta\}$ is the set of all true groundings of literal l and α_ϑ are the numbers of true groundings of $\varphi\vartheta$.

It might not yet be clear why the outlined method should run in time polynomial in the size of φ, in the size of e and in the number of distinguished variables. First, it is well-known that the sets Θ and the coefficients α_i, which are the numbers of true groundings of the tree-like feature φ, can be computed in polynomial time. Next, it is not hard to show that the values $M_\vartheta(e)$ can be computed using these pre-computed parameters by a dynamic-programming method. We note that the algorithm is essentially a generalization of the algorithm for computing covariance-like matrices and mean-like vectors in Gaussian features presented in [2].

The algorithm for computing values $M(e)$ of tree-like features can be used to efficiently compute values of monomial features which are not tree-like but have bounded tree-width. This can be done by computing a tree-decomposition [6] of width k of the feature and then by applying the algorithm for tree-like features on the tree-decomposition where only one occurrence of each distinguished variable is retained.

[1]This is without loss of generality because any representation with demanding more than one distinguished variable per literal can be rewritten into a representation where there is always only one distinguished variable per literal.

VI. Construction of Polynomial-Features

In this paper, we follow a straightforward transformation-based approach to hybrid relational learning with polynomial relational features. The approach is a generalization of existing aggregation-based systems that have been introduced in relational learning [3], [4] which surprisingly did not incorporate any multi-variate aggregate functions. First, a set $\mathcal{F}$ of monomial relational features is constructed. In the second step, all monomial features are evaluated w.r.t. all examples in the dataset and the values are stored in a table. This gives us an attribute-value table which can be processed by any attribute-value learning algorithm such as SVM or random forest. This is an instance of a general strategy known as *propositionalization* [7].

We integrated monomial-feature construction into RelF [5] which is a state-of-the-art feature-construction algorithm specialized for tree-like features. Based on the properties of polynomial relational features, we extended efficient filtering of *reducible* and *redundant* features during feature construction process while retaining completeness in the following sense: If there is a tree-like feature φ complying with the user-specified language-bias then the algorithm either constructs this feature or another feature ψ such that the value of any monomial based on φ is equal to the value of a monomial based on ψ.

We do not describe the language bias used by the feature construction algorithm in full detail here due to space constraints (the details can be found in [5]). We just present an informal description of it. In the feature construction system RelF and also in similar relational learning systems, language bias is usually specified using so-called *templates* or *mode declarations*.

For instance, let us have language bias expressed through the following *template* τ:

$$\begin{aligned}\tau \approx \quad & hasCar(-c), hasLoad(+c, -l), box(+l), \\ & tri(+l), weight(+l, *real)\end{aligned}$$

Templates specify which predicates can be used for construction of features. In our case, the features can be composed of predicates $hasCar/1$, $hasLoad/2$, $box/1$, $tri/1$ and

Algorithm 1 FEATURECONSTRUCTOR (Sketch): Given a RelF's language bias $\mathcal{L}$, the algorithm constructs a set of non-redundant polynomial features.

```
1:  Input: Language bias $\mathcal{L}$, Examples $E$;
2:  $Features \leftarrow \{\}$
3:  $LanguageBiasAtoms \leftarrow$ ordered atoms from $\mathcal{L}$
4:  for $\forall Atom \in LanguageBiasAtoms$ do
5:      $NewFeatures \leftarrow$ Combine$(Atom, Features, E)$
6:      $Features \leftarrow Features \cup NewFeatures$
7:      Filter (weakly) redundant features from $Features$
8:  end for
9:  $Features \leftarrow Features \cup$ Combine2$(Features, E)$
10: return $Features$
```

$weight/2$. Templates also specify how literals in features can be interconnected using variables and constants. We use signs $+$, $-$ and $\#$ for this. The sign $-$ denotes so-called *output arguments*, the sign $+$ denotes *input arguments*. Every variable (except the distinguished variables R_i) that appears in a feature must appear exactly once in an output argument and at least once in an input argument. An argument marked by $\#$ must always contain a constant symbol and an argument marked by $*$ must always contain a distinguished variable R_i. Arguments are typed. Any variable can appear only in arguments labelled by the same type. Therefore, the following is a correct feature:

$$\varphi = has(Car) \wedge hasLoad(Car, Load) \wedge weight(Load, R_1)$$

as well as

$$\begin{array}{rl}\varphi = & has(Car) \wedge hasLoad(Car, Load) \wedge box(Load) \\ & hasLoad(Car, Load2) \wedge tri(Load2)\end{array}.$$

