Universität des Saarlandes
Max-Planck-Institut für Informatik
AG5

Learning Rules With Categorical Attributes from Linked Data Sources

Masterarbeit im Fach Informatik
Master's Thesis in Computer Science
von / by

André de Oliveira Melo

angefertigt unter der Leitung von / supervised by

Dr. Martin Theobald

begutachtet von / reviewers

Dr. Martin Theobald

Prof. Dr. Max Mustermann

Januar / January 2013

Hilfsmittelerklärung

Hiermit versichere ich, die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben.

Non-plagiarism Statement

Hereby I confirm that this thesis is my own work and that I have documented all sources used.

Saarbrücken, den 29. Januar 2013,

(André de Oliveira Melo)

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlichtwird.

Declaration of Consent

Herewith I agree that my thesis will be made available through the library of the Computer Science Department, Saarland University.

Saarbrücken, den 29. Januar 2013,

(André de Oliveira Melo)

To my father Cicero, my mother Marlene and my sister Carolina

- Andre

Abstract

Inductive Logic Programming (ILP) is a consolidated technique for mining Datalog rules in knowledge bases. Nevertheless, it's inherently expensive and the hypothesis search space grows combinatorially with the knowledge base size. This problem is even more dramatic when literals with constants are considerated, therefore, it usually requires very aggressive pruning for making it feasible. With the objective of efficiently learning rules with ranges for numerical attributes, we propose a preprocessing step and an extension to ILP top-down algorithm, that suggests the most interesting categorical constants to be added in the refinement step. It consists of building a lattice that expresses the correlations between a root numerical attribute and multiple categorical relations and their constants. This lattice is then queried by the core ILP algorithm in the refinement step and provides a list of refinement suggestions ordered by a defined interestingness measure. This thesis discusses how to efficiently build the lattice, how it's incorporated in the core learning algorithm and evaluates different iterestingness measures.

Acknowledgements

Firstly, I would like to thank my advisor Dr. Martin Theobald, for his invaluable guidance. I feel deeply grateful for his technical assistance and motivational encouragement.

A special note of thanks to Prof. Dr. Max Mustermannfor giving me the opportunity to pursue this thesis at Infomation and Database Systems department under his supervision. It was a very enriching and pleasant experience to write my Master Thesis here.

Contents

Chapter 1

Introduction

In the last years, the volume of semantic data available, in particular in RDF format, has dramatically increased. Initiatives like the W3C Semantic Web and the Linked Open Data have great contribution in such development. The first provides a common standard that allows data to be shared and reused across different applications and the latter provides linkages between different datasets that were not originally interconnected. Moreover, advances in information extraction have also made a strong contribution, by crawling multiple non-structured resources in the Web and extracting RDF facts.

Nevertheless, information extraction still has its limitations and many of its sources might contain contradictory or uncertain information. Therefore, many of the extracted datasets suffer from incompleteness, noise and uncertainty. The firs one means that there are facts that are not existent in the dataset, the second one means that the dataset might contain facts that are not true and the latter one means that the truth of the facts is not certain. These problems make it much more challenging to automatically learn rules from the data.

In order to reduce such problems, one can apply a set of inference rules that describes its domain to the knowledge base. With that, it's possible to resolve contradictions as well as strengthen or weaken their confidence values. It's also possible to derive new facts that are originally not existent due to incompleteness. Such inference rules can be of two types:

1. *Hard Rules*: Consistency constraints which might represent functional dependencies, functional or inverse-functional properties of predicates or mutual exclusion. For example:
    - $marriedTo(x, y) : -marriedTo(y, x), (x \neq y)$

- $grandChildOf(x, y) : -childOf(x, z), childOf(z, y)$
- $parentOf(x, y) : -childOf(y, x)$
- $(z = y) : -wasBornIn(x, z), wasBornIn(x, y)$

2. *Soft Rules*: Weighted Datalog rules that frequently, but not always hold in the real world. As they might also produce incorrect information, derived facts can have a level of uncertainty which is represented by a confidence value. For example, it's know that married people usually live in the same place as their partner:

$$livesIn(x, y) : -marriedTo(x, z)livesIn(z, y)$$

So, if we have an incomplete knowledge base, which lacks information about where *Michelle Obama*l lives, but we know that she's married to *Barack Obama* and he lives in *Washington, D.C.*, we could then apply this soft rule to derive the fact *livesIn(MichelleObama, WashingtonDC)*.

Such rules are rarely known beforehand, but the data itself can be used to mine these rules using Inductive Logic Programming (ILP). It is a well-established framework for inductively learning relational descriptions (in the form of logic programs) from examples and background knowledge. Given a logical database of facts, an ILP system will generate hypothesis in a pre-determined order and test them against the examples. However in a large knowledge base, ILP becomes too expensive as the search space grows combinatorially with the knowledge base size and the larger the number of examples, the more expensive it is to test each hypothesis.

Although it is very expensive to learn rules with constants, such kind of rules can are extremely interesting as they can express particular characteristics of specific groups. For example, the rule $speaks(x, y)$:-$livesIn(x, z)$ is not very meaningful, it simply tells that people that live in anywhere will speak some language. Nevertheless, if we consider setting constant values for y and z, we can learn much more valuable rules such as $speaks(x, English)$:-$livesIn(x, Australia)$ or $speaks(x, Spanish)$:-$livesIn(x, Mexico)$.

Given the huge size of search space and the great interestingness of rules with constants, it is appropriate to reduce the search space by pruning constants or combinations of constants that do not add interesting information to the hypothesis.

[Talk more about ILP]

1.1 Motivation

Numerical properties are a special case, depending on the numerical attribute domain, in case of a continuous real number domain for example, setting a numerical constant as an individual value will very likely have a single entity associated to it. In such case it would be equivalent to simply specifying a constant for this given entity without the necessity of adding the numerical property

For example, no country has the exact same GDP or population as any other country. If we have the following rule:

$$speaks(x, Portuguese) \text{ :- } livesIn(x, y), hasPopulation(y, 193946886)$$

Assuming Brazil is the only country with a population of 193,946,886 inhabitants, this same rule would be equivalent to:

$$speaks(x, Portuguese) \text{ :- } livesIn(x, Brazil)$$

Of course, if we have to choose between one of the two rules, the latter one would be preferred as it is shorter and specifies the country in a clearer manner. Moreover, by setting numerical constants punctually, we would end up having a huge number of different constants to test in the hypothesis and most of them would be most likely discarded because of low support.

Therefore, it is interesting to split the attribute's domain into categories by discretizing it . Subsequently check if any of the buckets present different accuracy in comparison to its correspondent numerical constant-free rule (which we will call base-rule).

For example if we test the hypothesis and we find support=100 and confidence=0.4:

$$isMarriedTo(x, y) \text{ :- } hasAge(x, z)$$

and then we split z into three buckets:

- $k = 1 : z \in [0, 20]$
- $k = 2 : z \in (20, 40]$
- $k = 3 : z \in (40, \infty]$

we then generate three refined-rules by restricting the base-rule's domain from variable z:

- $isMarriedTo(x, y) : -hasAge(x, z), z \in [0, 20]$
  support=40, confidence=0.1
- $isMarriedTo(x, y) : -hasAge(x, z), z \in (20, 40]$
  support=40, confidence=0.5
- $isMarriedTo(x, y) : -hasAge(x, z), z \in (40, \infty]$
  support=20, confidence=0.8

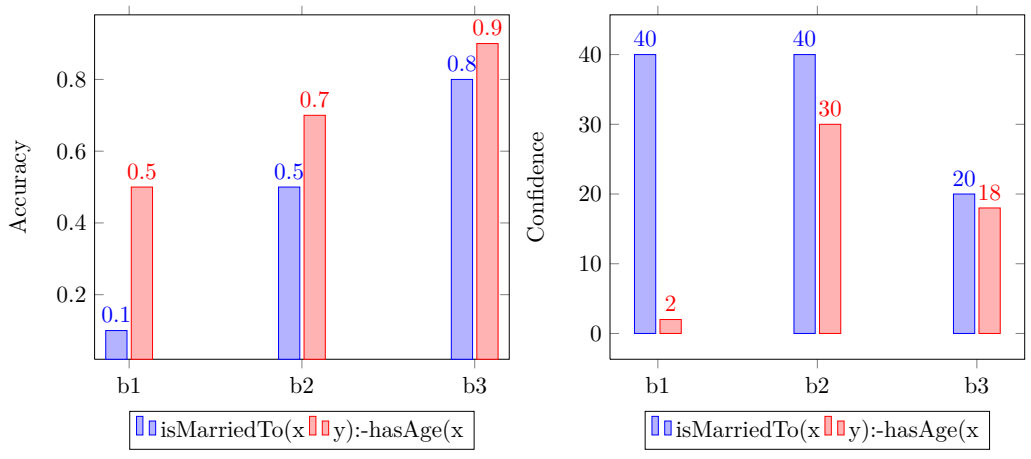
for $k = 2$ and $k = 3$, the hypothesis has significant gain by specifying numerical constants. Adding a relation to the body might produce totally different support and confidence distributions along the buckets. For example, if we add the relation hasChild(x,a), we could obtain other interesting rules:

Base-rule:

- $isMarriedTo(x, y) \leftarrow hasAge(x, z)hasChild(x, a)$
  support=50, confidence=0.625

Refined-rules:

- $isMarriedTo(x, y) \leftarrow hasAge(x, z)hasChild(x, a), z \in [0, 20]$
  support=2, confidence=0.5
- $isMarriedTo(x, y) \leftarrow hasAge(x, z)hasChild(x, a), z \in (20, 40]$
  support=30, confidence=0.7
- $isMarriedTo(x, y) \leftarrow hasAge(x, z)hasChild(x, a), z \in (40, \infty]$
  support=18, confidence=0.9



Adding more predicate to the rule might not bring any gain or even loss in accuracy to the base-rule, but when bucketing per age, present a different accuracy and support distribution and might even produce gain in accuracy for some specific buckets.

