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. Gerhard Weikum

März / March 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. März 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. März 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 first-order 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 considered. Therefore, it usually requires very aggressive pruning for making it feasible. Moreover, searching for rules with numerical intervals requires very expensive queries, and frequently do not bring any significant confidence gain. We propose a preprocessing step and an extension to ILP top-down algorithm, with the objective of efficiently learning rules with interesting intervals for numerical attributes, i.e., rules which present a significant gain when restricting a numerical attribute variable to a specific interval, combined with categorical constants. In the preprocessing step, we build a lattice that expresses the correlations between a root numerical attribute and multiple categorical relations and their constants, while in the refinement step of ILP, we query this lattice in order to know whether it is interesting to search for numerical intervals, as well as obtain a list of refinement suggestions ordered by a defined an interestingness measure. This thesis discusses how to efficiently build the correlation lattice, how to incorporate it in the core ILP learning algorithm, and compares different iterestingness measures. Also, we make experiments on large-scale linked data sources in order to evaluate the proposed approach.

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. Gerhard Weikumfor 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

Abstract vi

Acknowledgements viii

1 Introduction 1
- 1.1 Motivation . . . 3
- 1.2 Contributions . . . 5
- 1.3 Outline . . . 6

2 Related Work 7
- 2.1 Logic Programming . . . 7
- 2.2 Inductive Logic Programming . . . 8
  - 2.2.1 Searching the Hypothesis Space . . . 10
  - 2.2.2 Top-Down ILP . . . 11
  - 2.2.3 ILP Complexity . . . 13
- 2.3 Association Rule Mining . . . 13
  - 2.3.1 Measures . . . 14
  - 2.3.2 Anti-monotonicity of Support . . . 15
  - 2.3.3 Itemset Lattice . . . 15
  - 2.3.4 Apriori Algorithm . . . 16
- 2.4 Discretization of Numerical Attributes . . . 17
  - 2.4.1 Equal Width . . . 18
  - 2.4.2 Equal Frequencies . . . 19
- 2.5 Mining Optimized Rules for Numeric Attributes . . . 19
- 2.6 Information-Theoretic Measures . . . 21
  - 2.6.1 Shannon's Entropy . . . 22
  - 2.6.2 Kullback-Leibler Divergence . . . 23
  - 2.6.3 Mutual Information . . . 23
- 2.7 Semantic Web Applications . . . 24
- 2.8 Linked Open Data Applications . . . 25

3 Learning Rules With Numerical and Categorical Attributes 29
- 3.1 Interesting Rule Definition . . . 29
- 3.2 Categorical Property Definition . . . 31
  - 3.2.1 Discretizing Numerical Properties into Categories . . . 32

3.2.2 Combining Categorical Property with Linking Relation 32
3.2.3 Absence or Presence of a Property as a Category 33
3.2.4 Notation Used 34
3.3 Interestingness Measure 35
3.4 Preprocessing 36
3.4.1 Relation Preprocessing 36
3.4.2 39
3.5 Correlation Lattice 39
3.5.1 Building the Lattice 41
3.5.2 Safe Pruning Opportunities 42
3.5.3 Entropy Divergence Measures 46
3.5.4 Heuristics 47
3.6 Bucketing the Numerical Attributes 49
3.7 Incorporating Correlation Lattice into the Core ILP Algorithm 49
3.7.1 Querying the Correlation Lattice 49

4 Algorithmic Framework **53**
4.1 Knowledge Base Back-end 53
4.2 Preprocessing 54
4.2.1 Correlation Lattice Data Structure 54

5 Experiments **57**

List of Figures **57**

List of Tables **63**

Bibliography **65**

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 first 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 the knowledge base domain. With that, it is possible to resolve contradictions as well as strengthen or weaken their confidence values. It is also possible to derive new facts that are originally not existent due to incompleteness. Such inference rules can either represent consistency constraints (hard rules), or rules that hold for most but all cases in real world (soft rules).

In this thesis, we consider only inference rules that can be represented as safe Datalog clauses. In a nutshell, a Datalog clause is a logic program clause composed by a head predicates and a list of body predicates, each predicate is a tuple with variables or constants as arguments. A Datalog clause is safe if all the variables present in the head

are also present in the body, moreover, negated predicates are allowed in the body as long as every variable present in a negated predicate is also present in a non-negated predicate.

An example of safe Datalog rule, which states that married people usually live in the same place as their spouse, could be written as follows:

$$livesIn(X, Y) \text{ :- } marriedTo(X, Z), livesIn(Z, Y)$$

If we have an incomplete knowledge base, which lacks information about where *Michelle Obama* lives, but contains the facts $marriedTo(michelleObama, barackObama)$ and $livesIn(barackObama, washingtonDC)$, we could then apply the aforementioned 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). 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 hypotheses 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 be 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 $speaks(X, spanish)$:- $livesIn(X, mexico)$.

The problem of that, is that for such an example, it would be necessary to test rules with all combinations of countries and languages, which surely can lead to a very expensive process when applied on large databases. Given the huge size of the 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 have any correlation to the hypothesis.

The literal $bornInMonth(X, april)$, for example, does not have any correlation to $speaks(X, spanish)$:- $livesIn(X, mexico)$ or $speaks(X, english)$:- $livesIn(X, australia)$. So if we ad such literal to the body of any of theses clauses, it would most likely not

cause any change to the confidence, but crtanly a reduction in the support, therefore it is not interesting to test these hypotheses with the addition of such uncorelated literal.

1.1 Motivation

For numerical properties, it might be also relevant to search for interesting constants. Nevertheless, for very large or infinite numerical domains such as the real numbers for example, setting a numerical constant as an individual value will very likely have a single entity associated to it. In such a case, it would be equivalent to simply specifying a constant for this given entity without the necessity of adding the numerical property

For example, it is extremely unlikely that a country has the exact same GDP or population as any other country. If we have the following rule with the attribute $hasPopulation$, whose domain is the positive integer numbers $\mathbb{N}^+$:

$$speaks(X, portuguese) \text{ :- } livesIn(X, Z), hasPopulation(Z, 193946886)$$

As 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 discretize the numerical attribute's domain into categories, and subsequently check if any of the buckets present different confidence 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:

- $b_1 : Z \in [0, 20]$

- $b_2 : Z \in (20, 40]$
- $b_3 : Z \in (40, \infty]$

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 b_2 and b_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:

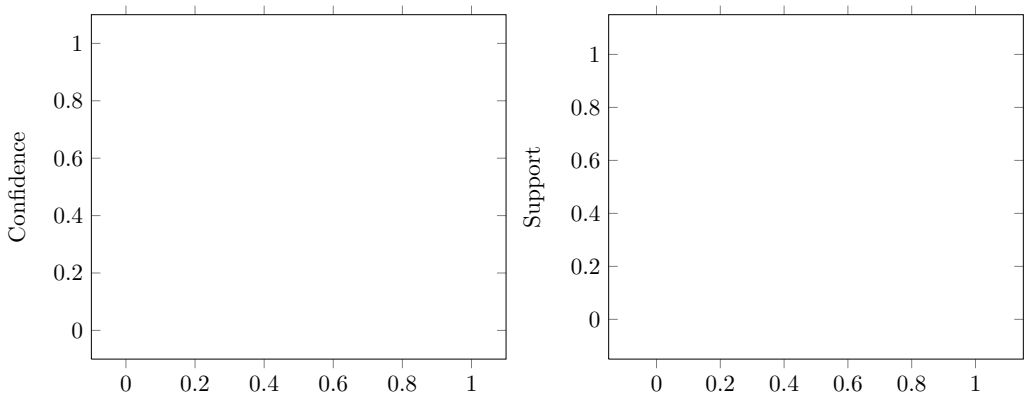
- $sMarriedTo(X, Y)$:- $hasAge(X, Z), hasChild(X, A)$
  support=50, confidence=0.625

Refined-rules:

- $isMarriedTo(X, Y)$:- $hasAge(X, Z), hasChild(x, a), Z \in [0, 20]$
  support=2, confidence=0.5
- $isMarriedTo(X, Y)$:- $hasAge(X, Z), hasChild(x, a), Z \in (20, 40]$
  support=30, confidence=0.7
- $isMarriedTo(X, Y)$:- $hasAge(X, Z), hasChild(x, a), Z \in (40, \infty]$
  support=18, confidence=0.9

Adding a given predicate to the rule might not bring any gain or even loss in confidence to the base-rule, but when bucketing per age, present a different confidence and support distribution and might even produce gain in confidence 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 lattice for each numerical property we want to for interesting intervals. In each graph, that has a numerical property as its root, we first query the examples distribution on the numerical attribute, and build a histogram by discretizing 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 the categories and check whether they still produce interesting sub-populations creating a lattice, like in frequent set mining apriori algorithm.