These simple rules defining valid features provide the users with means to navigate the feature construction algorithm to the desired regions of the feature space. When the form of templates is further restricted, the language bias satisfies the following properties: (i) it constrains features to be tree-like conjunctions, (ii) any tree-like feature can be represented using it and (iii) given a specification of the language bias, a complete set of non-redundant features can be constructed by Algorithm 1. These three properties of the language bias were proved in [5].

A sketch of the feature construction algorithm based on RelF [5] is shown in Algorithm 1. The algorithm can be viewed as constructing features from small building blocks - called sub-features. It starts with literals from the given language bias which will correspond to *leafs* of the constructed tree-like features. Then it combines these *leafs* with literals which can be their parents (in the resulting tree-like features). After that it continues by combining sub-features constructed in this way with other literals and other already generated sub-features. This process is repeated iteratively until all features are constructed. The process of combining sub-features into bigger sub-features is done by the procedure Combine (used in Algorithm 1) which gets an atom A specified in the language bias, a set of already generated sub-features and an example on its input. The procedure then finds all already generated sub-features φ which can be combined with the atom A while satisfying the given language bias and combines them with the atom A - producing a set of new sub-features.

The feature construction algorithm can produce an overwhelmingly large number of features therefore it is sometimes necessary to filter the set of constructed features. To this end we define *redundancy* of features. Let $\mathcal{E}$ be a set of examples and let $\mathcal{F}$ be a set of features. We say that φ is redundant w.r.t. $\mathcal{E}$ if there is another feature ψ such that $\mathcal{S}(\varphi, e) = c \cdot \mathcal{S}(\psi, e)$ for all $e \in \mathcal{E}$ where $c \cdot \mathcal{S}(\psi, e)$ denotes a multi-set obtained from the multi-set $\mathcal{S}(\psi, e)$ by multiplying multiplicity of each element from $\mathcal{S}(\psi, e)$ by $c \in \mathbf{R}$. The rationale for this definition is that if $\mathcal{S}(\varphi, e) = c \cdot \mathcal{S}(\psi, e)$ then $M(e) = N(e)$ for all $e \in \mathcal{E}$ and all monomials of the form $M = (\varphi, (d_1, \dots, d_k))$ and $N = (\psi, (d_1, \dots, d_k))$. Given a set of features, some of which are redundant, filtering of redundant features means finding a maximal subset of non-redundant features, which is trivial (we just keep one feature for each set of features with sample sets equivalent up to multiplication).

Thus far, we have explained how to recognize that a feature is redundant but we have not explained how we can recognize that a sub-feature (i.e. a building block) will remain redundant no matter with what Algorithm 1 would combine it. First, we may notice that the generated sub-features can only be combined with other sub-features through their roots (which literal is a root of a given sub-feature can be determined using the language bias, informally, roots of sub-features correspond to roots of trees when features are viewed as graphs). In fact, in the feature construction process of RelF, any sub-feature can be connected to other sub-features only through a single variable in its root. This motivates Proposition 3. Before stating it, we define two auxiliary terms: *graft* and *domain*.

Given two sub-features ϕ and ψ, their *graft* w.r.t. variables $V_1 \in vars(\phi)$ and $V_2 \in vars(\psi)$ (denoted by $\phi \oplus_{V_1,V_2} \psi$) is a sub-feature $\phi\theta_1 \wedge \psi\theta_2$ where $\theta_1 = \{V_1/X\}$, $\theta_2 = \{V_2/X\}$ and X is a variable which is contained neither in ϕ nor in ψ. Informally, *grafting* two sub-features means merely joining them together through some of their variables. For example, when $\phi = atom(X, c)$ and $\psi = bond(Y, Z) \wedge atom(Z, h)$ then $\phi \oplus_{X,Y} \psi = atom(A, c) \wedge bond(A, Z) \wedge atom(Z, h)$.