Nevertheless, adding a predicate with no correlation to the rule might not generate any gain. Thus, it's necessary to carefully choose the relations and eventual constants and discard the ones with no correlation to the rule.

1.2 Contributions

In this thesis, we propose a pre-processing step to build a graph we call Correlation Latticefor each numerical property we want to for interesting intervals. In each graph, that has a numerical property as root, we first query the examples distribution on the numerical attribute, and build a histogram by splitting them into k buckets. Subsequently, we pick a set of c categorical properties that can be joined with the root, extract the frequencies histogram and analyze how the distribution of sub-population created by joining them with the root is affected. Afterwards, we try to combine each of the categories and see if they still produce interesting sub-populations creating a lattice, like in frequent set mining apriori algorithm.

We discuss the pruning opportunities and also evaluate different heuristics and interestingness measures and their efficiency in finding rules with numerical intervals.

With information about different examples distributions contained in the generated lattice, once we add one of the root properties during the core ILP algorithm, we can then search for the most interesting categorical properties that could result in different accuracy distributions. For every categorical property we can also suggest the most interesting constants and other categorical properties to be combined in a subcategory of both.

1.3 Outline

Chapter 2

Related Work

In this chapter relevant work related to the thesis topic will be briefly presented. In section 2.1...

2.1 Logic Programming

[1] [2] In this section, basic logic programming and deductive database terminology such as *literal*, *clause*, *program clause*, *datalog clause* and *hypothesis* will be presented. Firstly, it is important to mention that variables are represented as uppercase letters followed by a string of lowercase letters and/or digits. Function and predicate symbols are lowercase letters also followed by a string of lowercase and/or digits.

An *atomic formula* L is a predicate symbol followed by a bracketed n-tuple of *terms*. A *term* can be a variable or a function symbol followed by a bracketed n-tuple of terms. A constant is a function symbol of arity 0. So for example, if f, g, and h are function symbols and X a variable, then $g(X)$ is term, $f(g(X), h)$ is also a term and h is a *constant*.

A *literal* is an *atomic formula* which can be negated or not. So both L and its negation $\overline{L}$ are literals for any *atomic formula* L. A clause c is a disjunction of literals, for example:

$$c = (L_1 \vee L_2 \vee \ldots \overline{L_i} \vee \overline{L_{i+1}} \vee \ldots) \equiv L_1 \vee L_2 \vee \ldots \leftarrow L_i \wedge L_{i+1} \wedge \ldots$$

Such disjunction of literals can also be written in following way:

$$\{L_1, L_2, \ldots, \overline{L_i}, \overline{L_{i+1}}, \ldots\}$$
$$L_1, L_2, \ldots \leftarrow L_i, L_{i+1}, \ldots$$

A *program clause* is a clause which contains exactly one positive literal. That is, it has the form:

$$\underbrace{T}_{head} \leftarrow \underbrace{L_1, L_2, \ldots}_{body}$$

A *datalog clause* is a program clause with no function symbols with arity greater different from zero. That means that only variables and constants can be used as predicate arguments. A datalog clause is considered $safe$ if all the variables present in the head literal T are also present in the body. Moreover, it may also allow negated literals in the body, as long as every variable existent in a negated body literal are also be present in a non-negated body literal.

2.2 Inductive Logic Programming

[2] Inductive Logic Programming (ILP) is a machine learning technique that combines inductive machine learning with the logic programming representation. Given a known background knowledge and a set of training examples represented as a logical database of facts (literals without any variables), an ILP system will learn a hypothesis in the form of a logic program.

The training data is a set of examples ε, where each examples is a grounded fact labeled as positive ($\oplus$) or negative ($\ominus$). We denote the set of positive examples as ε^+ and the set of negative examples as ε^-

The background knowledge $\mathcal{B}$ is a prior knowledge which contributes in learning the hypothesis. It indirectly restricts the hypothesis search space, as the learned hypothesis $\mathcal{H}$ should be consistent with the background knowledge as well as the training examples.

A hypothesis $\mathcal{H}$ is said to cover an example e given a background knowledge $\mathcal{B}$ ($covers(\mathcal{B} \cup \mathcal{H}, e)$) if the example satisfies the hypothesis and background knowledge. In logic programming where the a $\mathcal{H}$ is a set of program clauses and an example is a ground fact, it means that e is entailed by $\mathcal{B} \cup \mathcal{H}$.

$$covers(\mathcal{B} \cup \mathcal{H}, e) = true \text{ if } \mathcal{B} \cup \mathcal{H} \models e$$

We can also define a covering function for a set of examples which returns a a subset with the examples entailed by $\mathcal{B} \cup \mathcal{H}$:

$$covers(\mathcal{B} \cup \mathcal{H}, \varepsilon) = \{e \in \varepsilon | \mathcal{B} \cup \mathcal{H} \models e\}$$

Table 2.1: A simple ILP problem: learning the *daughter* relation.

Training Examples	Background Knowledge
daughter(mary,ann) $\oplus$	parent(ann,mary).
daughter(eve,tom) $\oplus$	parent(ann,tom).
daughter(tom,ann) $\ominus$	parent(tom,eve).
daughter(eve,ann) $\ominus$	parent(tom,ian).
	female(ann).
	female(mary).
	female(eve).

Therewith, we can define two important concepts in inductive learning:

- Completeness: A hypothesis $\mathcal{H}$ is complete with respect to background knowledge $\mathcal{B}$ and examples ε if all the positive examples are covered, or in other words: $covers(\mathcal{B}, \mathcal{H}, \varepsilon^+) = \varepsilon^+$
- Consistency: A hypothesis $\mathcal{H}$ is consistent with respect to background knowledge $\mathcal{B}$ and examples ε if no negative examples are covered, or in other words: $covers(\mathcal{B}, \mathcal{H}, \varepsilon^-) = \emptyset$

The objective of ILP is to find a hypothesis that is complete and consistent with respect to the given training examples and background knowledge. The example shown in table ?? (presented in [2]) illustrates a simple problem of learning a hypothesis for the target relation $daughter(X, Y)$.

If we consider, for example, the language of safe datalog clauses, it is possible to formulate the following complete and consistent hypothesis:

$$hypothesis = daughter(X, Y) \text{ :- } female(X), parent(Y, X)$$

2.2.1 Searching the Hypothesis Space

In ILP, the hypothesis space is determined by the language of the programs $\mathcal{L}$ consisting of the possible program clauses allowed by the language. Also, the vocabulary of predicate symbols is determined by the predicates from the background knowledge $\mathcal{B}$.

It is useful to structure the search space with partial ordering into a set of clauses based on θ-subsumption in order to systematically search the program clauses space. A clause

c θ-subsumes c' if there's a substitution θ such that clause $c\theta \subseteq c'$. This also introduces a notion of generality, where clause c is at least as generaly as c', or in other words, c' is a specialization of c.

For example, for $c = daughter(X,Y)$:-$parent(Y,X)$ and $c' = daughter(X,Y)$:-$female(X), parent(Y,X)$. We know that c' is a specialization of c because for $\theta = \emptyset$, $c \subset c'$ since $\{daughter(X,Y), \neg parent(Y,X)\} \subset \{daughter(X,Y), \neg female(X), \neg parent(Y,X)\}$.

If we have $c = livesIn(X,Y)$:-$marriedTo(X,Z), livesIn(Z,Y)$ and $c' = livesIn(X, germany)$:-$marriedTo(X,Z), livesIn(Z, germany)$, we know that c' is a specialization of c because for the substitution $\theta = \{Y/germany\}$, $c\theta = c'$.

This notion gives us an important property that cab be used for pruning parts of the search space:

- When generaling from c to c', all the examples covered by c are also covered by c'. So if c is inconsistent, then all its generalizations are inconsistent as well.
- When specializing from c to c', an example not covered by c will neither be covered by c'. So if c' does not cover any positive examples, neither do all its specializations.

These properties are the basis for two main search approaches:

- Bottom-up: starts with less general clauses and searches least general generalizations
- Top-down: starts with more general clauses and searches for most general consistent specializations

As in this thesis we work only with the top-down approach, we will discuss it in more details in the next subsection. Further information about bottow-up approach can be found in [2].

