Classification with Markov Logic Networks in the presence of domain knowledge

Liliana Mafalda Soares Fernandes
Email: liliana.fernandes@ist.utl.pt
Instituto Superior Técnico, Universidade de Lisboa
Lisbon, Portugal; October, 2017

***Abstract*—In this work, we study the problem of learning to classify a set of instances based on an available training set and domain knowledge represented through a taxonomy. Classification is a family of supervised machine learning algorithms that assigns the input as belonging to one of several pre-defined classes. Traditionally, classification approaches tended to fall into two largely separate strands: one focused on logical representations, and one focused on statistical ones. Logical approaches deal better with the complexity of the real world while the statistical approaches excel on dealing with the uncertainty that is present in any real applications. As the real world is complex and uncertain, we think is important to explore methods that can have both characteristics. Markov Logic Networks (MLNs) combine logic and probability by attaching weights to first-order clauses. By combining first-order logic (FOL) and probability we can use the advantages of both. MLNs have gained traction in the AI community in recent years because of this ability to combine the expressiveness of FOL with the robustness of probabilistic representations. We propose an approach to add domain knowledge, represented in taxonomies, in classification problems using MLNs to improve the performance of the algorithms.**

Keywords: Machine Learning, Classification, Taxonomies, Markov Logic Network

I. Introduction

Existing mining algorithms, particularly in classification, reached considerable levels of efficiency, and their extension to deal with more demanding data, such as data streams and big data, show their incontestable quality and adequacy to the problem. Despite their efficiency, their effectiveness on identifying useful information is somehow impaired, not allowing for making use of existing domain knowledge to focus the discovery [1].

Classification algorithms are the most commonly used machine learning methods [2]. Classification represents the family of supervised learning techniques where a set of dependent variables needs to be predicted based on another set of input attributes [3].

Data mining tasks are data-oriented, in which domain models are induced from data. The bulk of research in this field concentrates on inducing models from data. However, in some domains, human expertise forms an essential part of the corpus of knowledge needed to construct models of the domain. The discipline of knowledge engineering has focused on encoding the knowledge of experts in a form that can be encoded into computational models of a domain. Nowadays, knowledge engineering and machine learning remain largely separate disciplines, yet, in many fields, substantial human expertise exists alongside data. When both data and domain knowledge are available, why not use these two resources to improve the effectiveness of the classification [4]? The use of domain knowledge can bring significant benefits to machine learning applications, by resulting in simpler and more interesting and usable models. However, most of existing approaches are concerned with being able to mine specific domains, and therefore are not easily reusable, instead of building general algorithms that are able to incorporate domain knowledge, independently of the domain [5, 6, 7].

One problem that we can encounter in classification is, for building classifiers with good accuracy, we need a sufficiently large and representative training dataset. Because such data is not always available, this can be partially countered by domain knowledge. Domain knowledge may be related to examples not present in the available domain dataset and thus may improve the generality and robustness of classifiers induced on such datasets [8].

The first effort to use domain knowledge in machine learning was undoubtedly made through Inductive Logic Programming (ILP for short), in this kind of approach, in addition to the training set, an encoding of the known background knowledge is also provided. An ILP system will then derive a logic program as an hypothesis, which entails all the positive and none of negative examples. The fact that all information must be written in declarative languages (like Prolog and Datalog) is one of the drawbacks of ILP, and one of the reasons for not being widely used. Nevertheless, its structure promotes the representation and use of domain knowledge. ILP techniques must also deal with the tradeoff between expressiveness and efficiency of the used representations. Studies show that current algorithms would scale relatively well as the amount of background knowledge increases. But they would not scale, at all, with the number of relations involved, and in some cases, with the complexity of the patterns being searched [1]. Further, traditional ILP is unable to cope with the uncertainty of real-world applications such as missing or noisy data, a known drawback when compared to the statistical approach [9].

Pure logical approaches handles complexity but they don't scale very well and can't deal with the uncertainty in its

original form. On the other hand, pure statistical approaches can scale very well and deal with uncertainty but in some cases produce models that can't be represented in a way that is easy for humans to understand [10]. For automated classification methods to be adopted in practice, it is crucial that a relationship of trust may be established between domain experts and the models generated. When the cost of making mistakes is very high, numerical validation is usually not enough. This is why it is so important that the generated models are simple, understandable, and somewhat aligned with certain facts that are known to be true in the domain [9].

Statistical approaches also ignore the complexities of the real world. First, it is not possible to express or make use of existing domain knowledge, to explicitate relationships between attributes or hierarchies of features. And second, it does not allow for constraining the results according to facts which are known to be true, even if not represented in the subset of data being fed to the learning algorithm. As the real world is complex and uncertain the idea is to reach a middle ground and use logic to handle the complexity of existing domain knowledge and statistics to handle the uncertainty and noise in the observations [9].

There is not a lot of research on the use of domain knowledge in machine learning tasks, particularly in classification problems. One example of a work related to the use of domain knowledge in learning is the work of Nazeri and Bloedorn [11] in which the authors, presented a method to improve the quality of discovered rules by A-priori (associations) [12] and C4.5/J48 (decision Tree) [13] algorithms applied to data from aviation safety and intrusion detection domains, incorporating available domain knowledge into the learning techniques, by reducing the number of uninteresting rules. Barracosa and Antunes [14] proposed a new methodology to anticipate teachers performance based on past survey results. In their approach they demonstrate that they could improve classification accuracy, with decision trees (algorithm C4.5/J48), by identify a set of meta-patterns, that are useful for introducing some knowledge to enrich the source data. Bochare et al. [15] proposed an approach to incorporate domain knowledge into a model for predicting risk of breast cancer in post menopausal women using genomic data, family history, and age. This enriched dataset was used with four different classifiers (C4.5/J48 decision tree algorithm, Naïve Bayes [16], SVM [17] and Bayesian networks [18]). Pardel et al. [19] proposed an approach for automatically identifying organs from medical CT images using domain knowledge. More specifically, they proposed an algorithm for the classification of chest organs (trachea, lungs, bronchus) using a decision tree.

There are also some works in classification with domain knowledge represented by taxonomies. A taxonomy is as a vocabulary, in which each term usually has hierarchical relationships, which means that a taxonomy imposes a topical structure (a tree-like structure) on information. This hierarchical relationships are called IS-A (sub-class-of) relationship which represents a relation between a generalized concept and a specialized one [20]. One example that uses taxonomies as the representation for the domain knowledge is the work of Zhang et al. [21], in which the authors proposed an algorithm, that is a generalization of the Naïve Bayes Learner for learning classifiers from attribute value taxonomies and data. Another example of the use of taxonomies as the domain knowledge representation is the work proposed by Vieira and Antunes [9], a hierarchy guided decision tree learning algorithm, an extension of standard decision tree learning algorithm, that is able to take advantage of user supplied feature (or attribute value) hierarchies.