We compare the distributions obtained from the frequencies histograms using information theoretical measures, such as Kullback-Leibler divergence, as well as independence checks, such as Pearson's Chi-squared, in order to measure the interestingness of adding literals to clauses.

Moreover, 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 example 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 confidence 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.

In summary, we present in this thesis three main contributions:

- A measure of the interestingness of searching for a specific interval for a rule's numerical attribute variable.
- The correlation lattice and its pruning methods.

- Experiments on Linked Open Data datasets.

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 we introduce important logic programming concepts, and in section 2.2 we discuss in more details how Inductive Logic Programming (ILP) works. Later, in section 2.3 we briefly present association rule mining highlighting concepts relevant to this thesis. In section 2.4, we talk about discretization of numerical attributes and discuss the most popular unsupervised methods. After that, we present in section 2.5 a method for mining rules optimized numerical intervals. In section 2.6, we introduce the information-theoretical measures used in this thesis, and finally, in sections 2.7 and 2.8, we give a quick overview about Semantic Web and Linked Open Data.

2.1 Logic Programming

In this section, basic logic programming and deductive database terminology such as *literal*, *clause*, *program clause*, *datalog clause* and *hypothesis* will be presented. We use terminology and definitions described in Lloyd [1] and Lavrac and Dzeroski [2].

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 (including constants) are lowercase letters also followed by a string of lowercase and/or digits.

Following the 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.

Besides that, it is important to mention that Datalog rules support not only relational predicates but also arithmetical, such as $=$, $\neq$, $>$, $<$, $\geq$, $\leq$, which improves the expressiveness of the language allowing that, for instance, numerical attribute variables have their domain restricted to a specific interval.

2.2 Inductive Logic Programming

Inductive Logic Programming (ILP) is a machine learning technique that combines 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 clauses in the hypothesis aim at recognizing a concept $\mathcal{C}$, which is defined as a subset of objects $\mathcal{C} \subseteq \mathcal{U}$. For example, $\mathcal{U}$ may be the set of all students in a University and $\mathcal{C}$ all the students studying Informatics. Concepts and objects are described in a description

language $\mathcal{L}$, typically a subset of first-order logic, but various languages have also been used in ILP, including propositional logic and first order predicate calculus.

The training data for learning a concept $\mathcal{C}$ is a set of examples $\mathcal{E}$, where each examples is a grounded fact labeled as positive ($\oplus$) if the object is an instance of $\mathcal{C}$, or negative ($\ominus$) otherwise. We denote the set of positive examples as $\mathcal{E}^+$ and the set of negative examples as $\mathcal{E}^-$.

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.

As defined in Lavrac and Dzeroski [3], a hypothesis $\mathcal{H}$ is covers 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 $\mathcal{H}$ is a set of program clauses and an example is a ground fact, it means that e is entailed by $\mathcal{B} \cup \mathcal{H}$.

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

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

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

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

The objective of learning a concept requires a hypothesis that is complete and consistent with respect to the given concept training examples and background knowledge. The example shown in table 2.1 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:

Table 2.1: A simple ILP problem: learning the *daughter* relation [3] .

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).

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

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 $\mathcal{B}$.

Lavrac and Dzeroski [3] structures the search space with partial ordering 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 generally as c', or in other words, c' is a specialization of c.

For example, if we have the following c and c':

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

we know that for that c' is a specialization of c because for $\theta = \emptyset$, $c \subset c'$ since:

$$\{daughter(X, Y), \neg parent(Y, X)\} \subset \{daughter(X, Y), \neg female(X), \neg parent(Y, X)\}$$

To better illustrate this, if we have:

$$c = livesIn(X, Y) \text{ :- } marriedTo(X, Z), livesIn(Z, Y)$$
$$c' = livesIn(X, germany) \text{ :- } marriedTo(X, Z), livesIn(Z, germany)$$

we know that c' is a specialization of c because with the substitution $\theta = \{Y/germany\}$, $c\theta = c'$.

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

- When generalizing 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 any 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 Lavrac and Dzeroski [3].

2.2.2 Top-Down ILP

The search space of program clauses can be viewed as a lattice, structured by θ-subsumption generality ordering. Such a lattice is called a *refinement graph*, which can be used to direct the search from most general to specific hypothesis. Figure 2.1 illustrates a part of the refinement graph for the family relation example shown in table 2.1.

In such a graph, nodes are program clauses and arcs are refinement operations, which can be of two kinds: apply a substitution on the clause; add a literal to the body of the clause.

The top-down algorithm consists basically of a specialization loop that refines a clause ensuring consistency embedded inside a covering loop that adds clauses to the hypothesis ensuring completeness. Each loop has a stopping criterion, the covering loop iterates until it satisfies a sufficiency criterion and the specialization loop iterates until it satisfies a necessity criterion. The algorithm 1 shows in more details how the generic top-down ILP works.

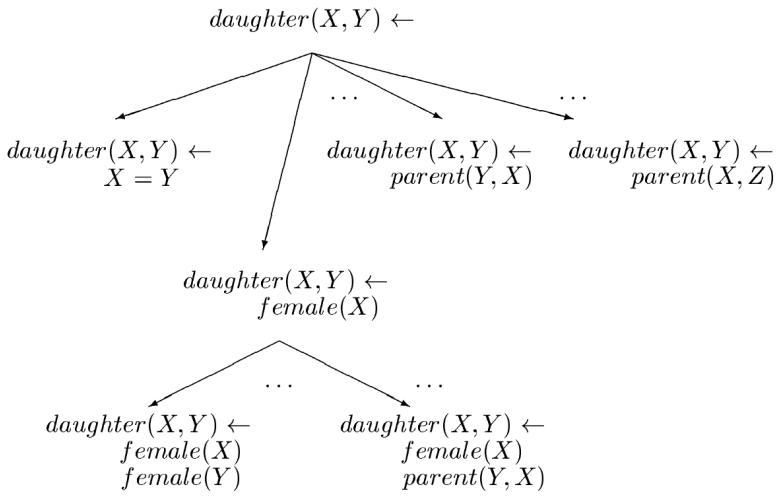


Figure 2.1: Part of the refinement graph for the family relations problem[3].

Algorithm 1: Generic top-down specialization ILP algorithm [3].

Input: $\mathcal{E}$: Training examples, $\mathcal{B}$: Background knowledge
Result: $\mathcal{H}$: Learned hypothesis
`// Initialize the current training set and hypothesis`
$\mathcal{E}_{cur} \leftarrow \mathcal{E}$;
$\mathcal{H} \leftarrow \emptyset$;
repeat Covering Loop
  `// Initialize clause with empty body`
  $c \leftarrow T$:- $\emptyset$;
  repeat Specialization Loop
    $c_{best} \leftarrow$ Best refinement of c; $c \leftarrow c_{best}$;
  until *Necessity Stopping Criterion is satisfied*;
  `// Add clause to the hypothesis`
  $\mathcal{H} \leftarrow \mathcal{H} \cup \{c\}$;
  `// Remove positive examples covered by the new hypothesis`
  $\mathcal{E}_{cur} \leftarrow \mathcal{E}_{cur} - covers(\mathcal{B} \cup \mathcal{H}, \mathcal{E}^{+}_{cur})$;
until *Sufficiency Stopping Criterion is satisfied*;

In domains with perfect data, the stopping criteria require that all positive examples are covered (completeness) and no negative examples are covered (consistency). However, in practice, many datasets have imperfect data. Such imperfection is usually of the following kinds, as described in Lavrac and Dzeroski [4]:

- Noise: random errors in the training examples and background knowledge.
- Insufficiently covered example space: too sparse training examples from which it is difficult to reliably detect correlations.
- Inexactness: inappropriate or insufficient description language which does not contain an exact description of the target concept.

- Missing facts in the training example.

In order to avoid the effects of imperfect data, ILP can use heuristics as stopping criteria which tolerate some level of incompleteness and inconsistency. The 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:

$$A(c) = P(e \in \mathcal{E}^+|c) = \frac{n^+(c)}{n^+(c) + n^-(c)} \tag{2.1}$$

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

2.2.3 ILP Complexity

As shown in Gottlob et al. [5], the complexity of ILP resides at the second level of polynomial hierarchy and it is Σ_2^P-complete. Σ_2^P means that it has complexity NP^{NP}, and that happens because there are two interlaced sources of NP complexity class:

- Choice problem, which is NP
- Checking problem, which is *co*-NP

The choice problem, in ILP, is the the problem of searching in the hypothesis space, while the checking problem is the problem of testing a given hypothesis. Since ILP has a great complexity, it is appropriate to employ various techniques with the objective of reducing runtime.

