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 7. 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 7. 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 RDF, 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 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.

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:

   - $marriedTo(x, y) \leftarrow marriedTo(y, x), (x \neq y)$
   - $grandChildOf(x, y) \leftarrow childOf(x, z), childOf(z, y)$
   - $parentOf(x, y) \leftarrow childOf(y, x)$
   - $(z = y) \leftarrow wasBornIn(x, z), wasBornIn(x, y)$

2. *Soft Rules*: Weighted Datalog rules that frequently, but not always hold in the real world. As they also produce incorrect information, each rule itself must have a confidence value which should be applied to derived facts, for example married people live in the same place as their partner has confidence 0.8:

$$livesIn(x, y) \leftarrow marriedTo(x, z)livesIn(z, y)\ [0.8]$$

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.*, both facts with confidence 1, we could then apply this soft rule to derive the fact *livesIn(MichelleObama, WashingtonDC)* with confidence 0.8.

Such rules are rarely known beforehand, or are too expensive to be manually extracted. Nevertheless, the data itself can be used to mine these rules using *Inductive Logic Programming (ILP).*

ILP 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.

Moreover, . In such case, it's necessary to arbitrarily reduce the search by restricting the set of constants to be included in ... [talk more about ILP?]

1.1 Motivation

Given the huge size of search space and the great interestingness of rules with constants, we need to smartly prune constants or combinations of constants that doesn't add interesting information to the hypothesis, learning datalog rules with ILP can be already extremely costly, especially if we consider hypothesis containing properties with constants.

Numerical properties are a special case, and they need to be treated differently. 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)<-livesIn(x,y)hasPopulation(y,193946886)

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

speaks(x,Portuguese)<-livesIn(x,Brazil)

Of course, if we have to choose between one of the two rules, the latter one would be preferred as it's 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's interesting to split the attribute's domain into k buckets and group examples by similarity of the numerical attribute. then 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:

$$isMarriedTo(x, y) \leftarrow hasAge(x, z)$$

and then we split z into three buckets:

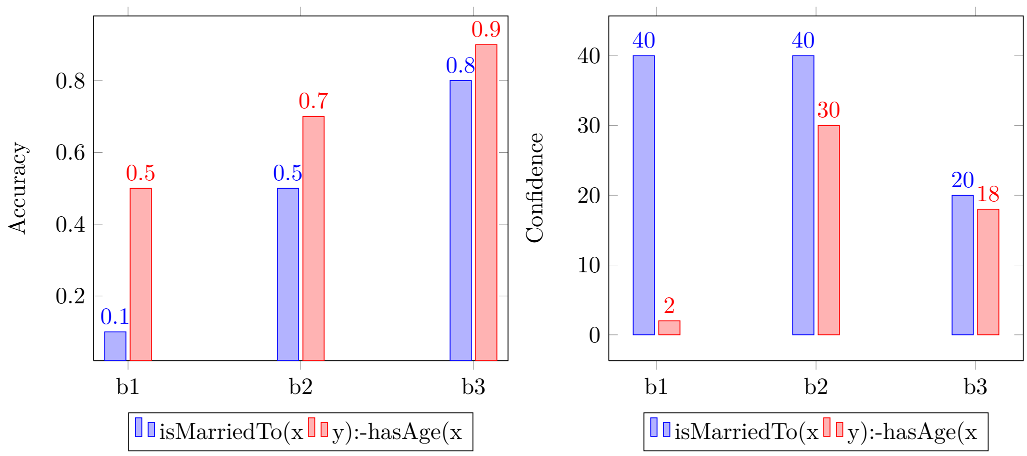
- $k = 1 : z \in [0, 20]$
- $k = 2 : z \in (20, 40]$
- $k = 3 : z \in (40, \infty]$

we then test the hypothesis for each of the three buckets and obtain

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

- $isMarriedTo(x, y) \leftarrow hasAge(x, z)hasChild(x, a)$
  support=50, confidence=0.625
- $isMarriedTo(x, y) \leftarrow hasAge(x, z)hasChild(x, a), z \in [0, 20]$
  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 some literals 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 relation with no correlation to the rule might not generate any gain. Thus, it's necessary to carefully choose the relations and discard the uncorrelated ones.

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, like in frequent set mining.

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 Correlation 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

2.1 Logic Programming

2.2 Inductive Logic Programming

2.3 First Order Inductive Learning

2.4 Mining Optimized Rules for Numeric Attributes

[1]

2.5 Minimum Description Length

[2]

2.6 Semamtic Web

2.7 Linked Open Data

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 Rules

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 accuracy thresholds.