There are also some works in classification with domain knowledge using instead of taxonomies, ontologies. Ontologies specify the primary concepts and the relationships among the concepts in a standardized machine readable format [22]. The relationship between two concepts in an ontology can be of any kind, such as "is a part of", "is an instance of", "is a type of", "is a product of" and so on. In this sense it's more expressive than a tree structure taxonomy, because it can specify the exact nature of the relationship between the two concepts being described. One example of the use of ontologies as domain knowledge representation in classification problem is the work proposed by Zhang et al. [23]. The authors presented an ontology-driven decision tree learning algorithm to learn classification rules at multiple levels of abstraction. Łukaszewski and Wilk [24] proposed an approach to the problem of the classification using the Naïve Bayes algorithm, with test costs understood as the cost of obtaining attribute values of classified examples. On medical diagnosis, it is not viable to experiment all tests to decide which tests should be carried out in order to control the tradeoff between the cost of these tests and the accuracy of a classifier. So the appropriate approach may be reducing the cost of the required tests while maintaining the prediction accuracy of a classifier. The authors assumed that attribute values are represented at different levels of abstraction. These levels model domain background knowledge and in order to formally represent this, the authors introduced an attribute value ontology. Vieira and Antunes [9] also proposed a method that adds domain knowledge, represented in web ontology language 2 [25], to a purely statistical decision tree learner.

The majority of the works presented previously were done using Naïve Bayes and decision tree algorithms but there are also examples of adding domain knowledge to classification problems with Neural Networks. One example is the work of Tan [26], where the author proposed a hybrid system termed cascade adaptive resonance theory mapping that incorporates symbolic knowledge into neural-network learning and recognition. Teng et al. [27] proposed an approach to integrate domain knowledge with Reinforcement Learning using a self-organizing neural network. Shu et al. [28] developed a deep network structure, capable of transferring labelling information across heterogeneous domains, especially from text domain to image domain. Zhou et al. [29] proposed a framework for event trigger identification. Event trigger identification is the detection of the words describing the event types, and is a crucial and prerequisite step in the pipeline process of biomedical event extraction. Biomedical event extraction is the process of automatically detecting description of molecular interactions in research articles. The authors proposed a framework that works as follows: first, scientific publications from Medline

are crawled to form a corpus where domain knowledge can be obtained. Then a neural language model is built from such a corpus using unsupervised learning.

It is our belief that with the introduction of domain knowledge we can built simpler classification models and at the same time, generalize better. However, to use these models, it is fundamental that we can understand the reasoning beyond the predictions, for this reason it is important to use human interpretable models.

As already mentioned, logical approaches tend to emphasize handling complexity, and statistical ones uncertainty. Markov Logic Networks (MLNs) are a simple approach to combine first-order logic and probabilistic graphical models in a single representation. By combining FOL and probability we can use the advantages of both [10]. MLNs have gained traction in the AI community in recent years because of this ability to combine the expressiveness of FOL with the robustness of probabilistic representations [10].

In this work we propose an approach that introduces domain knowledge, represented as a taxonomy, in the process of learning a model in the context of classification using MLNs. The main contribution of this work is a methodology that enables a MLN to take advantage of user supplied feature (attribute value) taxonomies and learn a model that is able to deal with data specified at different levels of abstraction (or in other words, at different levels in the attribute taxonomies). Our results have shown that this methodology improves the performance of the algorithm. We called the resulting MLN model, that can take advantage of user supplied feature (attribute value) taxonomies, Hierarchy based Markov Logic Network (HMLN).

In addition to an approach to add domain knowledge as taxonomies we also propose an approach to add domain knowledge as rules extracted from a tree obtained with a DT algorithm. In cases where it is easy to extract rules with good support, our results have shown that this methodology also improves the performance of the algorithm. However, as observed in our work, this is not always the case. In cases where it is not easy to extract rules with good support from a tree, using the best rules found in the literature, does not lead to better results.

This document is organized as follows: in section II we discussed in more detail Markov Logic Networks. In sections II-C and II-D we present the software used in this work and how to transform a tabular dataset to be used with MLNs, respectively. In section III we described our approach, the HMLN. In section IV we present and discuss the results obtained. Finally, in section V we present the conclusions of this work and some ideas for future work.

II. Markov Logic Networks

Markov Logic Networks (MLNS) were proposed by Richardson and Domingos [30] and are an approach to combine first-order logic and probabilistic graphical models in a single representation. Before MLNs, the approaches to do this, typically focused on combining probability with restricted subsets of first-order logic, like Horn clauses, frame-based systems or database query languages but, were often quite complex [30]. In contrary, the approach proposed by [30] is simple and yet combines probability and first-order logic with no restrictions other than finiteness of the domain.

A Markov Logic Network is a first-order knowledge base with a weight attached to each formula and can be viewed as a template for construct Markov Networks, because, once those formulas are grounded to the constants this becomes a Markov Network [30].

A first-order Knowledge Base (KB) can be seen as a set of hard constraints on the set of possible worlds: i.e., if a world violates even one formula, it has zero probability. The basic idea in MLNs is to soften these constraints: when a world violates one formula in the KB it is less probable, but not impossible. The fewer formulas a world violates, the more probable it is. Each formula has an associated weight that reflects how strong a constraint is: the higher the weight, the greater the difference in log probability between a world that satisfies the formula and one that does not, other things being equal. From the point of view of probability, MLNs provide a compact language to specify very large Markov networks, and the ability to flexibly and modularly incorporate a wide range of domain knowledge into them [30]. More formally, as described in [30]:

Definition 1: (***Markov Logic Network***) *A Markov logic network L is a set of pairs (F_i, w_i), where F_i is a formula in first-order logic and w_i is a real number. Together with a finite set of constants $C = \{c_1, c_2, \ldots, c_{|C|}\}$, it defines a Markov network $M_{L,C}$ as follows:*

1) *$M_{L,C}$ contains one binary node for each possible grounding of each predicate appearing in L. The value of the node is 1 if the ground atom is true, and 0 otherwise.*
2) *$M_{L,C}$ contains one feature for each possible grounding of each formula F_i in L. The value of this feature is 1 if the ground formula is true, and 0 otherwise. The weight of the feature is the w_i associated with F_i in L.*