2.2.2 Top-Down ILP

Top-down ILP algorithm relies on specializations. Given the θ-subsumtion definition, we have two main specialization techniques:

- Apply a substitution on the clause
- Add a literal to the body of the clause

The top-down algorithm consists basically of a specitialization loop that refines a clause ensuring consistency embedded inside a covering loop that adds clauses to the hypothesis ensuring completeness. In domains with perfect data, the stopping criterea require that all positive examples are covered (completeness) and no negative examples are covered (consistency). Nevertheless, in domains with imperfect data (due to noise, incompleteness, etc.), heuristic stopping criterea can be used to tolerate some level of incompleteness and inconsistency.

This simplest heuristic is the expected accuracy of a clause, which is defined as the probability that an example covered by the clause is labeled as positive:

$$A(c) = P(ein\varepsilon^{+}|c) = \frac{n^{+}(c)}{n^{+}(c) + n^{-}(c)} \tag{2.1}$$

where $n^{+}(c)$ is the number of postive and $n^{-}(c)$ the number of negative examples covered by c.

[Sampling,MaxLiterals]

2.3 First Order Inductive Learning

[**?**] [**?**]

2.4 Association Rule Mining

[**?**] [**?**]

Association Rule Mining is method for discovering interesting relations between variables in large databases. It is intended to work on a database composed by a set of transactions $D = \{t_1, t_2, \ldots, t_m\}$ and a set of items $I = \{i_1, i_2, \ldots, i_n\}$. Each transaction $\mathcal{T}$ consists of a set of items which is subset of I , which can be represented by a binary vector of size n indicating the presence of absence of items in the transaction.

The objective of association rule mining is to learn inference rules of the form $X \Rightarrow Y$, where $X, Y \subseteq I$ and $X \cap Y = \emptyset$. Rules are selected according to various interestingness measures that will be subsequently discussed.

We say a transaction $\mathcal{T}$ supports an itemset $\mathcal{X} \subseteq \mathcal{I}$ if $\mathcal{X} \subseteq \mathcal{T}$. The support measure $supp(X)$ is the defined as the ratio between the number of transactions supporting X and the total size of the database $\mathcal{D}$:

$$supp(X) = \frac{|\{\mathcal{T} \in \mathcal{D} | X \subseteq \mathcal{T}\}|}{|\mathcal{D}|} \tag{2.2}$$

The support of a rule $X \Rightarrow Y$ is defined as:

$$supp(X \Rightarrow Y) = supp(X \cup Y) \tag{2.3}$$

Them confidence of a rule $X \Rightarrow Y$ is defined as the conditional probability $P(Y|X)$:

$$conf(X \Rightarrow Y) = \frac{supp(X \cup Y)}{sup(X)} \tag{2.4}$$

Confidence and support are the two measures used in classical association rule mining. Albeit, there are other relevant measures such as *lift*. It measures how different the observed rule support is in comparison to that expected if X and Y were independent:

$$lift(X \Rightarrow Y) = \frac{supp(X \Rightarrow Y)}{supp(X) * supp(Y)} \tag{2.5}$$

A lift value 1 would imply that X and Y are independent, therefore any rule involving both itemsets doesn not make sense. A lift value greater than 1, provides information about the level of dependence between both variables. Higher lift values makes a rule potentially interesting.

2.4.1 Measures

2.4.1.1 Support

2.4.1.2 Confidence

2.4.1.3 Lift

2.4.2 Itemset Lattice

2.4.3 Apriori Algorithm

2.5 Mining Optimized Rules for Numeric Attributes

[3]

2.6 Information Theoretic Measures

[4]

2.7 Semantic Web Applications

2.8 Linked Open Data Applications

Chapter 3

Learning Rules With Numerical and Categorical Attributes

In this chapter we will discuss what kind of rules we are interested in, define what categorical properties are, describe in more details a Correlation Lattice, how to build it and integrate it in the core ILP learning algorithm.

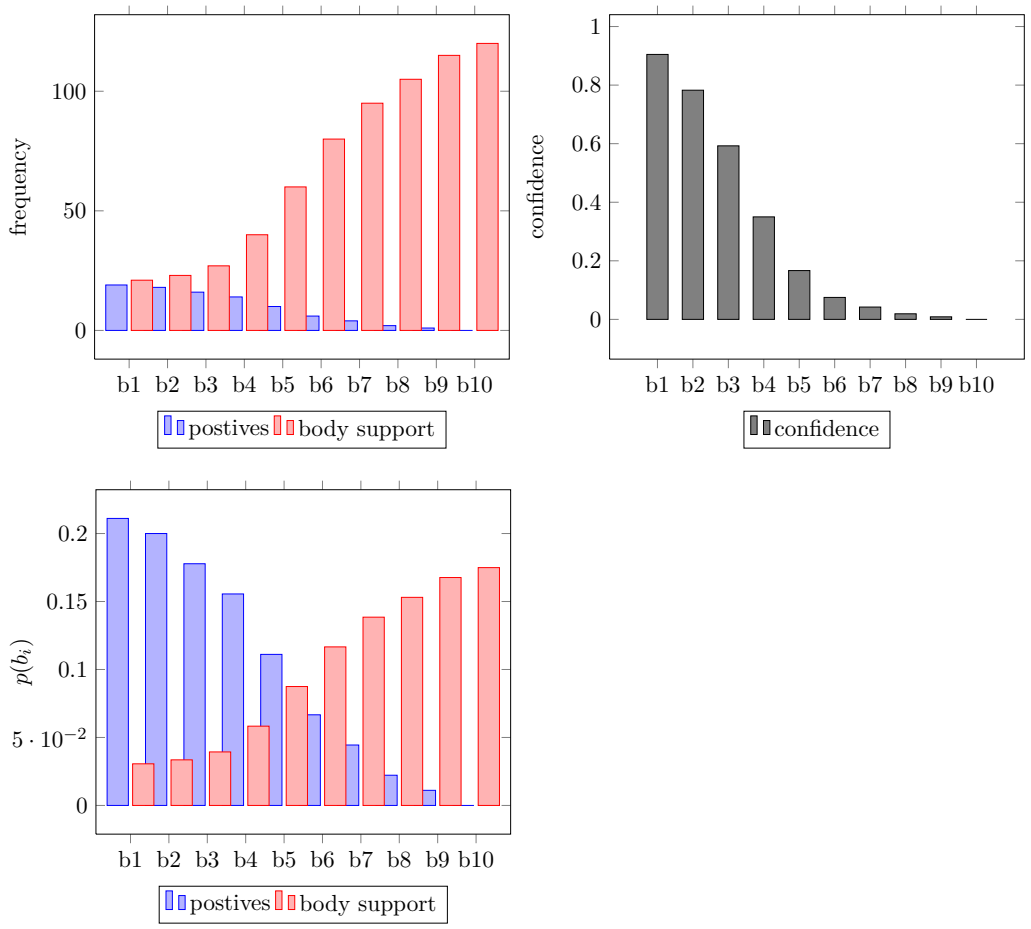
3.1 Interesting Rule Definition

In this section we formally define what kind of rules we are interested in obtaining with the algorithm proposed in this thesis. As briefly explained in the introduction, the objective is to learn rules with numerical properties, focusing on searching ranges in the numerical attribute's domain that satisfy support and confidence thresholds.

Searching numerical intervals for a base-rule that already satisfies confidence threshold is not so interesting. The base-rule itself implicitly specifies an interval which covers the whole numerical attribute domain from any possible refined-rules. Moreover, even if for some specific range we have a higher confidence value, the gain in comparison to the base-rule would not be so great as the base-rule already present a high confidence value (at least higher than the specified threshold).

More precisely, if we have a confidence threshold TS_{conf} and defining confidence gain from a refined-rule r_2 in comparison to a base-rule r_1 as g_{r_2,r_1} as:

$$g_{r_2,r1} = \frac{conf_{r_2}}{conf_{r_1}} \tag{3.1}$$



As we know that $r_1 \geq TS_{conf}$, then g_{r_2,r_1} is upper-bounded by:

$$g_{r_2,r1} \leq \frac{1}{TS_{conf}} \tag{3.2}$$

More interesting for us would be base rules that satisfy the support threshold but does not satisfy the confidence threshold. In such case, if we have a non-uniform confidence distribution along the buckets, it is possible for some intervals, its correspondent refined-rule satisfies both thresholds. Furthermore, the confidence gain is potentially much greater than otherwise.

For example in figure ??, we have an ideal example where positive examples and body support have completely different distributions and thus produce a very interesting confidence distribution.

In figure ??, we normalize the frequency histograms from the rule's positive examples and body support in order to better show how their distributions diverge.

3.2 Categorical Property Definition

In this section, we formally define a categorical property as used in the Correlation Lattice.