Searching numerical intervals for a base-rule that already satisfies accuracy threshold is uninteresting. The base rule itself implicitly specifies an interval which covers the whole numerical attribute domain... [Further develop]

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 1^{st} argument variable (assuming that the numerical attribute is in the 2^{nd} argument).

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 $y_i \in Y$:

$X_i \subset X \quad s.t. \quad X_i = \{x \in X \,|\, f(x) = y_i,\, y_i \in Y\}$
$X = \bigcup_{i=1}^{n} X_i$ 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 doesn't 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't 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 of the four join patterns, 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 of the four join patterns between two relations:

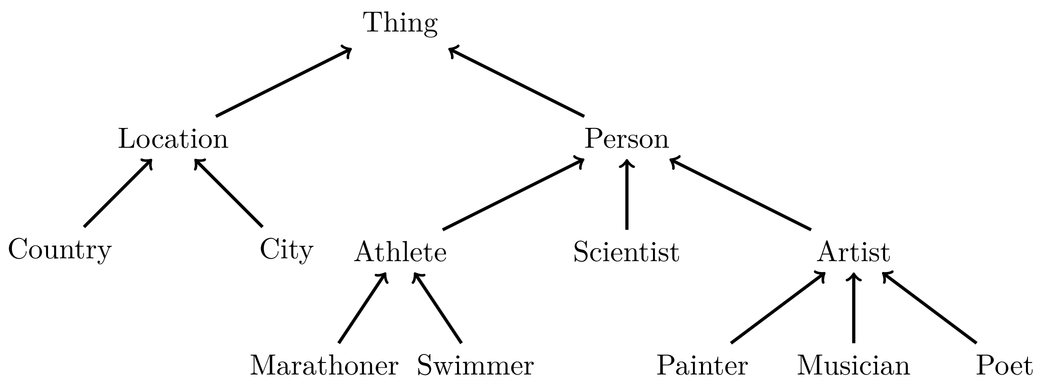
- Argument 1 on Argument 1: e.g. *hasIncome(**x**,y)hasAge(**x**,z)*
- Argument 1 on Argument 2: e.g. *hasIncome(**x**,y)isMarriedTo(z,**x**)*
- Argument 2 on Argument 1: e.g. *livesIn(y,**x**)isLocatedIn(**x**,z)*
- Argument 2 on Argument 2: e.g. *livesIn(y,**x**)wasBornIn(z,**x**)*

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 1$^{\text{st}}$ argument) and domain (type of 2$^{\text{nd}}$ 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's a really 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's allowed to join the pair of relations on the given joining arguments, or in other words, the type hierarchy doesn't prevent them from joining. The pseudo-code for performing such task is shown in the Algorithm 1:

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

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's worth to that beforehand,

and that's 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 doesn't 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't need to test such hypothesis in the core ILP algorithm.

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 ∈ r do
    L_{1,1}(r_i) ← r_i ;
    L_{2,2}(r_i) ← r_i ;
    L_{1,2}(r_i) ← ∅ ;
    L_{2,1}(r_i) ← ∅ ;
// For every possible pair of relations
foreach r_i ∈ r do
    foreach r_j ∈ r and j ≥ i do
        if r_i ≠ 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) ← L_{1,1}(r_i) ∪ r_j ;
                L_{1,1}(r_j) ← L_{1,1}(r_j) ∪ r_i ;
            if checkTypes(r_i, r_j, 2, 2) and checkSupport(r_i, r_j, 2, 2) then
                L_{2,2}(r_i) ← L_{2,2}(r_i) ∪ r_j ;
                L_{2,2}(r_j) ← L_{2,2}(r_j) ∪ r_i ;
        if checkTypes(r_i, r_j, 1, 2) and checkSupport(r_i, r_j, 1, 2) then
            L_{1,2}(r_i) ← L_{1,2}(r_i) ∪ r_j ;
            L_{2,1}(r_j) ← L_{2,1}(r_j) ∪ r_i ;
        if checkTypes(r_i, r_j, 2, 1) and checkSupport(r_i, r_j, 2, 1) then
            L_{2,1}(r_i) ← L_{2,1}(r_i) ∪ r_j ;
            L_{1,2}(r_j) ← L_{1,2}(r_j) ∪ 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 like 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")