The graphical structure of $M_{L,C}$ follows from definition 1: there is an edge between two nodes of $M_{L,C}$ iff the corresponding ground atoms appear together in at least one grounding of some formula in L [30]. To better visualise this, consider the follow example: we want to build a model to classify if a mushroom is poisonous or not. And we have a first-order KB that, for simplicity, has only the two formulas presented in (1), and adapted from [31].

$$\begin{aligned} SporePrintColor(x, green) &\Rightarrow Class(x, poisonous) \\ Habitat(x, leaves) \wedge CapColor(x, white) &\Rightarrow Class(x, poisonous) \end{aligned} \tag{1}$$

The first step in a MLN is the grounding of the formulas presented in the KB. Figure 1 shows the graph of the ground Markov network defined by the formulas in (1) and the constants mushroom A (A) and mushroom B (B). Each node in this graph is a ground atom (e.g., CapColor(Mushroom A, white)). The graph contains an arc between each pair of atoms that appear together in some grounding of one of the formulas. $M_{L,C}$ can now be used to infer the probability that Mushroom

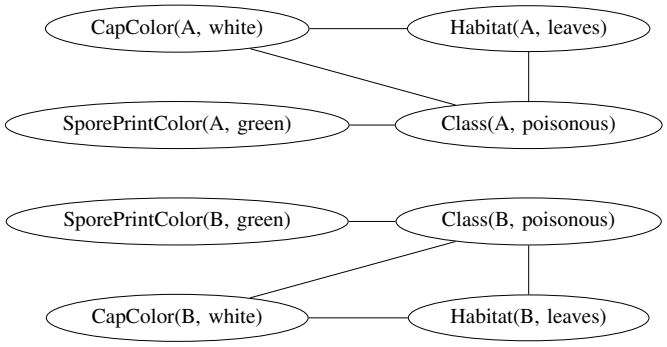


Fig. 1: Ground Markov network obtained by applying the formulas in (1), to the constants Mushroom A (A) and Mushroom B (B).

A and Mushroom B are poisonous given their cap color, their spore print color and their habitat.

Each state of $M_{L,C}$ represents a possible world. A possible world is a set of objects, a set of functions (mapping from tuples of objects to objects), and a set of relations that hold between those objects; together with an interpretation, they determine the truth value of each ground atom.

The possible groundings of a predicate in Definition 1 are obtained simply by replacing each variable in the predicate with each constant in C, and replacing each function term in the predicate by the corresponding constant.

A. Inference

In a inference problem, we want to find the most probable state of the world given some evidence. This is known as Maximum A Posteriori probability (MAP) inference in the Markov network literature [32]. In Markov logic this reduces to finding the truth assignment that maximizes the sum of weights of satisfied clauses. This can be done using any weighted satisfiability solver. Domingos et al. [32] have successfully used *MaxWalkSAT*, a weighted variant of the *WalkSAT* local-search satisfiability solver, which can solve hard problems with hundreds of thousands of variables in minutes. *MaxWalkSAT* performs a stochastic search by picking an unsatisfied clause at random and flipping the truth value of one of the atoms in it. With a certain probability, the atom is chosen randomly; otherwise, the atom is chosen to maximize the sum of satisfied clause weights when flipped. This combination of random and greedy steps allows *MaxWalkSAT* to avoid getting stuck in local optima while searching [32].

B. Training

Once the grounded Markov Network is constructed, the next step in a MLN, is to calculate the weights associated to each formula in the KB. There are two approaches to weight learning in MLNs: generative and discriminative. In discriminative learning, we know a priori which predicates will be used to supply evidence and which ones will be queried [33]. In many applications, like in classification problems, we know a priori which atoms will be evidence and which ones will be queried, and the goal is to correctly predict the latter given the former. If we partition the ground atoms in the domain into a set of evidence atoms X and a set of query atoms Y, the conditional likelihood of Y given X is [34]:

$$\begin{aligned} P(y|x) &= \frac{1}{Z_x} \exp\left(\sum_{i \in F_Y} w_i n_i(x,y)\right) \\ &= \frac{1}{Z_x} \exp\left(\sum_{j \in G_Y} w_j g_j(x,y)\right) \end{aligned} \tag{2}$$

where F_Y is the set of all MLN clauses with at least one grounding involving a query atom, $n_i(x,y)$ is the number of true groundings of the ith clause involving query atoms, G_Y is the set of ground clauses in $M_{L,C}$ involving query atoms, and $g_j(x,y) = 1$ if the jth ground clause is true in the data and 0 otherwise. The gradient of the conditional log-likelihood is

$$\begin{aligned} \frac{\partial}{\partial w_i} \log P_w(y|x) &= n_i(x,y) - \sum_{y'} P_w(y'|x) n_i(x,y') \\ &= n_i(x,y) - E_w\Big[n_i(x,y)\Big] \end{aligned} \tag{3}$$

Computing the expected counts $E_w[n_i(x,y)]$ is however intractable in all but the smallest domains, since counting the number of true groundings of a first-order clause in a database is #P-complete [34]. In large domains, the number of true groundings of a formula may be counted approximately, by uniformly sampling groundings of the formula and checking whether they are true in the data [35]. The expected counts can then be approximate from a small number of Markov chain Monte Carlo (MCMC) samples. MCMC is a technique for numerical integration using random numbers that draws samples from the required distribution and then forms sample averages to estimate the desired probability, using Markov Chains [36]. A Markov chain X is a discrete time stochastic process $\{X_0, X_1, ...\}$ with the property that the distribution of X_t given all previous values of the process, $X_0, X_1, ..., X_{t-1}$ only depends upon X_{t-1} [37]. MCMC draws samples by running a cleverly constructed Markov chain for a long time [36]. The MCMC algorithm typically used is the Gibbs sampling [36], but for MLNs the much faster alternative MC-SAT is available [38]. MC-SAT is an MCMC algorithm proposed by Poon and Domingos [39] that is able to handle deterministic and near-deterministic dependencies by using Wei et al. [40] *SampleSAT* as a subroutine to efficiently jump between isolated or near-isolated regions of non-zero probability, while preserving detailed balance [39].

MC-SAT applies slice sampling to Markov logic. Slice sampling is an alternative to Gibbs sampling that is often easier to implement than Gibbs sampling and more efficient (more information about slice sampling can be found in [41]). Using *SampleSAT* a hybrid strategy proposed by Wei et al. [40] which interleaves simulated annealing steps and *WalkSAT* steps, to sample a new state given the auxiliary variables. Simulated annealing is a probabilistic technique for approximating the

global optimum of a given function and can also be used as a method for sampling [40]. Detailed information about the MC-SAT algorithm can be found in [38].