One popular strategy is the limitation of the number of literals allowed in a clause, which limits the number of levels, which is easy to implement and can drastically reduce the size of the hypothesis search space, nevertheless, it may also reduce the quality of the learned hypothesis. Another popular strategy is the data sampling. When well applied, this can be a very good alternative, however, with semantic data, it can be very tricky to make a good sample without significant loss of information. There are several different sampling techniques for semantic data, however, it is out of the scope of this thesis.

2.3 Association Rule Mining

Association Rule Mining, as described in Agrawal et al. [6], 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 $\mathcal{D} = \{t_1, t_2, \ldots, t_m\}$ and a set of items $\mathcal{I} = \{i_1, i_2, \ldots, i_n\}$. Each transaction $\mathcal{T}$ consists of a set of items which is subset of I, which is usually 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.

2.3.1 Measures

In this subsection we will discuss measures used for selecting interesting rules in the mining process.

As defined in Agrawal et al. [6], 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:

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

The confidence of a rule $X \Rightarrow Y$, which can may be interpreted as the probability of the the head given the body $P(Y|X)$, is defined as:

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

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 does 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 more interesting.

2.3.2 Anti-monotonicity of Support

An important characteristic of the support measure is the anti-monotonicity. It means that for any itemset $X \subseteq \mathcal{I}$ if we have another itemset $Y \subseteq \mathcal{I}$ such that $X \subseteq Y$, then the support of Y cannot be greater than the support of X. In other words, all subsets from a frequent itemset are also frequent, and all supersets from an infrequent item are also infrequent.

When searching for frequent itemsets, this characteristic plays a key role in pruning the search space, which grows exponentially with the number of items n. The set of possible itemsets is the power set over $\mathcal{I}$, which has size $2^n - 1$. As association rules are usually required to satisfy a specified minimum support, if we know that a given itemset does not satisfy it, we can automatically prune all its supersets.

2.3.3 Itemset Lattice

The problem of searching frequent itemsets (FI), i.e., itemsets that satisfy the specified minimum support, can be structured in a lattice. Firstly, an itemset is frequent if it has support greater or equal than an arbitrarily specified minimum support.

There are two special kinds of itemsets:

- Maximal frequent itemset (MFI): a frequent itemset is maximal if none of its immediate supersets are frequent.
- Closed frequent itemset (CFI): a frequent itemset is closed if all of its immediate supersets have lower support.

Each of these kinds of itemsets has an special property. By knowing all the maximal itemsets of a database, we also know all the frequent itemsets. Nevertheless, it is not possible to know the support of each of the frequent itemsets. By knowing all the closed itemsets of a database, it is possible to know the all the frequent itemsets as well as their supports.

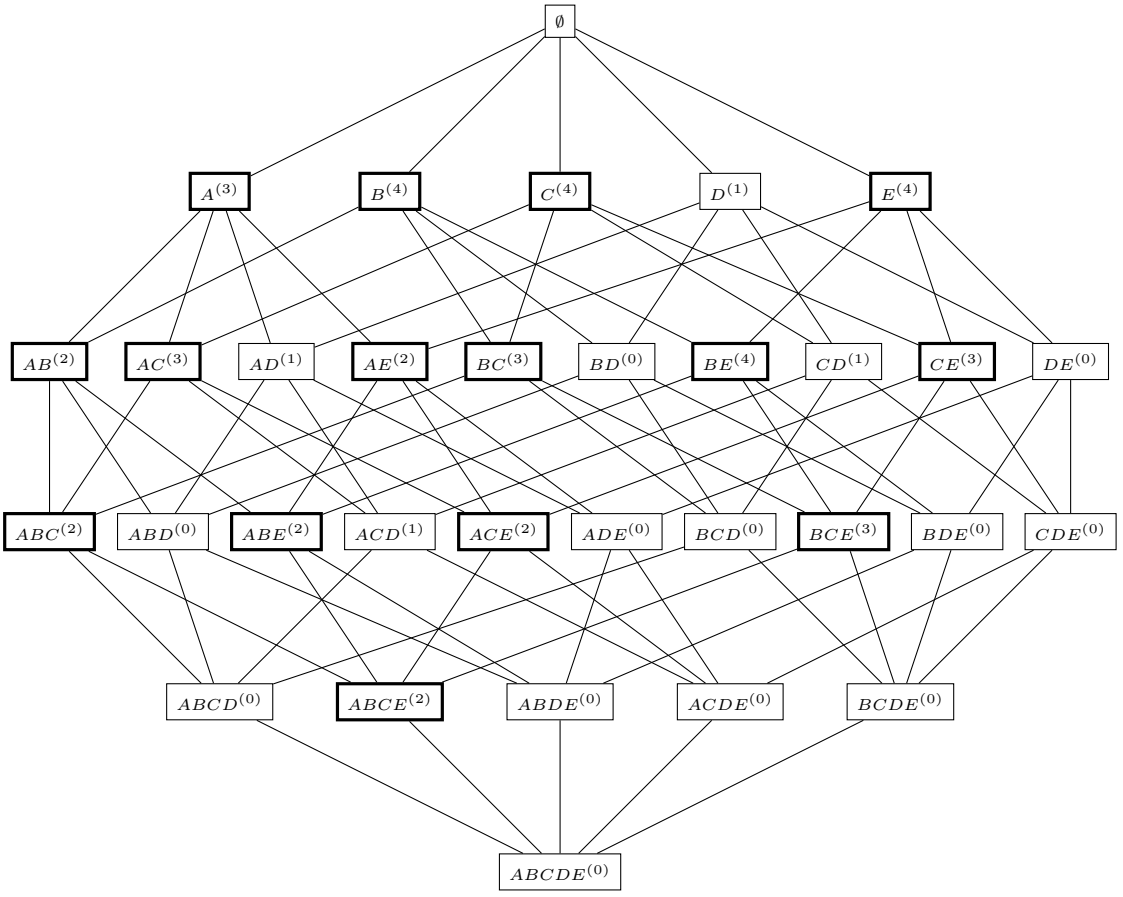
In the example of database $\mathcal{D}$ shown in Table 2.2 with the resulting itemset lattice shown in Figure 2.2, frequent itemsets are shown with thicker border. The minimum support threshold is set to 2/5, so the frequent itemsets are the ones with frequency greater than

or equal to 2. For this example, we would have as maximal itemset $\{ABCE\}$ and as closed itemsets $\{ABCE, BCE, BE\}$

TID	Items			
1	A	C	D	
2	B	C	E	
3	A	B	C	E
4	B	E		
5	A	B	C	E

Table 2.2: Example of transaction database $\mathcal{D}$ [7]

Figure 2.2: Itemset Lattice example [7].



2.3.4 Apriori Algorithm

The apriori algorithm, discussed in Pasquier et al. [7], is a method for searching frequent itemsets which explores the anti-monotonic property of the support measure. It computes the itemsets support iteratively, by ascending order of size. The process takes k iterations

where k is the size of the largest frequent itemset. In each iteration $i \leq k$, the database is scanned once and all the support of all itemsets of size i are computed.

In the first iteration, all itemsets of size 1 have their supports computed. On the $1 < i \leq k$ subsequent iterations, a set of candidates C_i is created by joining the frequent itemsets of size $i-1$ found in the previous iteration. Two itemsets of size i can only join if they share $i-1$ items, so their union can form an itemset of size $i+1$. Only after finding the set of candidate itemsets C_1, the database is scanned in order to determine the support of each of them. Algorithm 2 shows in more details how the apriori algorithm works.

Algorithm 2: Apriori frequent itemset discovery [7].