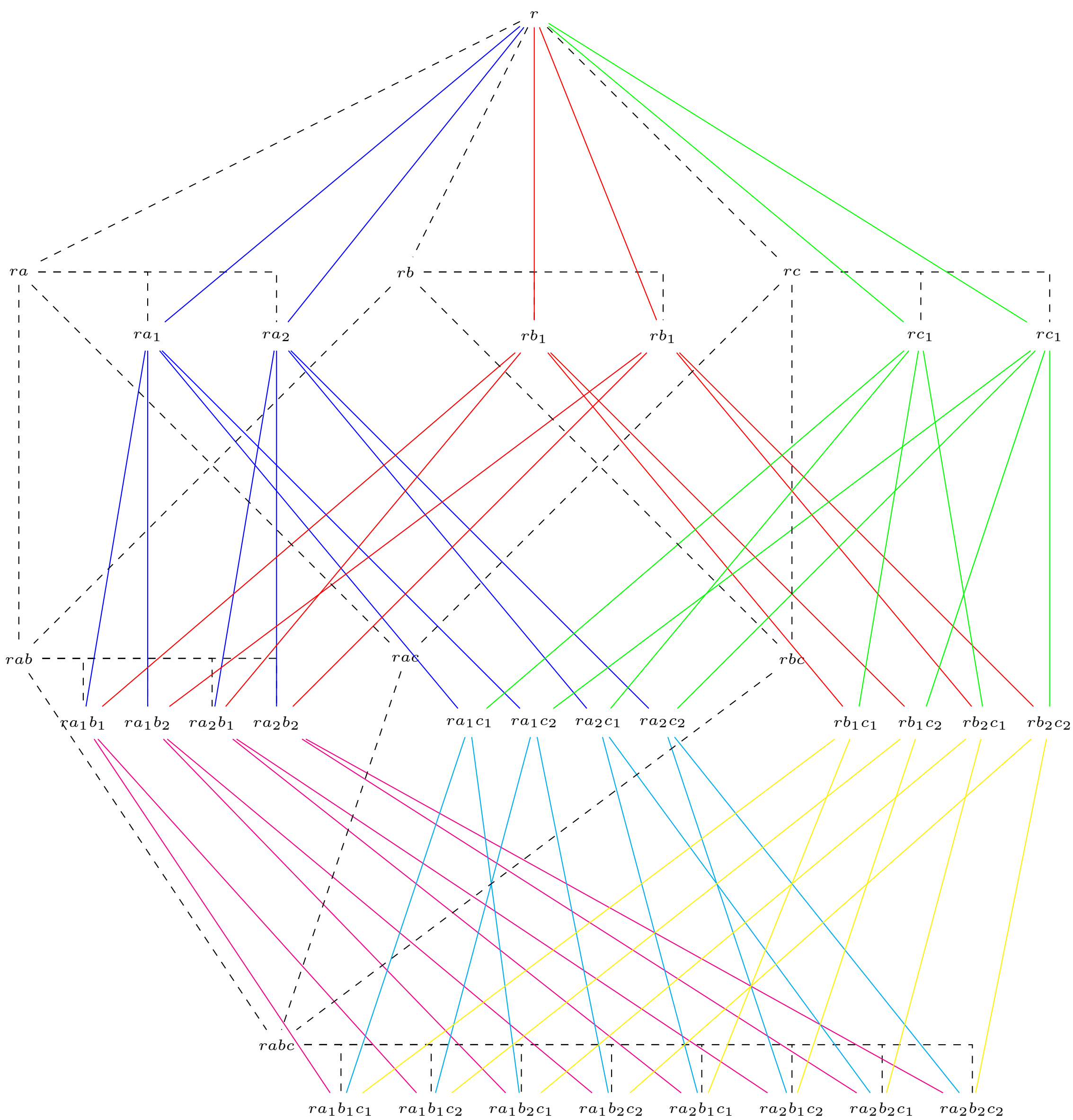
hasIncome(x,y)wasBornInMonth(x,"February"),hasEducation(x,"Preschool")
hasIncome(x,y)wasBornInMonth(x,"February"),hasEducation(x,"Kindergarten")
...
hasIncome(x,y)wasBornInMonth(x,"February"),hasEducation(x,"Doctorate Degree")

...

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.

Figure 3.2: Correlation Lattice example



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 ([3]), 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 doesn't reach the minimum support threshold.