C. Software and libraries used

In this work we use Python using the *matplolib*, *pandas* and *numpy* libraries. Particularly to run the MLNs we used the *Alchemy* 2.0 Software. *Alchemy* is a open-source software package for inference and learning in MLNs [42]. *Alchemy* was developed by the same research group that proposed the MLNs.

We run our work on one core of an Intel(R) Xeon(R) CPU E5-2680 v2 @ 2.80GHz.

D. Transforming the dataset to First-Order Logic

As previous discussed a MLN is a first-order KB with a weight attached to each formula. To use MLN in our work we need to have a KB, in other words, predicates and formulas. Since the datasets used in this work are tabular we had to have a way to convert them in to predicates and formulas. We used the methodology proposed by Silva and Cozman [43]. The same methodology is used in one of the examples in the *Alchemy* website [42]. In this methodology first we convert the attributes (including the class attribute) into representative FOL predicates. In figure 2, for example, for the attribute *Odor*, we create a predicate called $Odor(row, value\_odor!)$. The "!" operator in the *Alchemy* allows one to specify variables that have mutually exclusive and exhaustive values. In this case, it means that any row has exactly one value for the *Odor* attribute.

After convert the attributes to representative FOL predicates we need to build conjunctions between pairs of class predicate and attribute predicates. For example, for the attribute *Odor* we create a conjunction between this attribute and the class attribute resulting in the following formula: $Class(row, +value\_class) \wedge Odor(row, +value\_odor)$. The meaning of the "+" operator in the *Alchemy* is as following: a weight is learned for each combination of the values in the ground formulas for the variables preceded by this operator. With this we will have a formula for all of the combinations between all the possible values for the attributes *Odor* and *Class*.

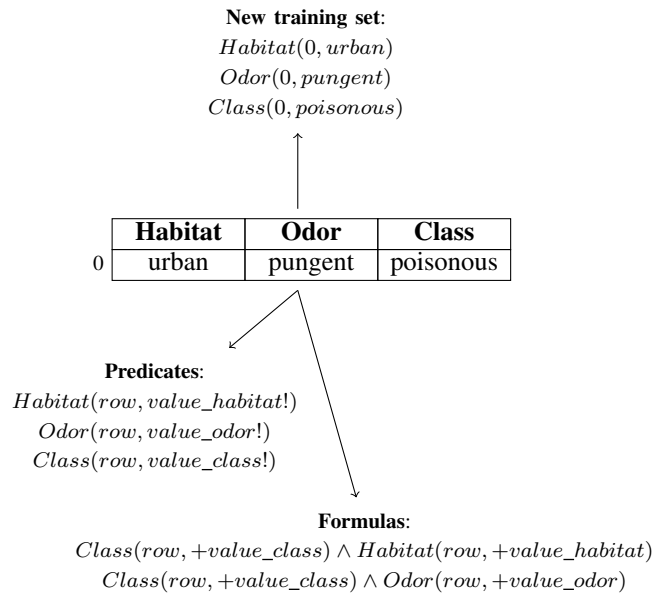


Fig. 2: Example of how to obtain the formulas and predicates from a tabular dataset as well how to transform the train set to be used in *Alchemy*.

III. Hierarchy based Markov Logic Network

The goal of our work is to add domain knowledge to our model to improve it's performance using MLNs and so, we need to develop a method to do it. For simplicity reasons, we decided to use taxonomies as the representation of the domain knowledge.

Our strategy to add the information from attribute taxonomies is, to iterate over all ground predicates in the training set and, if the respective attribute related to that predicate has a taxonomy, add to the training set a new ground predicate that have as the constant value, the value of its parent in the taxonomy. For example, if in our training set we have the ground predicate $Odor(0, pungent)$ and "bad" is the parent of "pungent" we add the ground predicate $Odor(0, bad)$ to the training set.

After adding the taxonomies, we train a MLN with the modified training set. When we introduced in section II-D how to transform our tabular data to FOL, we said that the predicates will have a "!" operator. This tells the *Alchemy* that each row as only one value for the predicate in question. In our methodology, to add the taxonomies to the training set we are basically adding a new predicate with the exact same row but a different value. What we actually do is remove the "!" operator before doing the training and add this operator again before performing the inference process. By doing this we will learn the weights for the formulas with concrete and also abstract values.

After learning the weights, we have to decided if we want to abstract or not the attributes before performing inference. In Markov Logic each formula has an associated weight that reflects how strong a constraint it is: the higher the weight, the greater the difference in log probability between a world that satisfies the formula and one that does not, other things being equal [30]. The weight of a formula may also be negative, which effectively means that the negation of the formula is likely to hold [44]. Or in other words, negative weights correspond to rules that are usually wrong, zero weights correspond to rules that have no influence and positive weights correspond to rules that are usually true [45]. The negative weights are handled by the propriety that a clause with weight $w < 0$ is equivalent to its negation with weight $-w$, and a clauses negation is the conjunction of the negations of all of its literals. Thus, instead of checking whether the clause is satisfied, we check whether it's negation is satisfied [39].

Having this in mind our criterion to choose if we abstract or not is to compare the average weight value for each level (leaves, level 1, level 2, etc) in the attribute taxonomy and choose the level that have the highest average weight value. We do this for each attribute so attributes can and probably will be, at different levels of abstraction.

In our approach, if we decide to abstract an attribute we remove the formulas with concrete values and increase the weights of the formulas with abstract values (formulas with values corresponding to the parents of the attribute taxonomy), according to the information gain [46] for the attribute in question. Initially, we simply removed the formulas with concrete values but would not always produce better results than the standard MLN.

The approach that worked the best was when we abstract an attribute and has a high information gain (we considered a value higher than 0.6). In this case, we need to adjust the weight to reflect it's importance. The way we do this is by multiplying the weight of the formula with abstract value by the number of formulas with values that corresponds to it's children in the taxonomy. To better explain this, let's assume that we decide to abstract the attribute $Odor$ and we have the following formulas: $w_1\ Class(row, poisonous) \wedge Odor(row, bad)$, $w_2\ \ Class(row, poisonous) \wedge Odor(row, musty)$ and w_3 $Class(row, poisonous) \wedge Odor(row, pungent)$. "Bad" is the parent of both "pungent" and "musty" in the $Odor$ taxonomy. Since we decide to abstract the attribute $Odor$ we are going to remove $w_2\ Class(row, poisonous) \wedge Odor(row, musty)$ and $w_3\ Class(row, poisonous) \wedge Odor(row, pungent)$ from the KB that we are going to use to perform the inference step. Let's also assume that $Odor$ has a information gain value higher than 0.6 in our training set. What we do is multiply the weight of the formula with the abstract value "bad" by two, i.e. the number of formulas that have as values children of the value "bad" in the $Odor$ taxonomy. Thus, in the KB only one formula will remain: $w_1^*\ Class(row, poisonous) \wedge Odor(row, bad)$ with $w_1^* = w_1 \times 2$.