Input: $\mathcal{D}$: Database of transactions
Result: $\mathcal{H}$: Learned hypothesis
$L_1 \leftarrow$ Frequent itemsets of size 1 ;
$L \leftarrow L_1$;
$k \leftarrow 1$;
while $L_{k-1} \neq \emptyset$ **do**
  $C_k \leftarrow$ `aprioriGen(`L_{k-1}`)` ;
  forall $t \in \mathcal{D}$ **do**
    $C_t \leftarrow \{c | c \in C_k \wedge c \subseteq t\}$;
    forall $c \in C_t$ **do**
      $c.frequency \leftarrow c.frequency + 1$;
  $L_k \leftarrow \{c | c \in C_k \wedge c.frequency \geq minSupport\}$;
  $L \leftarrow L \cup L_k$;
  $k \leftarrow k + 1$;
return L ;

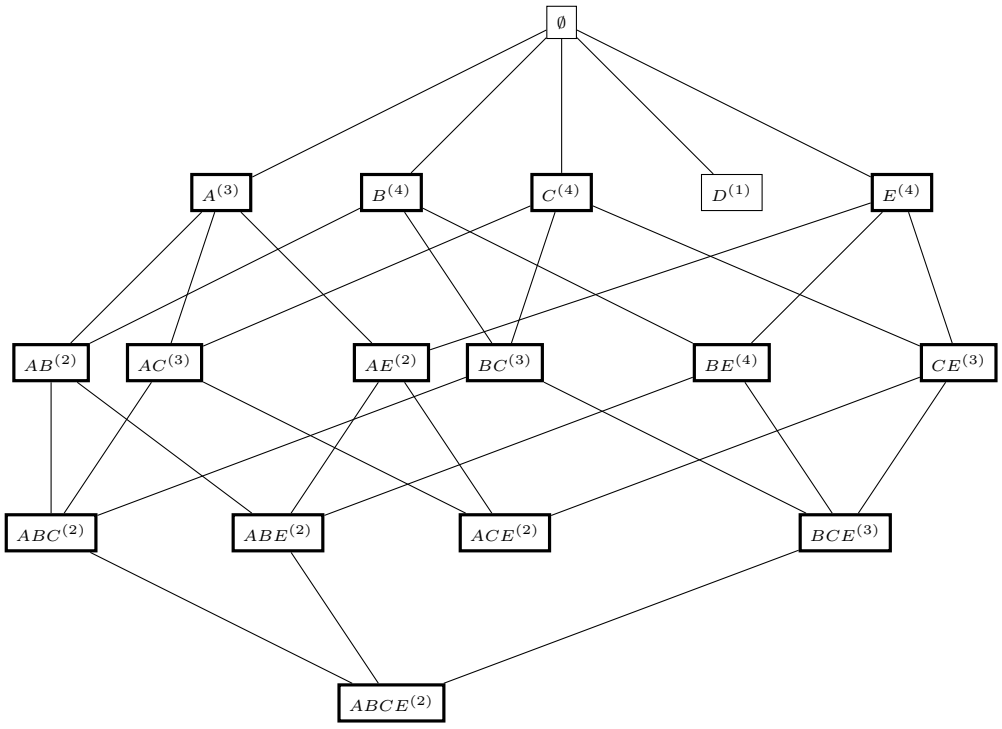
For the previous example of database from Table 2.2, the itemset lattice generated by the apriori algorithm is shown in Figure 2.3.

As we will show later in Section 3.5, the itemset lattice and the apriori algorithm inspired and served as base for the correlation lattice. Therefore, both structures look alike, and have similar building processes.

2.4 Discretization of Numerical Attributes

Discretization of constinuous features is a very important tool which is frequently used in statistics, data mining and machine learning. It is a pre-processing technique which is used for reducing the effects of observation errors. It is basically a form of quantization where the original data values in a given interval are replaced by a representative value of that interval.

Figure 2.3: Itemset Lattice generated by apriori algorithm for the example from Table 2.2



Discretization techniques can be divided in supervised and unsupervised methods. The first requires the objects to be labeled with classes, and the domain is discretized with the objetive of reducing the class entropies. The second one does not consider object classes, only object frequencies. Unsupervised methods normally require the number of buckets to be beforehand defined.

Various different supervised and unsupervised methods for the discretization of continuous features are discussed in Dougherty et al. [8]. To build the lattice, we don not 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 these in detail.

2.4.1 Equal Width

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

$$w = \frac{max - min}{k} \tag{2.6}$$

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.

2.4.2 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 m/k. Multiple objects 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 highest possible entropy, i.e., distribution is always uniform or as close to uniformity as possible.

2.5 Mining Optimized Rules for Numeric Attributes

In Brin et al. [9], a technique for efficiently learning association rules with interesting ranges for numerical attributes is presented. It focuses on finding numerical intervals which optimize a specific interestingness measure, given the confidence and support distribution of a rule along a numerical attribute.

Such rules have the form $(A_1 \in [l_1, u_1]) \wedge C_1 \rightarrow C_2$, where A_1 is a numerical attribute, l_1 and u_1 are the lower and upper boundaries of A_1, and C_1 and C_2 contain only instantiated conditions.

The authors propose algorithms for determining values for the boundaries l_1 and u_1 for the following cases:

- Optimized confidence: the rule confidence is maximized and support of the condition $(A_1 \in [l_1, u_1]) \wedge C_1$ is at least the specified minimum support.
- Optimized support: the support of the condition $(A_1 \in [l_1, u_1]) \wedge C_1$ is maximized and the rule confidence is at least the specified minimum confidence.
- Optimized gain: the rule gain is maximized and the confidence is at least the specified minimum confidence. Brin et al. [9] defines gain of a rule r as $gain(r) = supp(r) * (conf(r) - minConf)$, where $minConf$ is the confidence threshold.

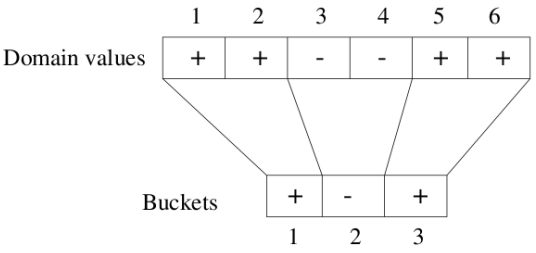


Figure 2.4: Example of buckets generated [9]

In Brin et al. [9], the two dimensional case with two different numerical attributes is also discussed, however, in this section we will focus on the single numerical attribute case.

Basically, it first takes the transactions with numerical attribute A_1 and discretizes its numerical domain of size n into $b < n$ buckets, in order to reduce the complexity of the algorithm. It uses a supervised bucketing algorithm, which uses as labels "+" if the confidence value is greater than the threshold (positive gain), and "−", if it is less (negative gain). This method does not compromise the optimality of the algorithm, as it collapses values with similar labels together, generating b buckets, each with its own support and confidence values. Figure 2.4 shows an example where a domain of size $n = 6$ is discretized into $b = 3$ buckets.

Algorithm 3 shows how the proposed one-dimensional algorithm works for gain optimization. Figure 2.5 illustrates that with an example with $b = 6$ buckets, with confidence values 10, −15, 20, −15, 20, and −15. $PSet$ is the optimized gain set containing $i - 1$ intervals, and $NSet$ contains the remaining intervals not stored in $PSet$. These sets are represented before and after the iterations $i = 1$ and $i = 2$. As shown in the figure, any optimized interval contained in $PSet$ has positive gain, and its neighboring intervals have negative gain.

It is important to highlight that this technique does not have the same purpose of the work presented in this thesis. While Brin et al. [9] focuses on finding the optimal interval with respect to a given measure, assuming that we already have support and confidence distribution along the numerical attribute, in this thesis, we focus on identifying rules which potentially have sub-interval with confidence gain in relation to its base rule, before we actually query the support and confidence distributions, which is very expensive, pruning rules that have no correlation to the numerical attribute to be investigated.