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

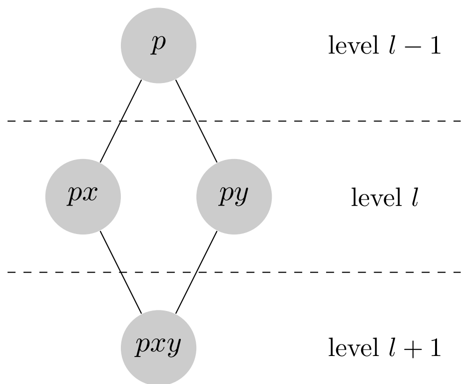
3.4.2.2 Independence Checks

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's 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 3.3: Node join example for independence test



For the example shown in figure 3.3, we can calculate the conditional probability $p_i(x|p)$ and $p_i(y|p)$ assuming conditional independence given common parent p in order to estimate $\hat{h_i}(pxy)$:

$$
\begin{aligned}
p_i(x|py) &= p_i(x|p) \\
&= \frac{h_i(xp)}{h_i(p)} \\
p_i(y|px) &= p_i(y|p) \\
&= \frac{h_i(yp)}{h_i(p)}
\end{aligned}
\tag{3.1}
$$

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

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:

$$\chi^2 = \sum_{i=1}^{k} \frac{(h_i - \hat{h}_i)^2}{\hat{h}_i} \tag{3.2}$$

[4]

Then it's possible to obtain the p-value and check whether there's 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 3.1 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 $x \leftarrow py$ doesn't add any 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 Distribution Divergence Measures

As seen in the previous sections, we are interested in rules whose base-rule has accuracy bellow 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 [5]:

$$D_{KL}(P||Q) = \sum_i \ln\left(\frac{P(i)}{Q(i)}\right) * P(i) \tag{3.3}$$

- Chi-squared (χ^2):

$$D_{\chi^2}(P||Q) = \sum_i \frac{(P(i) - Q(i))^2}{P(i)} \tag{3.4}$$

- Jensen-Shannon [6]:

$$D_{JS}(P||Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M), \tag{3.5}$$

Where P and Q are the distributions to be compared and $M = \frac{1}{2}(P + Q)$

As discussed in [6], although Jensen-Shannon is computationally more expensive, it has the advantage of being a symmetric and smoothed version of the Kullback-Leibler measure.

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}. Pruning by support is usually not sufficient to make it feasible and it's 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's 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's 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 Incorporating Correlation Latticeinto 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.

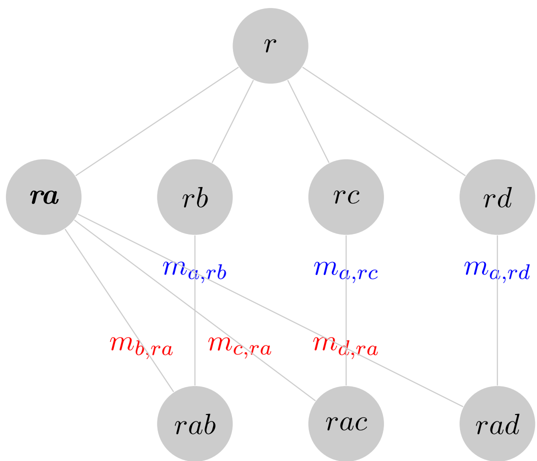
3.5.1 Querying the Correlation Lattice

In the ILP there's a fixed head literal and such literal can and should be included in the Correlation Lattice. In case it's present in the lattice, it plays a key role in the search. Once the root property is added in a clause that doesn't 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's 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's 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 $[m_{a,rd}]$
	d $[m_{a,rc}]$

As in this example the node ra has only two literals, and the root r can't 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 [7]. 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:

$Rule_1: \quad a_1 \leftarrow b_1c_1r$
$supp_i(Rule_1) = h_i(ra_1b_1c_1)$
$acc_i(Rule_1) = \frac{h_i(ra_1b_1c_1)}{h_i(rb_1c_1)}$

$Rule_2: \quad b_1 \leftarrow a_1c_1r$
$supp_i(Rule_2) = h_i(ra_1b_1c_1)$
$acc_i(Rule_2) = \frac{h_i(ra_1b_1c_1)}{h_i(ra_1c_1)}$

$Rule_3: \quad c_1 \leftarrow a_1b_1r$
$supp_i(Rule_3) = h_i(ra_1b_1c_1)$
$acc_i(Rule_3) = \frac{h_i(ra_1b_1c_1)}{h_i(ra_1b_1)}$