After this step in addition to having the updated KB we have a dictionary with the attributes that were chosen to be abstracted and also the corresponding level of abstraction. Before performing inference, we need to transform the test set because we have the value at leaves's level of the attributes taxonomies. We iterate over each ground predicate and replace the value of the predicate accordingly. For example, if we have in the test set, the ground predicate $Odor(0, pungent)$ and the attribute $Odor$ was chosen to be abstracted at level one in which the parent for the value "pungent" is "bad", we replace that ground predicate by $Odor(0, bad)$.

IV. Experimental results

In this chapter we present our experimental results. As mentioned previously, we tested two approaches to add domain knowledge to MLN models. One of this approaches, introduces domain knowledge, represented as a taxonomy, in the process of learning a model in the context of classification using MLNs. We called the this MLN model, that can take advantage of user supplied feature (attribute value) taxonomies, Hierarchy based Markov Logic Network (HMLN) as discussed in III. The results of this approach are presented in IV-A.

In addition to an approach to add domain knowledge as taxonomies we also propose an approach to add domain knowledge as rules extracted from a tree obtained with a DT algorithm. The results of this approach are presented in IV-B

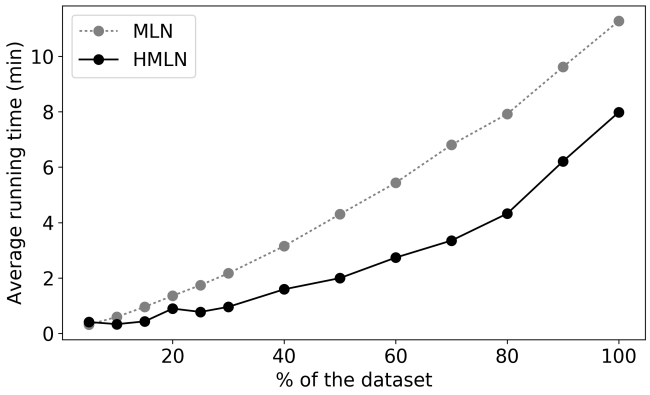
In our experiments we used the Mushroom and Nursery Datasets available in UC Irvine Machine Learning Repository. In the Mushroom Dataset the goal is to classify if a mushroom is edible or poisonous. This dataset has 22 attributes (one of this attributes is called Class and is what we want to predict) and 5644 instances (after removing the rows with missing values). The goal of the Nursery Dataset is to classify a application for a nursery school in one of these values: not recommend, recommend, very recommend, priority or special priority. This dataset has 9 attributes (one of this attributes is called Class and is what we want to predict) and 12960 instances (there are no missing values).

The taxonomies used in this work were kindly provided by Vieira and Antunes [9]. In the case of the Mushroom Dataset, for the 21 input attributes we have 14 attributes with a corresponding taxonomy. For the Nursery Dataset, from the 8 input attributes we have 6 attributes with a corresponding taxonomy.

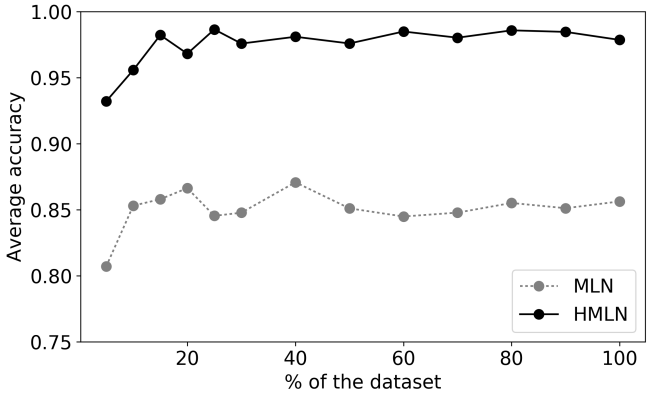
A. Hierarchy based Markov Logic Network

In this section we present the results for the approach, that we called Hierarchy based Markov Logic Network (HMLN), in which the Knowledge Base (KB) is built using conjunctions between the input attributes and the class attribute as the formulas in the KB and, the attributes taxonomies are added to the model. In figures 3 and 4, we present the results for both standard MLN and HMLN for the Mushroom and Nursery Datasets, respectively. We can see that the average accuracy of the models when we add the attribute taxonomies are always better. We can also observe that contrary to our expectations, adding taxonomies results in a decrease of the total running time. Our intuition was that adding the taxonomies would result in increased running times, since we are adding more predicates to our training set (as discussed in section III) and this results in more clauses (conjunctions between the attributes and the class attribute) that we have to learn the corresponding weight.

We have more clauses but the same number of class predicates for both HMLN and standard MLN models, in the training step. Remembering, in our training sets we have predicates related to the class attribute and predicates related to the input attributes, attributes that we want to use to build a model to predict the class attribute value. In section II-B, we describe that in learning the clauses weights we partition the ground predicates in the domain into a set of evidence predicates and a set of query predicates (in our case this set have only one element the predicate *Class*). Our set of query predicates is the same for both standard MLN and HMLN models but the set of evidence predicates is different because we add predicates with values corresponding to the attribute

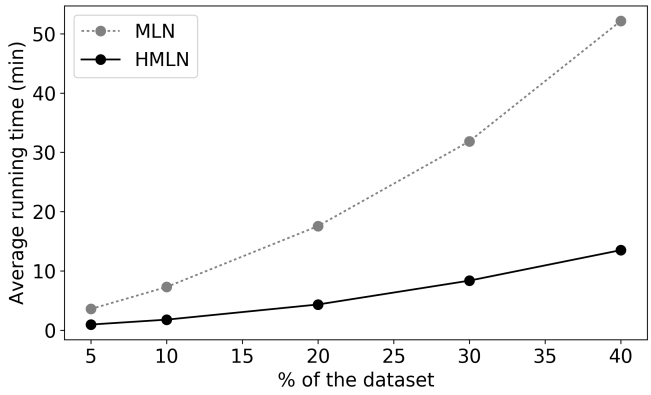


(a)

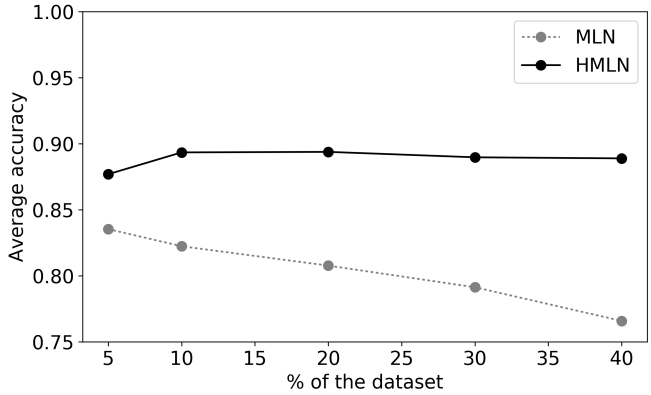


(b)

Fig. 3: a) Average running time b) Average accuracy in the test set versus percentage of dataset used to built the MLN model for the Mushroom Dataset. The grey line is the results when adding taxonomies (HMLN models) and the black line is the results without adding taxonomies (MLN models).