Algorithm 3: Algorithm for computing optimized gain set [9]

Input: **Input:** b: number of buckets, k: maximum number of optimum intervals
$PSet \leftarrow \emptyset$;
$NSet \leftarrow \{[1, b]\}$;
for $i \leftarrow 1$ *to* k **do**
  Let P_q be the interval in $PSet$ with the smallest value for $gain(min(P_q))$;
  Let N_q be the interval in $NSet$ with the smallest value for $gain(max(N_q))$;
  if $gain(min(P_q)) + gain(max(N_q)) < 0$ **then**
    Delete P_q from $PSet$;
    Split P_q into three sub-intervals (with $min(P_q)$ as the middle interval);
    Insert the first and third intervals to $PSet$ and second interval to $NSet$;
  else
    Delete N_q from $NSet$; Split N_q into three sub-intervals (with $max(N_q)$ as the middle interval);
    Insert the first and third intervals to $NSet$ and second interval to $PSet$;
return $PSet$;

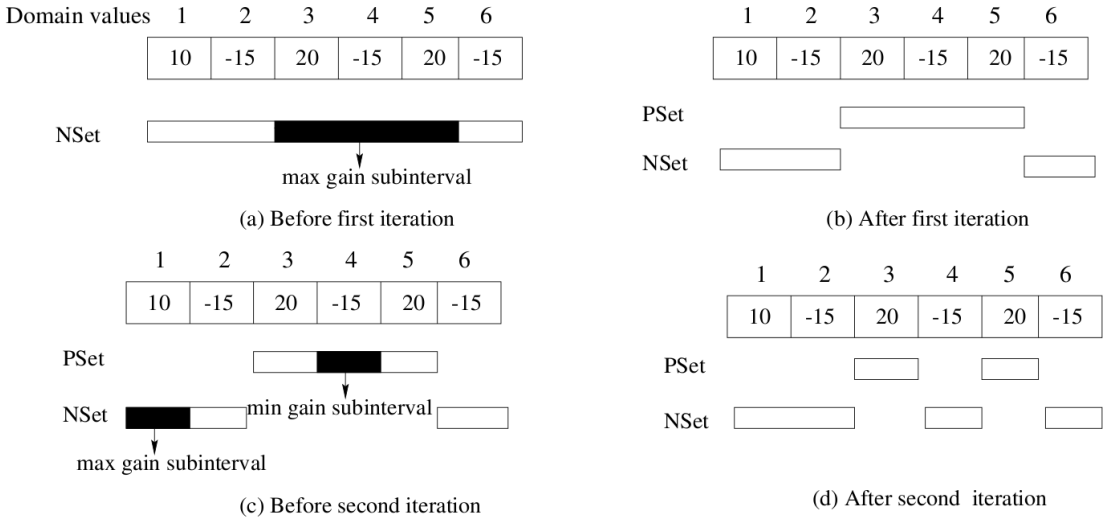


Figure 2.5: Execution trace of procedure shown in Algorithm 3. (a) Before first iteration, (b) after first iteration, (c) before second iteration, and (d) after second iteration[9].

2.6 Information-Theoretic Measures

Information-theoretic measures are widely used in various different learning processes. Its importance lies on its power to measure importance of attributes and relationships between attributes. An attribute is considered important in data mining if regularities are encountered in smaller populations obtained based on the values of such attribute, but not in the larger population. Such regularities are identifiable by lower entropy values, indicating that interesting attributes lead to entropy reduction. More specifically, the entropy reduction is the difference between the entropy of the decision attribute

and the conditional entropy of the decision attribute given a particular attribute, whose importance we want to investigate.

Let's say we have X as decision attribute and it divides the set of objects $\mathcal{U}$ into a group of disjoint subsets defined by the value $x \in \mathcal{X}$ of X, where $\mathcal{X}$ is the non-empty set of values for X. Also, let's define $I_X : \mathcal{U} \to \mathcal{X}$ is an information function that gives the value of attribute X for an object $t \in \mathcal{U}$. The set of objects $m(X = x) \subseteq \mathcal{U}$, whose value for X is x is defined as follows:

$$m(X = x) = m(x) = \{t \in \mathcal{U} | I_X(t) = x\} \tag{2.7}$$

With that, the probability distribution of X is defined by:

$$P(X = x) = P(x) = \frac{|m(x)|}{|\mathcal{U}|}, \quad x \in \mathcal{X} \tag{2.8}$$

Information theoretic measures are defined over probability distributions. They can measure the importance of an attribute, attribute association, dissimilarity, or similarity of populations. In the next subsections, we will discuss some of the most important measures.

2.6.1 Shannon's Entropy

The Shannon's entropy measure H is a non-negative function which may be interpreted as a measure of the information content or uncertainty about an attribute X. It is defined over a probability distribution of X:

$$\begin{aligned} H(P(X)) = H(X) &= E_{P(X)}[-logP(X)] \\ &= -\sum_{x \in \mathcal{X}} P(x) logP(x) \end{aligned} \tag{2.9}$$

The entropy of X is maximum if its probability distribution is uniform, and minimum if the whole population is concentrated on a specific value $x_c \in \mathcal{X}$, i.e., $P(x_c) = 1$ and $P(x) = 0, x \neq x_c$. The higher the entropy,the higher the uncertainty about X's attribute value, and an entropy of zero means complete certainty about the value of X since $P(x_c) = 1$.

2.6.2 Kullback-Leibler Divergence

The Kullback-Leibler divergence [10] $D(P||Q)$ between the population distributions $P(X)$ and $Q(X)$ measures the degree of deviation of P from another distribution Q:

$$\begin{aligned} D_{KL}(P||Q) &= E_{P(X)}\left[\frac{P(X)}{Q(X)}\right] \\ &= \sum_{x\in\mathcal{X}} P(x) log\left(\frac{P(x)}{Q(x)}\right) \end{aligned} \tag{2.10}$$

This divergence measure is a premetric but not a metric distance. It is non-negative and it becomes zero if $P(x) = Q(x)$, $\forall x \in \mathcal{X}$, but it does not satisfy the symmetry and the triangle inequality properties.

There are symmetrized divergence measures based on the Kullback-Leibler, e.g. the J-divergence 2.11 and the Jensen-Shannon divergence 2.12:

$$D_J(P||Q) = D_{KL}(P||Q) + D_{KL}(Q||P) \tag{2.11}$$

$$D_{JS}(P||Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M) \tag{2.12}$$

where M is the average of the two distributions, $M = \frac{1}{2}(P + Q)$.

The Jensen-Shannon divergence also has the advantage that the difference is always a finite value ($0 \leq D_{JS}(P||Q) \leq 1$, for the base-2 logarithm). Although, it still does not satisfy the triangle inequality property, its square root does and, therefore, is a metric. Further details about these divergence measures are presented in Briet and Harremos [11], Vinh et al. [12], and Guiaşu [13].

2.6.3 Mutual Information

The divergence measure can be used for computing the degree of independence between two attributes X and Y. This is done by simply measuring the divergence between the observed joined distribution $P(X, Y)$ and the independent distribution formed by the marginals $P(X)P(Y)$. We call such measure mutual information:

$$\begin{aligned} MI(X;Y) &= D(P(X,Y)||P(X)P(Y)) \\ &= E_{P(X,Y)}\left[log\frac{P(x,y)}{P(x)P(y)}\right] \\ &= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} P(x,y)log\frac{P(x,y)}{P(x)P(y)} \end{aligned} \tag{2.13}$$

Completely independent attributes have mutual information zero, and the greater the level of dependence, the greater the mutual information.

2.7 Semantic Web Applications

The Semantic Web is, according to its official website[1], a technology created by the World Wide Web Consortium (W3C) to enable the so-called "Web of data", enabling computers to do more useful work and develop systems that can support trusted interactions over the network.

It consists of a set of technologies and formats which aim to make web resources more readily accessible to automated processes. The Semantic Web Stack, in Figure 2.6[2], illustrates the semantic web architecture showing how its standards are organized and how it extends the classical hypertext web.

- Resource Description Framework (RDF): a framework for creating statements in the form of triples composed by subject, predicate and object. It uses URIs to name the predicates as well as the entities, which may be the subject or object in a triple. RDF can be encoded in a variety of formats, such as RDF/XML, N3, Turtle and N-Triples.
- RDF Schema (RDFS): a RDF vocabulary description language which provides basic elements for the description of ontologies (or RDF vocabularies) intended to structure RDF resources known as constructs (RDFS classes, associated properties, and utility properties).
- Web Ontology Language (OWL): a semantic markup language for publishing and sharing ontologies on the web. It is a vocabulary extension of RDF that allows an

[1] http://www.w3.org/standards/semanticweb/

[2] Semantic Web - XML2000, slide 10". W3C: http://www.w3.org/2000/Talks/1206-xml2k-tbl/slide10-0.html

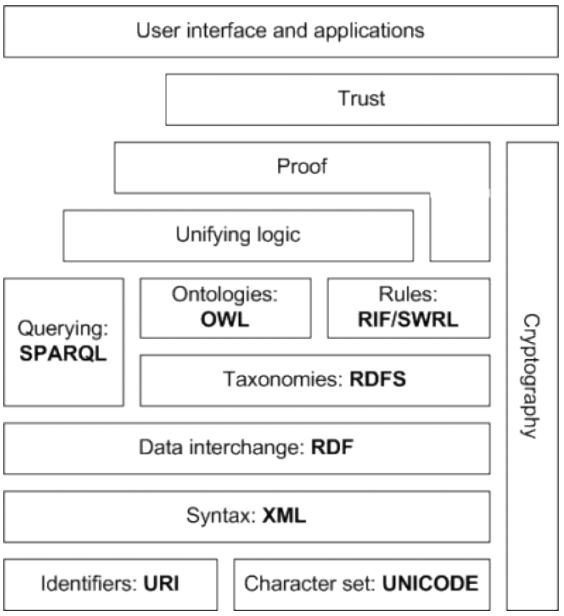


Figure 2.6: Semantic Web stack

ontology to be described with a set of axioms which places constraints on classes and the types of relationships permitted between them.