First of all, a candidate relation must be joined with root relation's non-numerical variable

A candidate categorical relation $r(x, y)$, should be equivalent to a non-injective function:

$r(x, y) \equiv f : X \to Y, \quad s.t. \quad |Y| < |X|$ and $|Y| > 1$
Therefore, $\nexists\, g : Y \to X, \quad s.t. \quad f(g(x)) = x, \quad \forall x \in X$

We can define subsets $X_i \in X$, with which X_i belonging to one category $y_i \in Y$:

$X_i \subset X \quad s.t. \quad X_i = \{x \in X \mid f(x) = y_i,\, y_i \in Y\}$
$X = \bigcup_{i=1}^{n} X_i$ and $X_i \cap X_j = \emptyset, \quad \forall i, j \in [1, n],\, i \neq j$

We can also broaden this definition by composing functional relations with a categorical or multiple categorical relations:

If we have:

$r_1(x, y) \equiv f_1 : X \to Y$
$r_2(y, z) \equiv f_2 : Y \to Z$

Where at least one of them is categorical, then $r'(x, z) \equiv f : X \to Z$, where $r'(x, z) = r_2(f_1(x), z)$ is also categorical

Numerical properties can also be turned into a categorical, by simply applying a bucketing function that maps a numerical domain into a finite set of k buckets:

$b : \mathbb{N} \to B$, where $B = \{1, 2, \ldots, k\}$

e.g.: [make an example]

So a numerical property:

$r(x, y) \equiv f : X \to \mathbb{N}$

combined with a bucketing function b, $r'(x, b(y))$ would be discretized and applicable in the Correlation Lattice.

Types are also covered by this definition. If we consider $r =$ *rdf:type* we see that r which maps entities into types satisfies the definition presented before.

Similarly different knowledge bases can be interconnected by simply applying this definition with the *owl:sameAs* relation. This allows to broaden the set of categorical

properties to be analyzed in the Correlation Latticeto those contained in interlinked datasets.

3.2.1 Absence or Presence of a Property as Categories

Another possibility is to define categories for presence or absence of supporting examples for properties. With this approach, one can also include non-categorical properties into the Correlation Latticeand have insight about how the presence or absence of such property affects the distribution of root's numerical attribute.

In this case we must consider a property which does not have examples for all its domain. For example, the property *isParentOf(x,y)* has as both domain and range type Person. If not all the Person instances of the knowledge base have supporting facts for the relation *isParentOf*, then we can split the instances in two categories: the ones that are covered by the property and the ones that are not. For example, let *hasIncome* be the root property and let's assume we have a knowledge base with the following facts:

hasIncome(John,20000)	isParentOf(John,Mary)
hasIncome(Mike,30000)	isParentOf(Mike,Paul)
hasIncome(Anne,25000)	isParentOf(Anne,Paul)
hasIncome(Mary,5000)	isParentOf(Anne,Mary)
hasIncome(Lisa,10000)	
hasIncome(Paul,0)	

Then we could split the root node set $X_{root} = \{x | \exists hasIncome(x, z)\}$ in two categories $X^1_{root}, X^2_{root} \in X_{root}$:

$X^1_{root} = \{x | x \in X_{root} \wedge \exists isParentOf(x, y)\}$
$X^2_{root} = \{x | x \in X_{root} \wedge \nexists isParentOf(x, y)\}$

Each group would consist of the following instances:

$X_{root} = \{John, Mike, Anne, Mary, Lisa, Paul\}$
$X^1_{root} = \{John, Mike, Anne\}$
$X^2_{root} = \{Mary, Lisa, Paul\}$

As in this thesis we are under open world assumption and we don not consider negated literals in the hypothesis, we ignore the absence and just include the presence of relations in the lattice.

3.2.2 Notation Used

As we will see in the next sections, in the Correlation Latticeall the relations are joined by the variable of root's first argument. So we will denote the presence of a property $r(x, y)$ category as simply r, where x is the join variable and y is free and not set to any constant. If we have a categorical property $a(x, y)$ with k different categories $A_i \in Y$ $(i = 1, .., k)$, we denote each of the $a(x, A_i)$ as a_i.

So if we have a root node r and a node at level 3 called rab_1c_3, that means we are considering the examples for the following join:

$r(x, y)a(x, z)b(x, B_1)c(x, B_3)$

3.3 Preprocessing

In this section, we will present the preprocessing steps required by our proposed algorithm. It basically consists of first building a joinable relations map for each join pattern, according to relations domain and range types as well as support threshold. Afterwards, we search the available categorical properties for each numerical relation that will be used in the Correlation Lattice. At last we build the so called Correlation Lattice, which belongs to the preprocessing step but will be discussed in the next section.

3.3.1 Relation Preprocessing

In this step, we focus on creating a map of joinable for each join pattern between two relations. As in this thesis we are working with RDF triples, we have four possible join patterns:

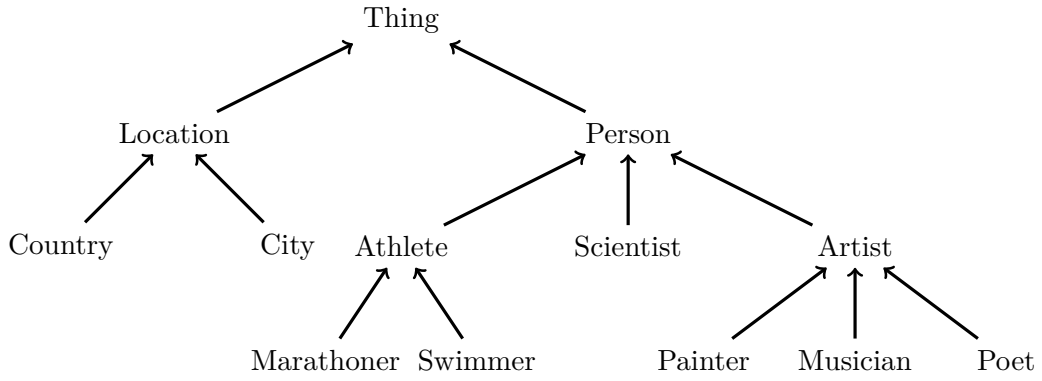
- Argument 1 on Argument 1: e.g. *hasIncome(**x**,y)hasAge(**x**,z)*
- Argument 1 on Argument 2: e.g. *hasIncome(**x**,y)isMarriedTo(z,**x**)*
- Argument 2 on Argument 1: e.g. *livesIn(y,**x**)isLocatedIn(**x**,z)*
- Argument 2 on Argument 2: e.g. *livesIn(y,**x**)wasBornIn(z,**x**)*

With that we could easily obtain all the possible pair of joinable relations for each join pattern, avoiding testing hypothesis containing invalid join pairs.

3.3.1.1 Exploiting Relation Range and Domain Types

A knowledge base is expected to have an ontology defining the structure of the stored data (the types of entities and their relationships). Additionally, every relation's range (type of 1st argument) and domain (type of 2nd argument) should be defined. These information can help us identify the allowed joining relations for each join pattern.

Figure 3.1: Type hierarchy example



Assuming that the knowledge base has its type hierarchy described with the relation *rdfs:subClassOf* and both argument types from each of the relations declared with *rdfs:domain* and *rdfs:range* it is a straightforward task.

For every possible pair of relations, we simply try to match the joining argument types from the 2 joining relations. We check whether they are equal or if one can be subsumed by the other. If so, then it is allowed to join the pair of relations on the given joining arguments, or in other words, the type hierarchy does not prevent them from joining. The pseudo-code for performing such task is shown in the Algorithm 1:

Nevertheless, it might be that in the knowledge base, the cardinality of such join might be zero or simply not exceed the support threshold. Thus, it is worth to that beforehand, and that is what will be explained in the next section.

3.3.1.2 Exploiting Support Monotonicity

Support is a monotonically decreasing measure in top-down ILP. So we know that by adding any literals to the hypothesis, we can only get a smaller or equal support. Therefore, for each pair of joinable relations in each of the join patterns, we can query the knowledge base and check whether they have enough supporting facts.

Thus, if any pair of relations does not reach the minimum support for a given join pattern, we know that any clause containing such join will therefore fail the support test as well, so we don not need to test such hypothesis in the core ILP algorithm.

Algorithm 1: Function *checkTypes*
Checks whether two relations are joinable for a given join pattern

Input: r_i, r_j: Joining relations, arg_i, arg_j: Joining arguments
Output: True if arg_i from r_i joins with arg_j from r_j, False otherwise

```
switch arg_i do
    case 1
        type_i ← r_i.domain ;
    case 2
        type_i ← r_i.range ;
switch arg_j do
    case 1
        type_j ← r_j.domain ;
    case 2
        type_j ← r_j.range ;
if type_i = type_j or subsumes(type_i,type_j) or subsumes(type_j,type_i) then
    return true;
else
    return false;
```

Algorithm 2: Function *checkSupport*
Checks whether join support exceeds threshold

Input: r_i, r_j: Joining relations, arg_i, arg_j: Joining arguments, *supportThreshold*: Support threshold
Output: True if join support exceeds threshold, False otherwise

```
switch (arg_i, arg_j) do
    case (1, 1)
        query ← "select count distinct ?x where { ?x <r_i> ?y . ?x <r_j> ?z }" ;
    case (1, 2)
        query ← "select count distinct ?x where { ?x <r_i> ?y . ?z <r_j> ?x }" ;
    case (2, 1)
        query ← "select count distinct ?x where { ?y <r_i> ?x . ?x <r_j> ?z }" ;
    case (2, 2)
        query ← "select count distinct ?x where { ?y <r_i> ?x . ?z <r_j> ?x }" ;
joinSupport ← executeQuery(query);
if joinSupport ≥ supportThreshold then
    return true;
else
    return false;
```

The preprocessing is done by algorithm 3, which applies 1 and 2 on all the possible join pair combinations and extracting the valid ones, we can build 4 joining maps, one for each join pattern. Each map has relations as keys and a set of joinable relations as value. In the refinement step at the ILP algorithm, these maps will be queried in order to obtain

suggestions of literals to be added.