Now, we define *domain*. Let φ be a sub-feature and $V \in vars(\varphi)$. Then domain $dom_V(\varphi, e)$ denotes the set of all terms $t \in terms(e)$ such that $e \models \varphi\theta$ where $\theta = \{V/t\}$. Intuitively, *domain* is the set of terms from an example which can be substituted (by a substitution θ) for a given variable V in sub-feature φ so that it would still be true w.r.t. that example, $e \models \varphi\theta$ in other words.

Proposition 3. *Let ψ_1 and ψ_2 be sub-features such that $vars(\psi_1) \cap vars(\psi_2) = \emptyset$. Let V_1 and V_2 be variables through which ψ_1 and ψ_2, respectively, can be connected to other sub-features according to a given language bias. Finally, let $\varphi_1 = \phi \oplus_{V_3,V_1} \psi_1$ and $\varphi_2 = \phi \oplus_{V_3,V2} \psi_2$ be features. If, for* *all examples e in a given dataset $\mathcal{E}$, it holds $dom_{V_1}(\psi_1, e) = dom_{V_2}(\psi_2, e)$ and if there is a real number c such that $\mathcal{S}(\psi_1\theta_{1,t}, e) = c \cdot \mathcal{S}(\psi_2\theta_{2,t}, e)$ for all examples $e \in \mathcal{E}$ and all substitutions $\theta_{1,t} = \{V_1/t\}$ and $\theta_{2,t} = \{V_2/t\}$ where $t \in dom(\psi_1, e) = dom(\psi_2, e)$ then φ_1 and φ_2 are redundant (relatively to each other).*

Proof: First, we can apply Proposition 2, which gives us:

$$\mathcal{S}(\varphi_1\theta_{1,t}, e) = \mathcal{S}(\phi\vartheta_t, e) \times \mathcal{S}(\psi_1\theta_{1,t}, e)$$

where $\vartheta_t = \{V_3/t\}$. The reason why we can do this is that $\varphi_1\theta_{1,t}$ is disconnected and can be decomposed as $(\phi\vartheta_t) \wedge (\psi_1\theta_{1,t})$. Next, we can get completely analogically the following:

$$\begin{aligned}\mathcal{S}(\varphi_2\theta_{2,t}, e) &= \mathcal{S}(\phi\vartheta_t, e) \times \mathcal{S}(\psi_2\theta_{2,t}, e) &= \\ &= \mathcal{S}(\phi\vartheta_t, e) \times (\tfrac{1}{c} \cdot \mathcal{S}(\psi_1\theta_{1,t}, e)) &= \\ &= \tfrac{1}{c} \cdot \mathcal{S}(\varphi_1\theta_{1,t}, e)\end{aligned}$$

This, together with the fact that we can get the sample sets $\mathcal{S}(\varphi_1, e)$ and $\mathcal{S}(\varphi_2, e)$ by computing the unions of the above sample sets over all $t \in terms(e)$, shows that φ_1 and φ_2 are indeed redundant (relatively to each other) according to how we defined redundancy. ■

The above proposition is used in the feature-construction algorithm to detect which sub-features will always give rise to redundant features no matter with what the algorithm combines them. As a consequence of this, the search space of features can be drastically reduced but all non-redundant features are still guaranteed to be constructed. Stated in an alternative way: the property of sub-features being redundant is monotone in our feature construction algorithm. This is a generalization of results from [5] to settings with polynomial relational aggregation.