- SPARQL: an RDF query language for databases which can be used to express queries across diverse data sources. It consists of triple patterns, conjunctions, disjunctions and optional patterns.
- Rule Interchange Format (RIF): defines a standard for exchanging rules among rule systems. It focuses on exchange rather than defining a single one-fits-all rule language, different databases might have different characteristics with different necessities.
- Semantic Web Rule Language (SWRL): a proposal for rule language combining sublanguages of OWL with the Unary/Binary Datolog Rule Markup Language[3] (RuleML), which specifies different derivation rules via XML Schema.

2.8 Linked Open Data Applications

The Semantic Web does not need only access to reachable and manageable data in a standard format, but also relationships among data from different sources. Therewith, it is possible to create a huge collection of interrelated datasets in the Web, also referred as Linked Data.

[3] http://ruleml.org/

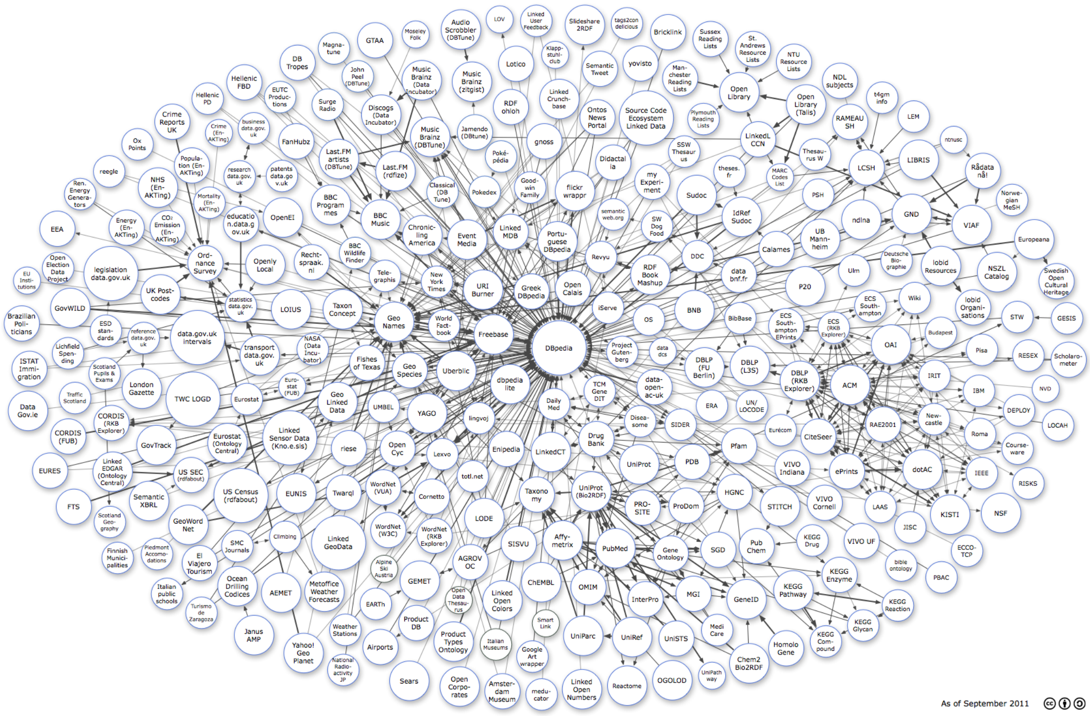


Figure 2.7: Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch. http://lod-cloud.net

Linked Data is a fundamental part of the Semantic Web essence, facilitating a large scale integration and reasoning on data on the Web. A typical example of a Linked Data collection is DBpedia [4]. It basically extracts information from Wikipedia and makes it available in RDF incorporating links to other datasets on the web, such as Geonames[5], YAGO[6], LinkedMDB[7] and USCensus[8]. With such linkage, it allows applications to exploit the extra information these datasets can provide, integrating facts from various different sources. Figure 2.7 shows the latest version of the whole Linked Open Data cloud diagram, and Figure 2.8 zooms to and highlights some of the datasets used in this thesis, also sho

[4] http://www.dbpedia.org/
[5] http://www.geonames.org/
[6] http://www.mpi-inf.mpg.de/yago-naga/yago/
[7] http://www.linkedmdb.org/
[8] http://www.linkedmdb.org/

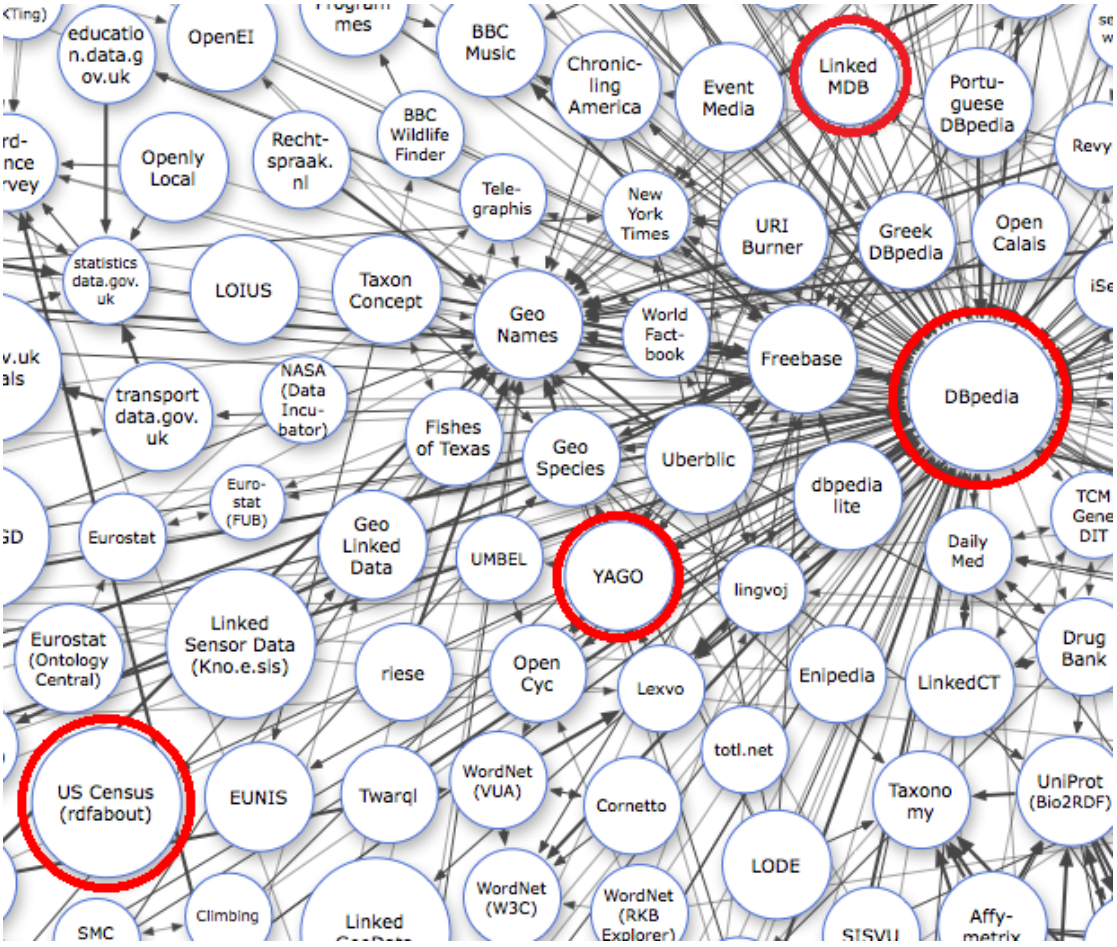


Figure 2.8: Zoom of Figure 2.7 highlighting datasets relevant to this thesis

Chapter 3

Learning Rules With Numerical and Categorical Attributes

In this chapter, we will discuss what kinds 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 into 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.

In this context, we define a rule with free numerical variable as *base-rule* and the same rule with the numerical variable restricted to a specific interval or value as a *refined-rule*. For instance, if we have two rules r_1 and r_2:

$$r_1 : employmentStatus(X, unemployed) \text{ :- } hasIncome(X, Y)$$
$$r_2 : employmentStatus(X, unemployed) \text{ :- } hasIncome(X, Y), Y > 100000$$

Then we say that r_1 is the base-rule of r_2 and r_2 is a refined-rule of r_1.