Algorithm 3: Preprocessing algorithm

Input: r: Set of Relations, *supportThreshold*: The support threshold
Result: $L_{m,n}(r_i)$: List of joinable relations, where $m, n \in \{1, 2\}, r_i \in r$

```
// Lists initialization
```
foreach $r_i \in r$ **do**
  $L_{1,1}(r_i) \leftarrow r_i$;
  $L_{2,2}(r_i) \leftarrow r_i$;
  $L_{1,2}(r_i) \leftarrow \emptyset$;
  $L_{2,1}(r_i) \leftarrow \emptyset$;

```
// For every possible pair of relations
```
foreach $r_i \in r$ **do**
  foreach $r_j \in r$ **and** $j \geq i$ **do**
    if $r_i \neq r_j$ **then**
      if $checkTypes(r_i, r_j, 1, 1)$ **and** $checkSupport(r_i, r_j, 1, 1)$ **then**
        $L_{1,1}(r_i) \leftarrow L_{1,1}(r_i) \cup r_j$;
        $L_{1,1}(r_j) \leftarrow L_{1,1}(r_j) \cup r_i$;
      if $checkTypes(r_i, r_j, 2, 2)$ **and** $checkSupport(r_i, r_j, 2, 2)$ **then**
        $L_{2,2}(r_i) \leftarrow L_{2,2}(r_i) \cup r_j$;
        $L_{2,2}(r_j) \leftarrow L_{2,2}(r_j) \cup r_i$;
    if $checkTypes(r_i, r_j, 1, 2)$ **and** $checkSupport(r_i, r_j, 1, 2)$ **then**
      $L_{1,2}(r_i) \leftarrow L_{1,2}(r_i) \cup r_j$;
      $L_{2,1}(r_j) \leftarrow L_{2,1}(r_j) \cup r_i$;
    if $checkTypes(r_i, r_j, 2, 1)$ **and** $checkSupport(r_i, r_j, 2, 1)$ **then**
      $L_{2,1}(r_i) \leftarrow L_{2,1}(r_i) \cup r_j$;
      $L_{1,2}(r_j) \leftarrow L_{1,2}(r_j) \cup r_i$;

3.3.2

3.4 Correlation Lattice

The idea is to build during preprocessing a graph inspired in the itemset lattice that describes the influence of different categorical relations on a given numerical attribute's distribution. We call such graph a Correlation Lattice. Comparing to a Itemset Lattice, in a Correlation Latticewe have a set of categorical relations (that are joined with root property) instead of items. In addition, Each node in the graph has an associated histogram with the support distribution over the root's numerical attribute.

To illustrate the idea, let's analyze a simple real-world example with the *hasIncome* relation. If we have two categorical relations, one strongly correlated to income, e.g. *hasEducation*, and one uncorrelated (or very weakly correlated), e.g. *wasBornInMonth*.

Let's assume that for the relation *wasBornInMonth(x,y)* we have the 12 months from the Gregorian Calendar as constants and for *hasEducation(x,y)* we can have 10 different categorical constants for y: "Preschool", "Kindergarten", "ElementarySchool", "MiddleSchool", "Highschool", "Professional School", "Associate's degree", "Bachelor's degree", "Master's degree" and "Doctorate degree".

It's expected that the income distribution will be roughly the same for people born in any of the months, whereas for different education levels, e.g. Elementary School and Doctoral Degree, their income distribution are expected to be different from each other them and different from the overall income distribution.

Based on this idea, we basically check how different categorical relations affect a numerical distribution. Such information, together with other measures such as support, provides valuable cues on what categorical attributes and what categorical constants might be the most interesting to be added to the hypothesis in the core ILP algorithm.

3.4.1 Building the Lattice

We first start with the chosen numerical property. We get all the facts existent for this property, define the buckets and count the examples producing a frequency histogram.

Subsequently, for the chosen set of categorical relations, we do the same with the examples obtained by joining the root with each of the categories. For every category we create a node in the lattice with assiciated frequency histogram. We also each the root as parent node, and add the new nodes as children from the root.

In a further step, we try to join every possible pair of categorical relations and including the constants. For the given, example with the relations *hasEducation* and *wasBornInMonth* we would then create the nodes:

hasIncome(x,y)wasBornInMonth(x,"January"),hasEducation(x,"Preschool")
hasIncome(x,y)wasBornInMonth(x,"January"),hasEducation(x,"Kindergarten")
...
hasIncome(x,y)wasBornInMonth(x,"January"),hasEducation(x,"Doctorate Degree")

hasIncome(x,y)wasBornInMonth(x,"February"),hasEducation(x,"Preschool")
hasIncome(x,y)wasBornInMonth(x,"February"),hasEducation(x,"Kindergarten")
...
hasIncome(x,y)wasBornInMonth(x,"February"),hasEducation(x,"Doctorate Degree")

...

hasIncome(x,y)wasBornInMonth(x,"December"),hasEducation(x,"Preschool")
hasIncome(x,y)wasBornInMonth(x,"December"),hasEducation(x,"Kindergarten")
...
hasIncome(x,y)wasBornInMonth(x,"December"),hasEducation(x,"Doctorate Degree")

This process works just like in an itemset lattice until a predetermined maximum level or until all the possible combinations are exhausted. Figure 3.2 shows how a Correlation Latticelooks like.

3.4.2 Pruning Opportunities

In this section we discuss about safe pruning opportunities that can be explored whilst building the Correlation Lattice: support and conditional independence. As these might not be sufficient to reduce lattice size to a feasible level, later, in section(??), we will also discuss possible pruning techniques.

3.4.2.1 Support

As described in ([5]), in top-down ILP every refinement causes the support to decrease, therefore we know that for every node in the Correlation Lattice, its support will be greater or equal than any of its children, so support is a monotonically decreasing measure so we can safely prune a node that does not reach the minimum support threshold.

In this thesis, we define support as the absolute frequency of supporting facts.

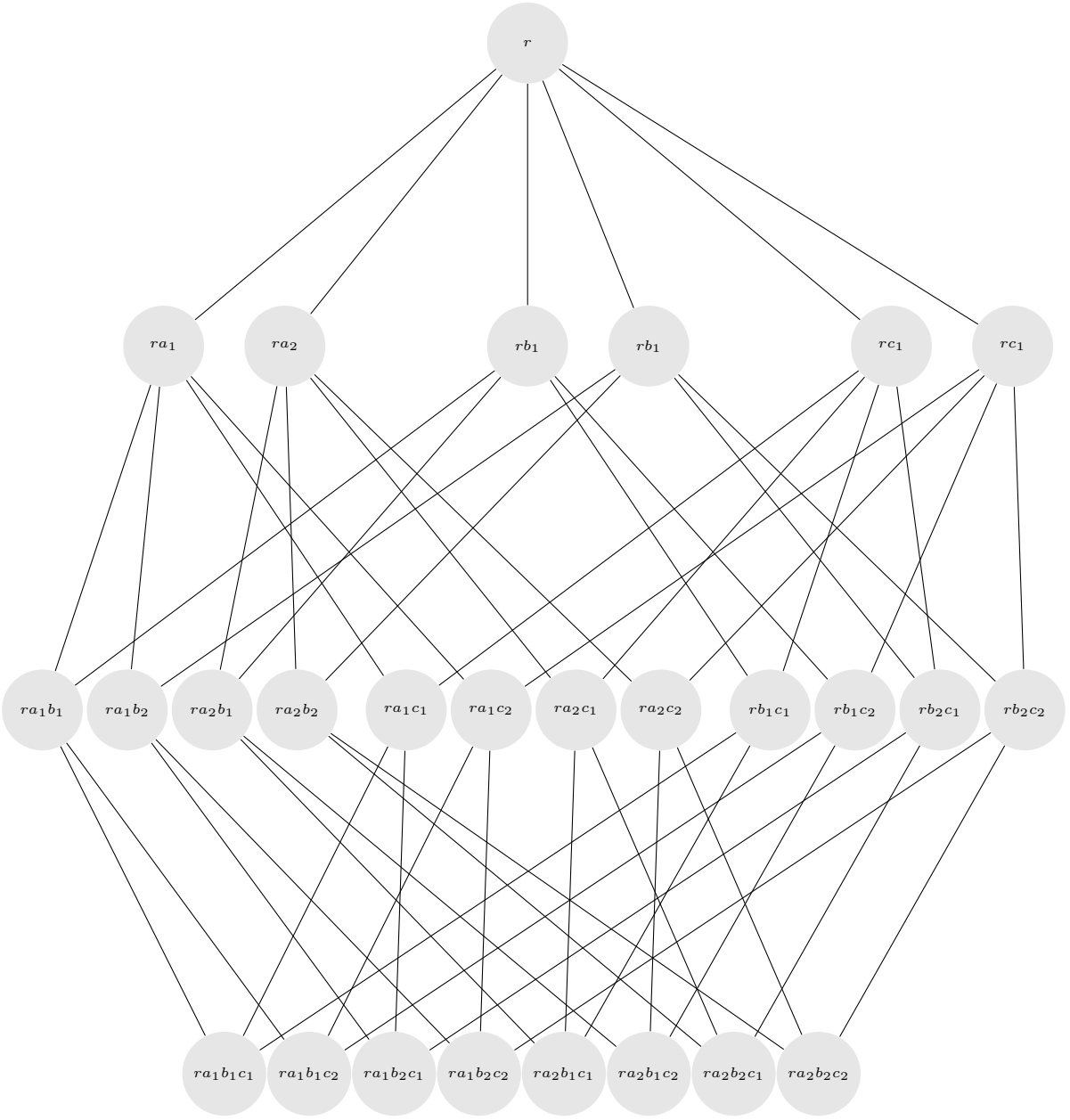
3.4.2.2 Independence Checks

By simplicity, we assume that every possible pair of categorical relations are independent given their common parent and we search for evidence to prove the contrary.