(a)



(b)

Fig. 4: a) Average running time b) Average accuracy in the test set versus percentage of dataset used to built the MLN model for the Nursery Dataset. The grey line is the results when adding taxonomies (HMLN models) and the black line is the results without adding taxonomies (MLN models).

parents in the attributes taxonomies. To learn the weights, the default algorithm used by *Alchemy* is the MC-SAT. What we observe in our experiments is that in the case of HMLN models the time that it takes to finish performing the MC-SAT is much less than in the case of the MLN models. We believe that by having a greater number of evidence predicates while maintaining the same query predicates allows MC-SAT algorithm to more quickly converge.

B. Knowledge Base consisting of rules extracted from Decision Trees

In the previous section, we presented our approach, which we called HMLN, to add the attributes taxonomies to MLNs models to improve their performance and showed the effectiveness of our approach. In this approach, the KB is composed by formulas which are conjunctions between the input attributes and the class attribute. After training what we will have is a formula for each possible combination of values for the input attribute and class attribute that is present in the training set. The number of formulas in our KB grows exponentially with the possible values of the attributes. And thus, we could end up with a MLN that have a lot of formulas in the KB for which we need to learn a weight and in return will take much more time to compute. So, ideally we would like to have a KB with a fixed number of formulas made by domain experts. Since we don't have such KB, we decided to mimic this by using the Decision Tree (DT) algorithm and extract two or three rules using the Mushroom Dataset. We used two different DT implementations to test this approach: the C4.5/J48 implementation using the *Weka* Software [47] and the CART implementation using the machine learning library for Python *scikit-learn* [48].

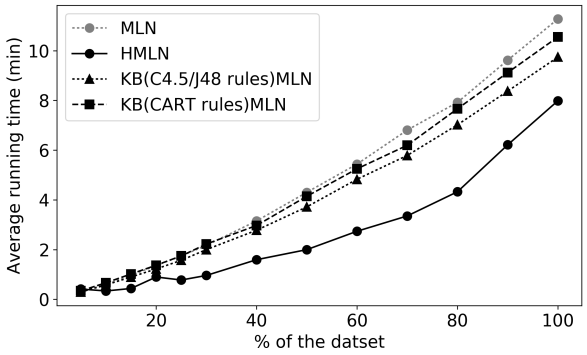
The first step in both train and inference in MLN is to convert all the formulas to the clausal normal form (information about how to convert a formula to clausal normal form can be found in [49]). Using the DT C4.5/J48 implementation we extracted three rules that when converted to clausal normal results in six formulas. Using the DT CART implementation we extracted two rules that when converted to clausal normal results in seven formulas. This number is independent of the train set size and is a lot less clauses, for which we have to learn the weight, when compared to the approach of using as formulas in the KB, conjunctions between input attributes and class attribute. With this approach, with the HMLN, in average, we had 257 clauses and with the standard MLN model we had 191 clauses.

In figure 5 b), we can observe the results when using rules extracted form a tree obtained with the DT C4.5/J48 implementation and the the DT CART implementation as the formulas of the KB, that we called *KB(C4.5/J48 rules)MLN* and *KB(CART rules)MLN*, respectively. Both *KB(C4.5/J48 rules)MLN* and *KB(CART rules)MLN* models are always better than the standard MLN in which the KB was built using conjunctions between the input attributes and the class and, the HMLN when using ten or less percent of the dataset. Using fifteen or more percent of the dataset the HMLN is slightly better than *KB(C4.5/J48 rules)MLN*. The *KB(CART rules)MLN* model is always slightly better than the HMLN. We were expecting that since we have a lot less clauses in the *KB(C4.5/J48 rules)MLN* and *KB(CART rules)MLN* models they would be much faster than the standard MLN in which the KB was built using conjunctions between the input attributes and the class and, also faster than HMLN but observing 5 a) this is not the case. Both *KB(C4.5/J48 rules)MLN* and *KB(CART rules)MLN* models are slightly faster than the standard MLN, but the HMLN is still much faster. It seems that the impact that the attribute's taxonomies have in helping the weight learning algorithm to converge faster is much more than the impact of having less clauses.

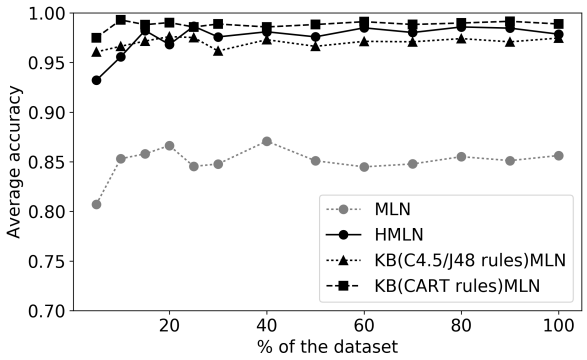
In our work, we also tested this approach with the Nursery Dataset. But, the trees that resulted were much more complex than the ones obtained for the Mushroom Dataset so, finding rules with good support is more difficult. We used then, rules from the literature as formulas in the KB. However the models in which the KB were built with these rules were always worse than the standard MLN.

We could think of this approach of, using a DT model to build a tree and extract rules to be used as formulas in the KB of a MLN model, as another method of adding domain knowledge to improve the performance of the algorithm because ideally the formulas would be made by a domain expert and probably they would be very similar to the ones that we obtained.

Our approach here consists in, first, obtaining a set of rules by using a DT learner and then using these rules to augment the training set that is fed to the MLN learner. This kind of strategy is known as an ensemble that is one of the most active areas of research in machine learning [50]. An ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way to classify new examples [50].