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 its base-rule would not be very significant as the base-rule already presents a high confidence value (at least higher than the specified threshold).

More precisely, if we have a confidence threshold TS_{conf}, and define confidence gain from a refined-rule r_2 in comparison to its 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 which satisfy the support threshold but do not satisfy the confidence threshold, or which satisfies a quite low confidence threshold. In such case, if we have a non-uniform confidence distribution along the buckets, it is possible that, for some specific intervals, its correspondent refined-rule satisfies both thresholds, therefore potentially having a significant confidence gain.

In Figure 3.1, we illustrate that with an example where positive examples and body support have completely different distributions and thus produce a very interesting confidence distribution. Such a distribution is obtained thanks to the divergent rule's positives (examples that satisfy the head and body) and body support (examples that support the body) distributions. This is shown in Figure 3.2, where we obtained these distributions by normalizing the frequency histograms from the rule's positive examples and body support.

Figure 3.1: Example of frequency histograms from body support and positives support (left), and resulting confidence distribution (right)

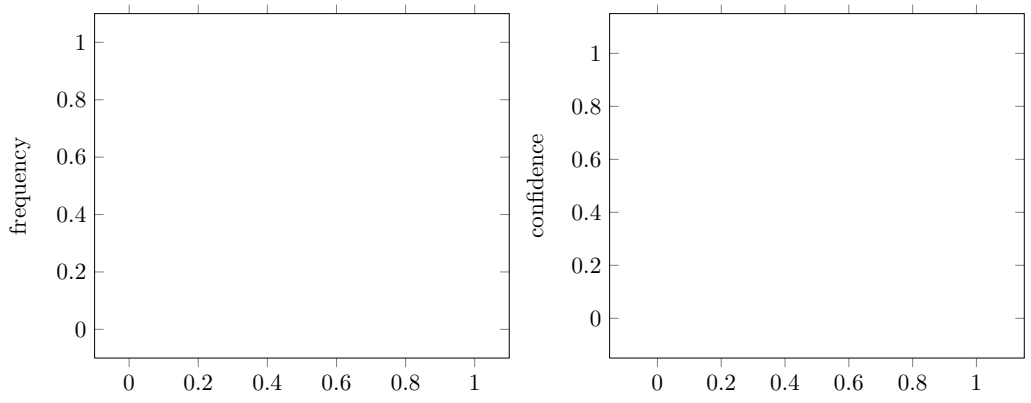
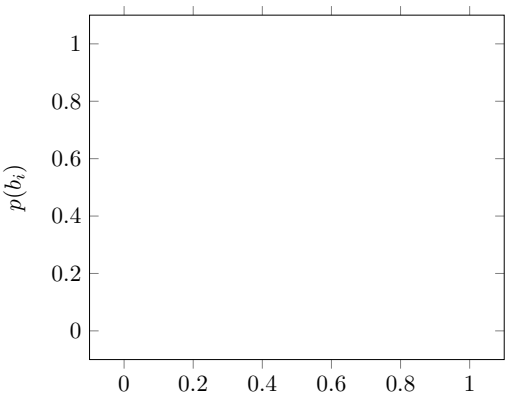


Figure 3.2: Support distribution of body and positives from figure 3.1



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 a root's non-numerical variable $X \in \mathcal{X}$,which will be used for joining with all the categorical properties.

A candidate categorical relation $r(X, Z)$, where X is the join variable and Z is within a categorical domain $\mathcal{Z}$ with a finite number of categories $\{z_1, z_2, \ldots, z_n\}$, such that:

$$\mathcal{X}_{z_i} = \{x | x \in \mathcal{X} \wedge \exists r(x, z_i)\}$$

Typically, $|\mathcal{X}| \gg n$ and $\mathcal{X}_{z_i} \subseteq \mathcal{X}$ with $|\mathcal{X}_{z_i}| < |\mathcal{X}|$

Table 3.1: Example data for categorical relation *hasSex*

hasIncome(john,20000)	hasSex(john,male)
hasIncome(mike,30000)	hasSex(mike,male)
hasIncome(anne,25000)	hasSex(anne,female)
hasIncome(mary,5000)	hasSex(mary,female)
hasIncome(lisa,10000)	hasSex(lisa,female)
hasIncome(paul,0)	hasSex(paul,male)

For Table 3.1, if we define $\mathcal{X} = \{x | \exists hasIncome(x, Y), Y \in \mathcal{Y}\}$, then we would have the following sub-populations of $\mathcal{X}$:

$$\begin{aligned}\mathcal{X}_{male} &= \{x | x \in \mathcal{X} \wedge \exists hasSex(x, male)\} \\ &= \{john, mike, paul\} \\ \mathcal{X}_{female} &= \{x | x \in \mathcal{X} \wedge \exists hasSex(x, female)\} \\ &= \{anne, mary, lisa\}\end{aligned}$$

The *rdf:type* property, for example, is also covered by this definition with the entity types being the categories.

A categorical property can be overlapping, i.e. with intersecting categories, or not. A non-overlapping categorical property $r(X, Z)$, such as $hasSex$, is defined as:

$$\mathcal{X}_{z_i} \cap \mathcal{X}_{z_j} = \emptyset, \quad \forall z_i, z_j \in \mathcal{Z}, i \neq j$$

This definition of categorical property might be too restrictive, therefore, in the next sections, we broaden this definition by applying different techniques.

3.2.1 Discretizing Numerical Properties into Categories

Numerical properties with very large or infinite domain can also be dealt as categorical, by simply applying a bucketing function that maps its numerical domain into a finite set of k buckets. For example, we can use a discretization function b to turn a numerical property $r(X, Z)$, with $Z \in \mathbb{R}$, into a non-overlapping categorical property:

$$b : \mathbb{R} \rightarrow \mathcal{B}, \text{ where } \mathcal{B} = \{b_1, b_2, \ldots, b_k\}$$

$$r(X, b(Z)) \equiv r'(X, B), \quad B \in \mathcal{B}$$

This can be easily done with any discretization method, such as equal width or equal frequencies, which were presented on Section 2.4.

3.2.2 Combining Categorical Property with Linking Relation

We can also broaden this definition by composing a categorical property with linking relations. Therewith, it is possible to use categorical relations that do not directly join with root's join variable X

If a functional relation r_1 and a categorical r_2:

$r_1(X, W) \equiv f_1 : \mathcal{X} \to \mathcal{W}$
$r_2(W, Z), \quad Z \in \mathcal{Z} = \{z_1, z_2, \ldots, z_k\}$ is a categorical domain

Then $r'(X, Z) \equiv r_1(X, W), r_2(W, Z) \equiv r_2(f_1(X), Z)$ is a composed categorical relation.

With that, different knowledge bases can be interconnected by simply applying this definition with *owl*:*sameAs* as linking relation. This allows to broaden the set of categorical properties to be analyzed in the correlation lattice to those contained in interlinked datasets.

Table 3.2: Example of data linked to example from Table 3.1

owl:sameAs(john,johnSmith)	hasProfession(johnSmith,teacher)
owl:sameAs(mike,mikeBell)	hasProfession(mikeBell,actor)
owl:sameAs(anne,anneSmith)	hasProfession(anneSmith,doctor)
owl:sameAs(lisa,lisaBell)	hasProfession(lisaBell,teacher)

Table 3.2 illustrates that, as it contains data about profession, not existent in Table 3.1. By combining *owl*:*sameAs* with $hasProfession$, we can also categorize the data from the first example into professions as follows:

$$\begin{aligned}
\mathcal{X}_{teacher} &= \{x | x \in \mathcal{X} \wedge \exists(owl : sameAs(x, W) \wedge hasProfession(W, teacher)\} \\
&= \{john, lisa\} \\
\mathcal{X}_{doctor} &= \{x | x \in \mathcal{X} \wedge \exists(owl : sameAs(x, W) \wedge hasProfession(W, actor)\} \\
&= \{anne\} \\
\mathcal{X}_{actor} &= \{x | x \in \mathcal{X} \wedge \exists(owl : sameAs(x, W) \wedge hasProfession(W, doctor)\} \\
&= \{mike\}
\end{aligned}$$

3.2.3 Absence or Presence of a Property as a Category

Another possibility is to define categories for the presence or absence of supporting examples for properties. With this approach, one can also include non-categorical properties into the correlation lattice and have insight about how the presence or absence of such property affects the distribution of a 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:

Table 3.3: Example of absence and presence of property as categories

hasIncome(john,20000)	isParentOf(john,mary)
hasIncome(mike,30000)	isParentOf(mike,paul)
hasIncome(anne,25000)	isParentOf(anne,paul)
hasIncome(mary,5000)	isParentOf(lisa,mary)
hasIncome(lisa,10000)	
hasIncome(paul,0)	

Then we could split the root node set $\mathcal{X}$ into two categories $\mathcal{X}_{parent} \subseteq \mathcal{X}$ and its complement $\mathcal{X}^c_{parent} \subseteq \mathcal{X}$:

$$\begin{aligned}\mathcal{X}_{parent} &= \{x | x \in \mathcal{X} \wedge \exists isParentOf(x, Z)\} \\ &= \{john, mike, anne, lisa\} \\ \mathcal{X}^c_{parent} &= \{x | x \in \mathcal{X} \wedge x \notin \mathcal{X}_{parent}\} \\ &= \{mary, paul\}\end{aligned}$$

In this thesis we are under an open world assumption, meaning that by the absence of a given fact, we do not assume that its negation is true. Therefore, we do not consider negated literals in the hypothesis, we ignore the absence and just include the presence of properties in the lattice.