For 2 nodes to be joined, they must have a common parent, i.e. two nodes at level l (with $l+1$ literals) are joinable if they share l literals. Therefore, it is straightforward to calculate the conditional probabilities of each of the joining nodes given the common parent, and estimate the frequency distribution for the conditional independence case.

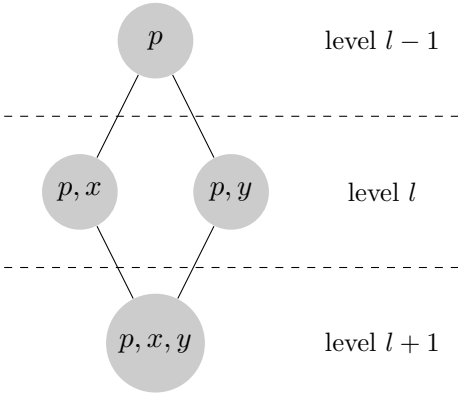
Let's say we have the following join case:

Figure 3.2: Correlation Lattice example



Each node p contains a histogram $h(p)$ with the example frequencies and we define $h_i(p)$ as the frequency of bucket b_i in $h(p)$. For the example shown in figure 3.3, we can calculate the conditional probability $p_i(x|p)$ and $p_i(y|p)$. With that, we can calculate $\hat{h_i}(pxy)$, which is the estimation of $h_i(pxy)$ assuming that x and y are independent given common parent p.

Figure 3.3: Node join example for independence test



$$
\begin{aligned}
p_i(x|py) &= p_i(x|p) \\
&= \frac{h_i(xp)}{h_i(p)} \\
p_i(y|px) &= p_i(y|p) \\
&= \frac{h_i(yp)}{h_i(p)}
\end{aligned}
\tag{3.3}
$$

$$
\begin{aligned}
\hat{h_i}(pxy) &= p_i(x|py)p_i(y|p) * h_i(p) \\
&= p_i(x|p)h_i(yp) \\
\hat{h_i}(pxy) &= p_i(y|px)p_i(x|p) * h_i(p) \\
&= p_i(y|p)h_i(xp)
\end{aligned}
$$

After that, we query the actual frequency distribution on the knowledge base and do a Pearson's chi-squared independence test. As null hypothesis and alternative hypothesis we have:

- $H_0 = x$ *and* y *are conditionally independent given their common parent* p
- $H_1 = x$ *and* y *are conditionally dependent given their common parent* p

Number of degrees of freedom is the number of buckets minus one:

$$df = k - 1$$

We calculate the critical value χ^2:

$$\chi^2 = \sum_{i=1}^{k} \frac{(h_i - \hat{h_i})^2}{\hat{h_i}} \tag{3.4}$$

[6]

Then it is possible to obtain the p-value and check whether there is enough confidence to reject the null hypothesis H_0.

In other words if we find out that x and y are conditionally independent given p, we could rewrite the equation 3.3 as the following rules being equivalents, with similar accuracy distributions:

$$x \leftarrow py \equiv x \leftarrow p$$
$$y \leftarrow px \equiv y \leftarrow p$$

Therefore, we know that if we have x fixed as head of clauses in the core ILP, and we currently have $x \leftarrow p$ joining the node px with py to obtain the rule $x \leftarrow py$ does not add any valuable information. The same applies for having y, and obtaining $y \leftarrow px$ from $y \leftarrow p$ by joining the same pair of nodes. This property plays an important role in the integration of the correlation lattice into the core ILP as we will explain in more details later in Section (???).

Nevertheless it cannot be safely pruned from the lattice.

This applies to nodes at any level l, with $p \leq l$ parents and C_2^p possible join pairs. If any of the join pairs has enough evidence of being dependent, then 2 edges are created connecting each of the joined nodes to the result of their join

3.4.3 Entropy Divergence Measures

As seen in the previous sections, we are interested in rules whose base-rule has accuracy below threshold, but contains one or multiple specific intervals with accuracy above threshold. For this to happen, we need a rule with non-uniform accuracy distribution, or in other words, divergent body support and rule positive examples distributions.

Therefore, we are interested in adding categories that produces distributions different from their parent nodes'. In order to measure such divergence between distributions, some of the state-of-the-art such as the following ones can be used.

- Kullback-Leibler [7]:

$$D_{KL}(P||Q) = \sum_i \ln\left(\frac{P(i)}{Q(i)}\right) * P(i) \tag{3.5}$$

- Chi-squared (χ^2):

$$D_{\chi^2}(P||Q) = \sum_i \frac{(P(i) - Q(i))^2}{P(i)} \tag{3.6}$$

- Jensen-Shannon [8]:

$$D_{JS}(P||Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M), \tag{3.7}$$

Where P and Q are discrete distributions to be compared and $M = \frac{1}{2}(P + Q)$. In the lattice context, these distributions would be the frequency histogram normalized to 1, and nodes directly connected by edges would have their distributions compared.

As discussed in [8], although Jensen-Shannon is computationally more expensive, it has the advantage of being a symmetric and smoothed version of the Kullback-Leibler measure.

These divergence measures are important for identifying potential accuracy distributions. If the divergence between the frequency distributions of a parent and child node is big, that means that if we have a rule with the parent node as body and the additional literal from the child as head, its accuracy distribution will be not uniform and therefore potentially interesting for searching for refined-rules with numerical intervals.

3.4.4 Heuristics

As seen before, the number of nodes in Correlation Latticegrows exponentially with the number of categorical relations and its constants. For n categorical relations, each with m constants, the total number of nodes is 2^{nm}. As we limit the number of levels in the lattice to l, the total number of nodes reduces to:

$$\sum_{i=1}^{l} \binom{nm}{i} \tag{3.8}$$

if $l = nm$:

$$\sum_{i=1}^{mn} \binom{nm}{i} = 2^{nm} \tag{3.9}$$

Pruning by support is usually not sufficient to make it feasible and it is necessary to apply heuristics to prune it more aggressively.

Pruning by divergence is clearly not safe. Let's suppose we have a root numerical property $r(x, y)$, and two categorical relations $a(x, z)$ with constants A_1 and A_2 and $b(x, w)$ with constants B_1 and B_2. For simplicity, let's assume y is divided in two buckets and root r has an uniform distribution [0.5 0.5] from frequencies [2 2]. It is possible to have ra_1 and ra_2 as well as rb_1 and rb_2 with the same uniform distribution with frequencies [1 1]. Nevertheless when combined we can have the following:

ra_1b_1 : [1.0 0.0]
ra_1b_2 : [0.0 1.0]
ra_2b_1 : [0.0 1.0]
ra_2b_2 : [1.0 0.0]

As shown above, divergence is not a monotonically decreasing measure, thus it can only be used as heuristics.

Moreover, using a divergence measure alone might also be problematic. Histograms with low support are more likely to present a higher divergence than histograms with higher support, supposing that they were drawn from the same original distribution. Consequently, and algorithm using only the divergence as heuristics would end up giving preference to nodes with lower support. Therefore, it is important to use divergence combined with support as pruning heuristics. [Restructure, it's confusing]

As seen seen in Section (???), checking for conditional independence of categories is very insightful and can detect equivalent rules like in Figure 3.3 where we know that $x \leftarrow py \equiv x \leftarrow p$ and $y \leftarrow px \equiv y \leftarrow p$. Nevertheless, we cannot prune the node pxy from the lattice as we x and y might not be independent given pz, and for instance, a rule $x \leftarrow pyz$ might not be equivalent to $x \leftarrow py$.

3.5 Bucketing Numerical Attributes

So far we have mentioned that the root's numerical attribute domain should be discretized by dividing it in buckets in order to build frequency histograms and compare distributions. In this section, we will discuss about how to perform such discretization.