VII. Polynomial Relational Features and Hybrid Markov Logic

In this section, we highlight relations between hybrid Markov logic [1] and polynomial relational features used in the transformation-based approach. Hybrid Markov logic (HMLN) is an extension of Markov logic to hybrid domains, i.e. to domains which contain both discrete relational data and numerical data. A hybrid Markov logic network is a set of pairs (F_i, w_i) where F_i is a first-order formula or a numeric term and $w_i \in R$. When one fixes a set of constants C then hybrid Markov logic defines a probability distribution over possible worlds as follows:

$$P(X = x) = \frac{1}{Z} \exp\left(\sum_i w_i s_i(x)\right)$$

where $s_i(x)$ is either number of true groundings of feature F_i w.r.t x if F_i is not a numeric-feature, or the sum of *values* w.r.t. x when F_i is a numeric feature. If one has conditional distributions represented as HMLNs for two classes and the task is to learn a classifier distinguishing examples from the two classes then the resulting decision boundary has equation (where $+$ and $-$ superscripts distinguish the weights and features of the models for the two classes)

$$\sum_i w_i^+ s_i^+(x) - \sum_i w_i^- s_i^-(x) = t$$

which defines a linear hyperplane. This form has already been exploited in the work on max-margin Markov logic [8]. If one used polynomial numeric features in the HMLN, the resulting classifier would become very similar to what we obtain with polynomial relational features and a linear classifier, e.g. support vector machine, the main difference being that $s_i(x)$ would be an *average* of values of polynomials applied on the ground instances of feature F_i whereas it is a *sum* in the case of HMLN. A convenient property of averages used in polynomial-feature framework is that they are not so sensitive to the size of the examples (possible worlds).

Seen from this perspective, the framework of polynomial relational features presented in this paper can be seen as being very close to a lifted hybrid Markov logic. Note, however, that there have been no lifted algorithms for HMLN to date. Furthermore, the polynomial relational features can be used in conjunction with almost any attribute-value classifier, not just the linear ones, thus possibly offering greater flexibility in discriminative classification. On the other hand, it is true that HMLNs are able to solve other problems than just discriminative classification.

VIII. Experiments

We evaluated performance of the polynomial-feature-based method in three relational learning domains. We compared it with Tilde [9] and with results from literature. We performed experiments with tree-like polynomial relational features with degree one, two and three in order to evaluate impact of degree of monomials on predictive accuracy. For each dataset, we used two types of relational descriptions with different complexity. We used random forest classifiers with 100, 500 and 1000 trees (see Table I).

Our first set of experiments was done on the well-known Mutagenesis dataset [10], which consists of 188 organic molecules marked according to their mutagenicity. We performed two experiments in this domain. In the first experiment, we used only information about bonds and their types (*single*, *double*, *triple*, *resonant*) and information about charge of atoms, but not about their types. In the second experiment, we also added information about atom types. The accuracies obtained by our method (Table'I) are consistently higher than the best accuracy 86% achieved by Tilde in [9]. The best results are obtained for monomial features of degree 3.

Our second set of experiments was performed on the NCI 786 dataset which contains 3506 molecules labelled according to their ability to inhibit growth of renal tumors. Again we performed two experiments in this domain. In the first experiment, we used only information about bonds and their types and information about charge of atoms and in the second experiment we also added information about atom types. Monomials of degree 3 turned out to be best for the

	Degree 1	**Degree 2**	**Degree 3**
	100/500/1000	100/500/1000	100/500/1000
Mutagenesis - Charge	88.8/88.3/88.3	88.8/88.8/88.3	**89.9/89.9/89.4**
Mutagenesis - Atoms + Charge	87.8/88.3/88.3	88.3/88.8/88.8	**89.9/89.9/89.9**
NCI 786 - Charge	61.0/61.0/61.0	66.5/66.5/66.8	**67.2/68.0/68.0**
NCI 786 - Atoms + Charge	70.3/70.1/69.9	70.5 **/70.8/70.8**	**70.6**/70.2/70.4
PD138/NB110 - Charge	**84.7/82.3/82.3**	81.0/79.5/81.0	81.0/79.8/79.8
PD138/NB110 - Propensities	82.7/84.7/84.3	83.9/84.7/84.3	**85.1/85.5/85.5**

TABLE I
ACCURACIES ESTIMATED BY 10-FOLD CROSS-VALIDATION USING TRANSFORMATION-BASED LEARNING WITH MONOMIAL FEATURES AND RANDOM FORESTS FOR DEGREES OF MONOMIAL FEATURES: 1, 2 AND 3.

first representation whereas monomials of degree 2 performed best for the second representation. Tilde did not perform well on this dataset, so at least, we compared our results with results reported in [5] for kFOIL (63.1), nFOIL (63.7) and RelF (69.6). The accuracies obtained with monomial features for the *atoms + charge* representation were consistently higher than these results.