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

Figure 5.1: levels=3

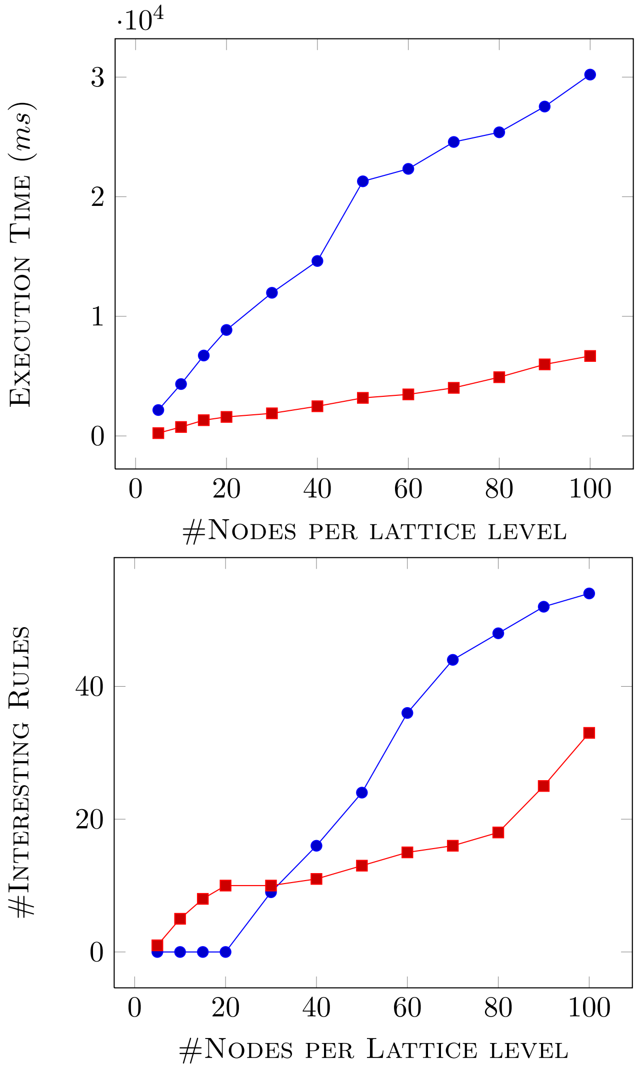
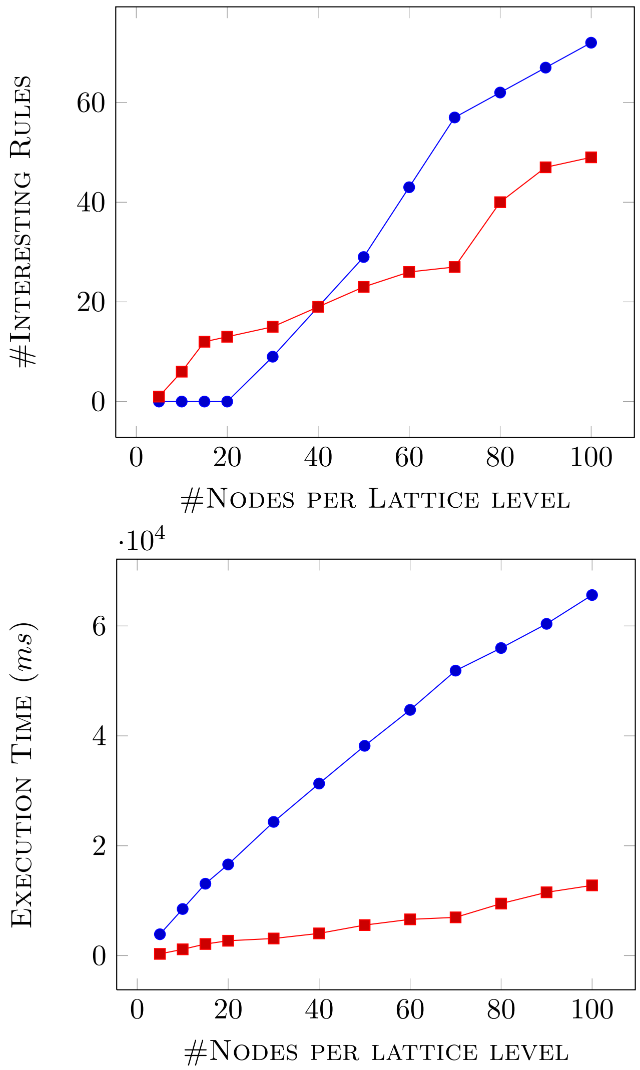


Figure 5.2: levels=4



List of Figures

List of Tables

Bibliography

[1] 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.

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

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

[4] 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.

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

[6] 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.

[7] 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.