The buckets used throughout the Correlation Latticeshould be consistent, and the bucketing method as well as the boundaries are specified in the very beginning with the root node. It should have as input an arbitrarily defined number of buckets k.

Various different supervised and unsupervised methods for discretization of continuous features are discussed in [9]. To build the lattice we don't use labels for the examples so we just use unsupervised methods.

Supervised methods can be used for finding interesting intervals in the rules, by considering positive and negative examples as labels. However, it is out of the scope of this thesis, thus we will not discuss them in details.

3.5.1 Equal Width

This unsupervised discretization method consists of simply querying both the maximum and minimum values of the root's numerical attribute, then for splitting this interval into k buckets of same width w:

$$w = cfracmax - mink \tag{3.10}$$

It's the simples and most straightforward discretization method, but its main problem is that it is highly sensitive to outliers that may drastically skew the resulting distribution.

3.5.2 Equal Frequencies

In this also unsupervised discretization method, the domain is divided in buckets with equal frequency. That is, given a total frequency m, the domain is discretized in k buckets such that each has frequency $cfracmk$. Multiple frequencies with same numerical attribute value might make it impossible to find bucket boundaries that equally distribute the frequencies.

The greatest advantage of this method is that it always results in a distribution with lowest possible entropy, i.e. distribution is always uniform or as close to uniformity as possible.

3.6 Incorporating Correlation Latticeinto the Core ILP Algorithm

The ILP core learning algorithm requires some modifications in order to support the Correlation Lattice. As stated before, we use the top-down search, starting from the most general clauses then further refining them by introducing new substitutions or literals until stopping criteria is reached.

[...]

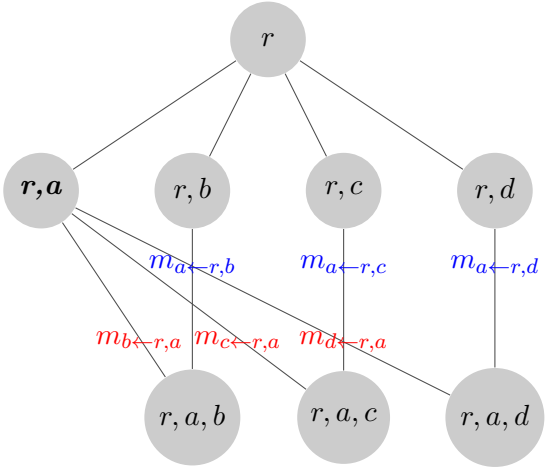
3.6.1 Querying the Correlation Lattice

In the ILP there is a fixed head literal and such literal can and should be included in the Correlation Lattice. In case it is present in the lattice, it plays a key role in the search. Once the root property is added in a clause that does not exceed the accuracy threshold.

In such case we are interested in searching the most interesting literals to add. First step would be to search in the lattice the literals already present in both head and body of the clause that are joined with the root. Nevertheless, what we are not really interested in the benefit of adding a literal to the conjunction of rule's head and body, but in the benefit of adding the head to a body with a new literal.

For example, if we have the clause $a \leftarrow r$, r is the root of a Correlation Latticeand the head a is included in it. After searching in the graph for the literals joined with root present in the clause we would come to node ra. We are not searching for the most interesting child from ra, but for the children, which has a parent without the head literal that has the has the highest interestingness measure when adding the head.

Figure 3.4: Node join example for independence test



As shown in Figure 3.4, if we can add b, c or d to ra, we are interested about the measures in blue instead of the ones in red. Nevertheless, searching for values for $m_{a,rb}$, $m_{a,rc}$ and $m_{a,rd}$ is a bit tricky. It is necessary to check for all children of ra, which ones have parents without head literal a, then gather all the measures and rank them.

In order to make it simpler, we can create a map during lattice build. This map would contain an entry for all node it is joined to, with head as key and literals to add sorted by measure as value. So for example when we are obtaining the node rab by joining ra and rb, we add to ra an entry with a as key, b as new literal and associated measure $m_{a,rb}$ and we also add to rb an entry with b as key, a as new literal ad associated measure $m_{b,ra}$.

In the example of Figure 3.4, ra would contain the following map:

a	b $[m_{a,rb}]$
	c $[m_{a,rc}]$
	d $[m_{a,rd}]$

As in this example the node ra has only two literals, and the root r cannot be the head, we use the node ra_1b_1 in Figure 3.2 to better illustrate it:

a1	c1 $[m_{a_1,rb_1c_1}]$
	c2 $[m_{a_1,rb_1c_2}]$
b1	c1 $[m_{b_1,ra_1c_1}]$
	c2 $[m_{b_1,ra_1c_2}]$

In a node at level l with $l+1$ literals (one being the root), we can have up to l keys in the map. Therewith, we can much more easily find the best suggestions for any possible head literal without having to visit other nodes.

First we need to introduce a new refinement operator that extracts the support and accuracy distributions along a chosen numerical property and searches for interesting ranges. With that there are

In the specialization step, we can detect whether the clause contains any numerical property, which is root of a Correlation Lattice. If so, it can search for interesting ranges numerical ranges,

Algorithm 4: Refinement step

Input: *clause*: Set of literals from the clause, $lattice(r_{root})$: Set of existent Correlation Lattices, lit_{body}: Literal in clause that will be joined with new Literal, lit_{new}: Literal to be added to the clause, arg_{body}: Join argument from body literal, arg_{new}: Join argument from new literal

Output: True if arg_i from r_i joins with arg_j from r_j, False otherwise

$r_{new} \leftarrow lit_{new}.relation$;

if $arg_{new} = 1$ ***and*** r_{new} *is numerical* ***and*** $\exists\ lattice(r_{new})$ **then**

  $node \leftarrow lattice(r_{new}).root$.`search(`$head$`)` ;

  foreach $lit_i \in clause.body$ **do**

    $node \leftarrow node$.`search(`lit_i`)` ;

  `checkNumericalRanges(`$clause$,lit_{new},$node$`)` ;

Algorithm 5: checkNumericalRanges

Input:

Output:

$v \leftarrow$ Numerical variable from lit_{new} ;

$acc \leftarrow$ `queryAccuracyDistribution(`$clause$,v`)` ;

$sup \leftarrow$ `querySupportDistribution(`$clause$, v`)` ;

$intervals \leftarrow$ `searchInterestingIntervals(`acc,sup,$accTS$,$supTS$`)` ;

foreach $interval_i \in intervals$ **do**

  $newClause \leftarrow \{clause \cup lit_{new} \cup \{v \in interval_i\}\}$;

  Add $newClause$ to rule tree ;

$suggestions \leftarrow node.getSuggestions(clause.head)$;

foreach $lit_i \in suggestions$ ***and*** $i \leq k$ **do**

  $node_i \leftarrow node$.`search(`lit_i`)` ;

  `checkNumericalRanges(`$clause \cup lit_i$,lit_{new},$node_i$`)` ;

Chapter 4

Algorithmic Framework

4.1 Knowledge Base Backend

For storing and retrieving RDF data we use RDF3X [10]. It has the advantage of being specialized and optimized for RDF data, using a strong indexing approach with compressed B^+-Tree indices for each of six permutations of *subject (S)*, *predicate (P)* and *object (O)*: *SPO*,*SOP*,*OSP*,*OPS*,*PSO* and *POS*.

RDF3X particularities...

4.2 Preprocessing

4.2.1 Correlation Lattice

4.2.1.1 Graph Node

Every node essentially contains the following attributes:

- Set of pointers to parent nodes
- Set of pointers to child nodes
- Set of pointers to constant nodes
- Histogram with facts distribution over root numerical property

4.2.1.2 Building the Correlation Lattice

For building the Correlation Lattice, we start with the root node, which has a numerical property as literal and no constants assigned, e.g. *hasIncome(x,y)*. We then query the distribution of positive examples over the property in the whole Knowledge Base.

SELECT COUNT ?y WHERE { ?x <hasIncome> ?y } GROUP BY (?y)

It's also necessary to specify the bucketing technique and the number of buckets in order to extract the histogram from the obtained query results. These buckets are used to build the histograms of all nodes in the graph.

Afterwards, we select the the categorical properties that will be used in the lattice. For each of the selected properties, we join them with the root numerical property (for simplicity we'll assume all the categorical properties are joined with both 1^{st} arguments) and we query the distribution again. In the first level, it's necessary to extract a histogram for each of the categorical constants in the selected properties. Therefore, it's a good strategy to group the results also by these categorical constants so If we select *hasEducation* for example, we would then fire the following SPARQL query:

SELECT COUNT ?z ?y WHERE { ?x <hasIncome> ?y . ?x <hasEducation> ?z }
GROUP BY (?z,?y)

With such query, it's possible to extract a histogram for the node *hasIncome(x,y)hasEducation(x,z)* and its correspondent constants.

4.2.1.3 Searching Rules in Correlation Lattice

In the Correlation Latticeitself, it's possible to extract valuable rules. Given its characteristic of all the relations being joined on the same root argument, those rules represent how different categories are related along root's numberical constants.