Our third set of experiments dealt with prediction of DNA-binding propensity of proteins. Several computational approaches have been proposed for the prediction of DNA-binding function from protein structure. It has been shown that electrostatic properties of proteins are good features for predictive classification. A recent approach in this direction is the method of Szilágyi and Skolnick [11] who created a logistic regression classifier based on 10 features also including electrostatic properties. Our first model for predicting whether a protein binds to DNA used only distributions of charged amino acids in fixed-size windows and the secondary structure of the proteins. In our second model, we added also information about average *propensity* of amino acids in the fixed-size windows to bind to DNA which had been measured by Sathyapriya et al. [12]. The accuracies obtained by our method on the second model were consistently higher than 81.4% accuracy obtained by Szilágyi and Skolnick with logistic regression or 82.2% that we obtained using random forest on Szilágyi's and Skolnick's features. The best results were obtained for monomials of degree 3. Surprisingly, for the experiments using only electric charge, the highest accuracies were obtained by monomials of degree 1. Nevertheless, results obtained for all degrees of monomials were higher than 75.8% accuracy obtained by Tilde.

IX. CONCLUSIONS

We have presented a conceptually simple framework for relational learning with multivariate polynomial functions. We have shown how the polynomial relational features can be used for estimation of higher-order moments of distributions in the relational context, generalizing results on Gaussian logic as a by-product. We have also demonstrated how they can be used in predictive setting where we were able to obtain state-of-the-art accuracies using only limited amounts of numerical information.

ACKNOWLEDGMENT

This work was supported by the Czech Grant Agency through project 103/11/2170 *Transferring ILP techniques to SRL* and by the Czech Technical University in Prague through the student grant competition project SGS11/155/OHK3/3T/13.

REFERENCES

[1] J. Wang and P. Domingos, "Hybrid markov logic networks," in *Proceedings of the 23rd national conference on Artificial intelligence - Volume 2*. AAAI Press, 2008.

[2] O. Kuželka, A. Szabóová, M. Holec, and F. Železný, "Gaussian logic for predictive classification," in *ECML/PKDD: European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2011.

[3] M.-A. Krogel and S. Wrobel, "Transformation-based learning using multirelational aggregation," in *ILP '01: Inductive Logic Programming*, 2001.

[4] C. Vens, A. V. Assche, H. Blockeel, and S. Dzeroski, "First order random forests with complex aggregates," in *ILP: Inductive Logic Programming*, 2004, pp. 323–340.

[5] O. Kuželka and F. Železný, "Block-wise construction of tree-like relational features with monotone reducibility and redundancy," *Machine Learning*, vol. 83, no. 2, pp. 163–192, 2011.

[6] F. Rossi, P. van Beek, and T. Walsh, Eds., *Handbook of Constraint Programming*. Elsevier, 2006.

[7] N. Lavrač and P. A. Flach, "An extended transformation approach to inductive logic programming," *ACM Trans. on Comput. Logic*, vol. 2, no. 4, pp. 458–494, 2001.

[8] T. N. Huynh and R. J. Mooney, "Max-margin weight learning for markov logic networks," in *ECML/PKDD: European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases*, 2009, pp. 564–579.

[9] H. Blockeel, L. D. Raedt, and J. Ramon, "Top-down induction of clustering trees," in *ICML '98: International Conference on Machine Learning*, 1998, pp. 55–63.

[10] A. Srinivasan and S. H. Muggleton, "Mutagenesis: ILP experiments in a non-determinate biological domain," in *ILP '94: Inductive Logic Programming*, 1994, pp. 217–232.

[11] A. Szilágyi and J. Skolnick, "Efficient prediction of nucleic acid binding function from low-resolution protein structures," *Journal of Molecular Biology*, vol. 358, no. 3, pp. 922 – 933, 2006.

[12] R. Sathyapriya, M. S. Vijayabaskar, and S. Vishveshwara, "Insights into proteindna interactions through structure network analysis," *PLoS Comput Biol*, vol. 4, no. 9, p. e1000170, 09 2008.