(a)



(b)

Fig. 5: a) Average running time b) Average accuracy in the test set versus percentage of dataset used to build the MLN model for the Mushroom Dataset. The grey line is the results when adding taxonomies and the formulas in the KB are conjunctions between attributes and the class attribute. The black line with dots is the results without adding taxonomies and again the formulas of the KB are conjunctions. The black line with squares are the results when using a DT C4.5/J48 implementation to generate rules to use as formulas in the KB. And finally, the black line with triangles are the results when using a DT CART implementation to generate rules to use as formulas in the KB

V. Conclusions

Two key challenges in most machine learning applications are uncertainty and complexity. Logic, especially, first-order logic (FOL) provides an expressive, compact and elegant way to express domain knowledge. Probabilistic methods are better at handling the uncertainty and noise in real data. However, both of these are necessary to build intelligent systems and handle real-world applications. Markov Logic Networks (MLNs) combine logic and probability by attaching weights to first-order clauses. By combining FOL and probability we can use the advantages of both. MLNs have gained traction

in the AI community in recent years because of this ability to combine the expressiveness of FOL with the robustness of probabilistic representations.

The goal of this work was to extend the work by Vieira and Antunes [9], by introducing domain knowledge, represented through taxonomies, in the process of learning a set of MLNs to improve the models performance. The use of domain knowledge can bring significant benefits to machine learning applications, by resulting in simpler and more interesting and usable models.

In this work we proposed an approach, an extension of the works of Vieira and Antunes [9] and Richardson and Domingos [30], that introduces domain knowledge, represented as a taxonomy, in the process of learning a model in the context of classification using MLNs. We called this MLN model, that can take advantage of user supplied feature (attribute value) taxonomies, Hierarchy based Markov Logic Network (HMLN).

In addition to an approach to add domain knowledge as taxonomies we also propose an approach to add domain knowledge as rules extracted from a tree obtained with a Decision Tree (DT) algorithm.

Our results have shown that HMLNs, outperforms the standard MLN model and is in fact a lot more faster. The approach add domain knowledge as rules extracted from a tree obtained with a Decision Tree (DT) algorithm, in cases where it is easy to extract rules with good support, our results have shown that this models also outperforms the standard MLN model. However, as observed in our work, this is not always the case. In cases where it is not easy to extract rules with good support from a tree, using the best rules found in the literature, does not lead to better results.

We also concluded that, despite the fact than when we can extract rules from a DT, with good support, using ten or less percent of the dataset the approach of using rules extracted from tree to populate the Knowledge Base (KB) is better than the HMLN, in general, there isn't much difference in the accuracy for both approaches. And being the HMLN more faster we can conclude that HMLN approach is better than using a DT to extract some rules and use them as formulas in the KB.

We validate our work using the datasets Mushroom and Nursery, available in UC Irvine Machine Learning Repository since we already had the attributes taxonomies that Vieira and Antunes [9] used in their work. Since these datasets are both tabular and in MLNs we work with formulas and predicates in FOL we also presented an approach to extracted from a tabular dataset formulas and predicates for to be possible to use a tabular dataset with our approach.

To conclude, we can observe in our work that using domain knowledge with MLNs improves the model's accuracy.

VI. Future Work

Although the results presented have demonstrated the effectiveness of our approach, it can, of course, be further developed in a number of ways.

One of the things that would be interesting to explore is the use of FOL, that allows greater expressiveness, to add domain knowledge to the MLN. In Vieira and Antunes [9]'s work they also use ontologies as the domain knowledge representation and this can be good starting point to explore adding domain knowledge that cannot be represented simply as taxonomies to MLNs.

Another important work to be done is to explore in more detail why adding the attributes taxonomies helps the model converge much faster.

Finally, it would be interesting to test our approach on problems with a richer, more expressive domain than the tabular datasets we used here.

References

[1] C. Antunes and A. Silva, "New trends in knowledge driven data mining." in *ICEIS (1)*, 2014, pp. 346–351.

[2] N. Dogan and Z. Tanrikulu, "A comparative analysis of classification algorithms in data mining for accuracy, speed and robustness," *Information Technology and Management*, vol. 14, no. 2, pp. 105–124, 2013.

[3] O. Maimon and L. Rokach, *Data mining and knowledge discovery handbook*. Springer, 2005, vol. 2.

[4] R. Dybowski, K. B. Laskey, J. W. Myers, and S. Parsons, "Introduction to the special issue on the fusion of domain knowledge with data for decision support," *Journal of Machine Learning Research*, vol. 4, no. 3, pp. 293–294, 2004.

[5] L. Cao, D. Luo, and C. Zhang, "Knowledge actionability: satisfying technical and business interestingness," *International Journal of Business Intelligence and Data Mining*, vol. 2, no. 4, pp. 496–514, 2007.

[6] L. Cao, "Domain driven data mining (d3m)," in *Data Mining Workshops, 2008. ICDMW'08. IEEE International Conference on*. IEEE, 2008, pp. 74–76.

[7] ——, "Domain-driven data mining: Challenges and prospects," *IEEE Transactions on Knowledge and Data Engineering*, vol. 22, no. 6, pp. 755–769, 2010.

[8] V. Mirchevska, M. Luštrek, and M. Gams, "Combining domain knowledge and machine learning for robust fall detection," *Expert Systems*, vol. 31, no. 2, pp. 163–175, 2014.

[9] J. Vieira and C. Antunes, "Decision tree learner in the presence of domain knowledge," in *Chinese Semantic Web and Web Science Conference*. Springer, 2014, pp. 42–55.

[10] P. M. Domingos, S. Kok, H. Poon, M. Richardson, and P. Singla, "Unifying logical and statistical ai." in *AAAI*, vol. 6, 2006, pp. 2–7.

[11] Z. Nazeri and E. Bloedorn, "Exploiting available domain knowledge to improve mining aviation safety and network security data," *The MITRE Corporation, McLean, Virginia*, vol. 22102, 2004.

[12] R. Agrawal, R. Srikant *et al.*, "Fast algorithms for mining association rules," in *Proc. 20th int. conf. very large data bases, VLDB*, vol. 1215, 1994, pp. 487–499.

[13] J. R. Quinlan, *C4. 5: programs for machine learning*. Elsevier, 2014.

[14] J. Barracosa and C. Antunes, "Anticipating teachers performance," in *KDD 2011 Workshop: Knowledge Discovery in Educational Data*, 2011, pp. 77–82.

[15] A. Bochare, A. Gangopadhyay, Y. Yesha, A. Joshi, Y. Yesha, M. Brady, M. A. Grasso, and N. Rishe, "Integrating domain knowledge in supervised machine learning to assess the risk of breast cancer," *International Journal of Medical Engineering and Informatics*, vol. 6, no. 2, pp. 87–99, 2014.

[16] P. Domingos and M. Pazzani, "On the optimality of the simple bayesian classifier under zero-one loss," *Machine learning*, vol. 29, no. 2-3, pp. 103–130, 1997.