For every non-root node in the Correlation Lattice, any of its parents can be seen as rule's body and the remaining literal as head, e.g.:

Let's denote a_i as a relation $a(x, y)$ with constants A_i for y, so for example $a_1 \equiv a(x, A_1)$:

For the node $ra_1b_1c_1$ with parents ra_1b_1, ra_1c_1 and rb_1c_1, we can extract and easily evaluate three rules:

$Rule_1: \quad a_1 \leftarrow b_1 c_1 r$
$supp_i(Rule_1) = h_i(r a_1 b_1 c_1)$
$acc_i(Rule_1) = \frac{h_i(r a_1 b_1 c_1)}{h_i(r b_1 c_1)}$

$Rule_2: \quad b_1 \leftarrow a_1 c_1 r$
$supp_i(Rule_2) = h_i(r a_1 b_1 c_1)$
$acc_i(Rule_2) = \frac{h_i(r a_1 b_1 c_1)}{h_i(r a_1 c_1)}$

$Rule_3: \quad c_1 \leftarrow a_1 b_1 r$
$supp_i(Rule_3) = h_i(r a_1 b_1 c_1)$
$acc_i(Rule_3) = \frac{h_i(r a_1 b_1 c_1)}{h_i(r a_1 b_1)}$

Subsequently we can analyze the frequency and confidence distributions and determine whether any of the rules are interesting, using any of the techniques discussed in [???] and find possible interesting intervals.

4.2.1.4

Chapter 5

Experiments

***Roughly explaining the experiments I did so far, we want to evaluate whether the heuristics are good. So we use US Census data to build a lattice (for income property) with the limitation of having up to n nodes at each level. So far I compared the KL-divergence*Support with support only as measures. For every level I sort the nodes by the measure (nodes with multiple parents will have multiple KL-divergence, in this case I take the maximum one) and try to join the nodes with highest measures first until n nodes for the next level is reached. In other words, I greedily build the a lattice with limited number of nodes per level.

After that, I try to extract rules with interesting ranges directly from the lattice and test them against a test partition. Interesting rules are defined as in ReferencesInteresting Rules with support threshold of 25 examples and accuracy threshold of 0.75. After that, I compare the number of interesting rules learned, the time taken to build the lattice and the accuracy and accuracy gain compared to the base rule.

Accuracy and accuracy gain seem to depend exclusively on the accuracy threshold, so they are roughly the same for both measures. Processing time is much smaller for the KL-divergence as it doesn't simply focus on huge relations and as expected, it also presents a greater number of rules with interesting ranges (at least for smaller number of nodes per level).

Hopefully I'll soon have results from applying the lattice in the core ILP. I managed to overcome some implementation problems I had when creating the lattice in YAGO.

Figure 5.1: Lattices with 3 levels

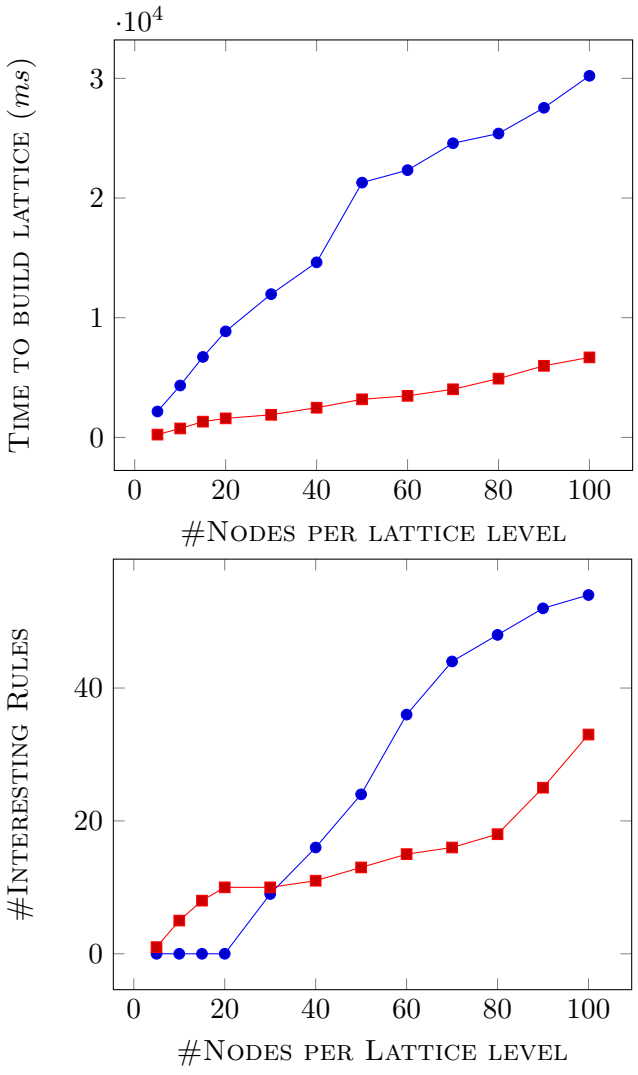
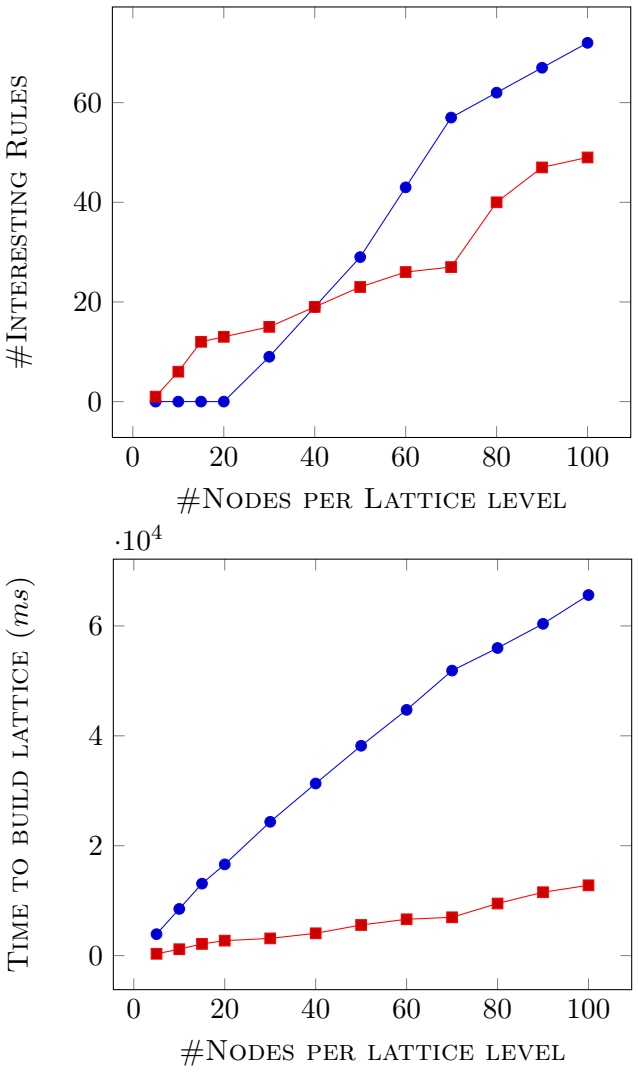


Figure 5.2: Lattice with 4 levels



List of Figures

List of Tables

Bibliography

[1] John W. Lloyd. *Foundations of Logic Programming, 2nd Edition*. Springer, 1987. ISBN 3-540-18199-7.

[2] Nada Lavrac and Saso Dzeroski. A reply to pazzani's book review of "inductive logic programming: Techniques and applications". *Machine Learning*, 23(1):109–111, 1996.

[3] Sergey Brin, Rajeev Rastogi, and Kyuseok Shim. Mining optimized gain rules for numeric attributes. In *Proceedings of the Fifth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 135–144. ACM Press, 1999.

[4] Toon Calders, Christian W. Günther, Mykola Pechenizkiy, and Anne Rozinat. Using minimum description length for process mining. In *SAC*, pages 1451–1455, 2009.

[5] Nada Lavrac and Saso Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York, 1994. URL http://www-ai.ijs.si/SasoDzeroski/ILPBook/.

[6] Szymon Jaroszewicz and Dan A. Simovici. Pruning redundant association rules using maximum entropy principle. In *In Advances in Knowledge Discovery and Data Mining, 6th Pacific-Asia Conference, PAKDD?02*, pages 135–147, 2002.

[7] S. Kullback and R. A. Leibler. On information and sufficiency. *Ann. Math. Statist.*, 22(1):79–86, 1951.

[8] J. Briet and P. Harremos. Properties of classical and quantum jensen-shannon divergence. *Physical Review A: Atomic, Molecular and Optical Physics*, 79(5), May 2009. ISSN 1050-2947.

[9] James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In *MACHINE LEARNING: PROCEEDINGS OF THE TWELFTH INTERNATIONAL CONFERENCE*, pages 194–202. Morgan Kaufmann, 1995.

[10] Thomas Neumann and Gerhard Weikum. The rdf-3x engine for scalable management of rdf data. *The VLDB Journal*, 19(1):91–113, February 2010. ISSN 1066-8888. doi: 10.1007/s00778-009-0165-y. URL http://dx.doi.org/10.1007/s00778-009-0165-y.