[17] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[18] J. Pearl, "Bayesian networks," *Department of Statistics, UCLA*, 2011.

[19] P. W. Pardel, J. G. Bazan, J. Zarychta, and S. Bazan-Socha, "Automatic medical objects classification based on data sets and domain knowledge," in *International Conference: Beyond Databases, Architectures and Structures*. Springer, 2015, pp. 415–424.

[20] K. Kozaki, K. Hihara, and R. Mizoguchi, "Dynamic is-a hierarchy generation for user-centric semantic web," in *Workshop Ontologies come of Age in the Semantic Web(OCAS2011) 10 th International Semantic Web Conference Bonn, Germany, October 24, 2011*, 2011, p. 29.

[21] J. Zhang, D.-K. Kang, A. Silvescu, and V. Honavar, "Learning accurate and concise naïve bayes classifiers from attribute value taxonomies and data," *Knowledge and Information Systems*, vol. 9, no. 2, pp. 157–179, 2006.

[22] G. P. Malafsky and B. Newman, "Organizing knowledge with ontologies and taxonomies," *Fairfax: TechI LLC. Available at*, 2009.

[23] J. Zhang, A. Silvescu, and V. Honavar, "Ontology-driven induction of decision trees at multiple levels of abstraction," in *International Symposium on Abstraction, Reformulation, and Approximation*. Springer, 2002, pp. 316–323.

[24] T. Łukaszewski and S. Wilk, "Classification with test costs and background knowledge," *Knowledge-Based Systems*, vol. 92, pp. 35–42, 2016.

[25] B. Motik, P. F. Patel-Schneider, B. Parsia, C. Bock, A. Fokoue, P. Haase, R. Hoekstra, I. Horrocks, A. Ruttenberg, U. Sattler *et al.*, "Owl 2 web ontology language: Structural specification and functional-style syntax," *W3C recommendation*, vol. 27, no. 65, p. 159, 2009.

[26] A.-H. Tan, "Cascade artmap: Integrating neural computation and symbolic knowledge processing," *IEEE Transactions on Neural Networks*, vol. 8, no. 2, pp. 237–250, 1997.

[27] T.-H. Teng, A.-H. Tan, and J. M. Zurada, "Self-organizing neural networks integrating domain knowledge and reinforcement learning," *IEEE transactions on neural networks and learning systems*, vol. 26, no. 5, pp. 889–902, 2015.

[28] X. Shu, G.-J. Qi, J. Tang, and J. Wang, "Weakly-shared deep transfer networks for heterogeneous-domain knowledge propagation," in *Proceedings of the 23rd ACM international conference on Multimedia*. ACM, 2015, pp. 35–44.

[29] D. Zhou, D. Zhong, and Y. He, "Event trigger identification for biomedical events extraction using domain knowledge," *Bioinformatics*, vol. 30, no. 11, pp. 1587–1594, 2014.

[30] M. Richardson and P. Domingos, "Markov logic networks," *Machine learning*, vol. 62, no. 1, pp. 107–136, 2006.

[31] W. Duch, R. Adamczak, and K. Grabczewski, "A new methodology of extraction, optimization and application of crisp and fuzzy logical rules," *IEEE Transactions on Neural Networks*, vol. 12, no. 2, pp. 277–306, 2001.

[32] P. Domingos, S. Kok, D. Lowd, H. Poon, M. Richardson, and P. Singla, "Markov logic," in *Probabilistic inductive logic programming*. Springer, 2008, pp. 92–117.

[33] H. Mittal, S. S. Singh, V. Gogate, and P. Singla, "Fine grained weight learning in markov logic networks," 2015.

[34] P. Singla and P. Domingos, "Discriminative training of markov logic networks," in *AAAI*, vol. 5, 2005, pp. 868–873.

[35] P. Domingos and D. Lowd, "Markov logic: An interface layer for artificial intelligence," *Synthesis Lectures on Artificial Intelligence and Machine Learning*, vol. 3, no. 1, pp. 1–155, 2009.

[36] W. R. Gilks, S. Richardson, and D. Spiegelhalter, *Markov chain Monte Carlo in practice*. CRC press, 1995.

[37] G. O. Roberts, "Markov chain concepts related to sampling algorithms," *Markov chain Monte Carlo in practice*, vol. 57, 1996.

[38] D. Lowd and P. Domingos, "Efficient weight learning for markov logic networks," in *European Conference on Principles of Data Mining and Knowledge Discovery*. Springer, 2007, pp. 200–211.

[39] H. Poon and P. Domingos, "Sound and efficient inference with probabilistic and deterministic dependencies," in *AAAI*, vol. 6, 2006, pp. 458–463.

[40] W. Wei, J. Erenrich, and B. Selman, "Towards efficient sampling: Exploiting random walk strategies," in *AAAI*, vol. 4, 2004, pp. 670–676.

[41] R. M. Neal, "Slice sampling," *Annals of statistics*, pp. 705–741, 2003.

[42] P. Domingos, D. Jain, S. Kok, D. Lowd, L. Mihalkova, H. Poon, M. Richardson, P. Singla, M. Sumner, and J. Wang. (2012) Alchemy: Open source ai. [Online]. Available: http://alchemy.cs.washington.edu

[43] V. A. Silva and F. G. Cozman, "Markov logic networks for supervised, unsupervised and semisupervised learning of classifiers," in *IV Workshop on MSc Dissertation and PhD Thesis in Artificial Intelligence (WTDIA)*, 2008.

[44] F. Niu, C. Ré, A. Doan, and J. Shavlik, "Tuffy: Scaling up statistical inference in markov logic networks using an rdbms," *Proceedings of the VLDB Endowment*, vol. 4, no. 6, pp. 373–384, 2011.

[45] K. Beedkar, L. Del Corro, and R. Gemulla, "Fully parallel inference in markov logic networks." in *BTW*. Citeseer, 2013, pp. 205–224.

[46] J. T. Kent, "Information gain and a general measure of correlation," *Biometrika*, vol. 70, no. 1, pp. 163–173, 1983.

[47] E. Frank, M. Hall, G. Holmes, R. Kirkby, B. Pfahringer, I. H. Witten, and L. Trigg, "Weka-a machine learning workbench for data mining," in *Data mining and knowledge discovery handbook*. Springer, 2009, pp. 1269–1277.

[48] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[49] R. J. Brachman, H. J. Levesque, and R. Reiter, *Knowledge representation*. MIT press, 1992.

[50] T. G. Dietterich *et al.*, "Ensemble methods in machine learning," *Multiple classifier systems*, vol. 1857, pp. 1–15, 2000.