3.2.4 Notation Used

As we will see in the next sections, in the correlation lattice all the relations are joined by the variable of a root's join argument X. So we will denote the presence of a property $a(X, Z)$ category as simply a, where the variable Z is free and not set to any constant. If we have a categorical property $a(X, z_i)$ with k different categories $z_i \in \mathcal{Z} = \{z_1, z_2, \ldots, z_k\}$, we denote each of the $a(X, z_i)$ as a_i.

So, for example, if we set the root relation $r = hasIncome(X, Y)$ and categorical relations:

$$a(X, Z) = isMarriedTo(X, Z)$$
$$b(X, W) = hasSex(X, W), \quad W \in \mathcal{W} = \{male, female\}$$
$$c(X, V) = bornInMonth(X, V), \quad V \in \mathcal{V} = \{january, february, \ldots, december\}$$

With such notation, we can write large clauses in a much more concise way, such as in the examples below:

$$rab_1c_3 \equiv hasIncome(X, Y), isMarriedTo(X, Z), hasSex(X, male), bornInMonth(march)$$
$$rab_2c_{11} \equiv hasIncome(X, Y), isMarriedTo(X, Z), hasSex(X, female), bornInMonth(november)$$

3.3 Interestingness Measure

As discussed before, we are interested in rules that have divergent positives and body support distributions, or in other words, adding the head to the body clause should result in a different distribution along the target numerical attribute Y. Therefore, we define the interestingness of adding a literal l to a clause c as the dissimilarity of $c \wedge l$ and c support distribution along Y.

It is possible to use any state-of-the-art dissimilarity measure, such as the ones presented in Section 2.6. Kullback-Leibler or its metrified version Jensen-Shannon are good options, and can be directly applied on normalized frequency histograms. However, using a divergence measure alone as interestingness measure is maybe cause some problems.

Firstly, it is important to take into account the sampling error. The lower the support of a clause, the higher the expected fluctuations of the observed distribution in relation to the sampling distribution. That means that if we draw two distributions from the same sampling distribution, one with more examples than the other, and measure the divergence between each of them and the sampling distribution, the smaller sample is expected to have a higher divergence.

Because of that, divergence measure alone tends to rank clauses with smaller support as more interesting. In order to compensate such effect, we can combine a divergence measure with the support, by simply multiplying their values. Moreover, including support into our interestingness measure is important because, after all, we are still interested in rules with high support.

3.4 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 lattice itself, which belongs to the preprocessing step but will be discussed in the next section.

3.4.1 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 basic 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**)*

Additionally, we have other two patterns that are a combination of ones mentioned before:

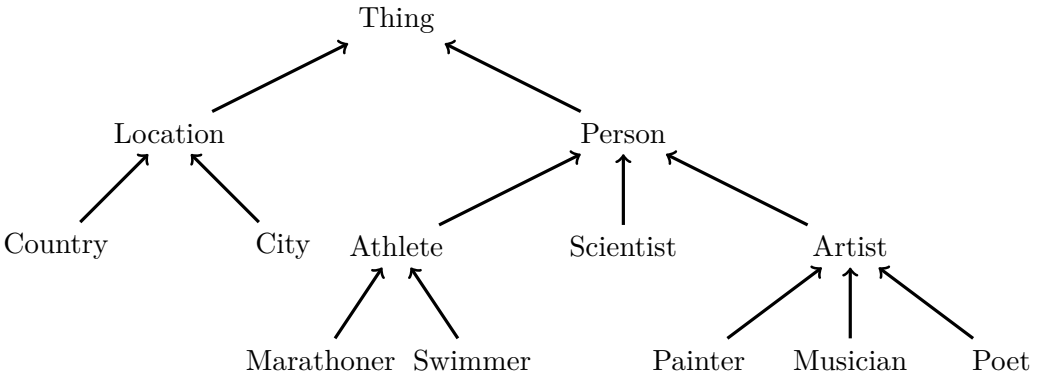
- Argument 1 on 1 and Argument 2 on 2: e.g. *livesIn(**X**,**Y**),wasBornIn(**X**,**Y**)*
- Argument 1 on 2 and Argument 2 on 1: e.g. *hasPlayers(**X**,**Y**),supportsTeam(**Y**,**X**)*

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

3.4.1.1 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 3.3: Type hierarchy example



Assuming that the knowledge base has its type hierarchy described with the relation *rdfs:subClassOf*, such as the example shown in Figure 3.3, where each arrow means that the tail is a subclass of the head. Also, we assume that every property has both argument types declared with *rdfs:domain* and *rdfs:range*. With such information, it is a straightforward task to prune out the pairs of joining relations which are not allowed by the type hierarchy.

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, i.e., one is a subclass of 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 4:

Algorithm 4: 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 isSubClassOf(type_i,type_j) or isSubClassOf(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 is worth to that beforehand, and that is what will be explained in the next section.

3.4.1.2 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 do not need to test such a hypothesis in the core ILP algorithm.

Algorithm 5: 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 }";
    case (11, 22)
        query ← "select count distinct ?x ?y where { ?x <r_i> ?y . ?x <r_j> ?y }" ;
    case (12, 21)
        query ← "select count distinct ?x ?y where { ?x <r_i> ?y . ?y <r_j> ?x }" ;
joinSupport ← executeQuery(query);
if joinSupport ≥ supportThreshold then
    return true;
else
    return false;
```

The preprocessing is done by Algorithm 6, which applies 4 and 5 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 6: 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**
- $L_{1,1}(r_i) \leftarrow r_i$;
- $L_{2,2}(r_i) \leftarrow r_i$;
- $L_{11,22}(r_i) \leftarrow r_i$;
- $L_{1,2}(r_i) \leftarrow \emptyset$;
- $L_{2,1}(r_i) \leftarrow \emptyset$;
- $L_{12,21}(r_i) \leftarrow \emptyset$;

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

foreach $r_i \in r$ **do**
- **foreach** $r_j \in (L_{1,1}(r_i) \cap L_{2,2}(r_i))$ **do**
  - **if** $checkSupport(r_i, r_j, 11, 22)$ **then**
    - $L_{11,22}(r_i) \leftarrow L_{11,22}(r_i) \cup r_j$;
- **foreach** $r_j \in (L_{1,2}(r_i) \cap L_{2,1}(r_i))$ **do**
  - **if** $checkSupport(r_i, r_j, 12, 21)$ **then**
    - $L_{12,21}(r_i) \leftarrow L_{12,21}(r_i) \cup r_j$;

3.4.2

3.5 Correlation Lattice

The idea is to build, during the preprocessing step, a graph inspired in the itemset lattice that represents the correlations between various categories and a given numerical attribute. We call such graph a correlation lattice. Comparing to an itemset lattice, in a correlation lattice we 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 extracted from the U.S. Census[1] data, with the $hasIncome(X, Y)$ relation. Also, there are have two categorical relations, one strongly correlated to income, e.g. $hasEducation$, and one uncorrelated (or very weakly correlated), e.g. $quarterOfBirth$.

For the relation $quarterOfBirth(X, Z)$, we have the 4 year quarters as possible constants for Z:

Q1: January, February, March
Q2: April, May, June
Q3: July, August, September
Q4: October, November, December

For the relation $hasEducation(X, W)$, we have 17 different levels as possible constants for W:

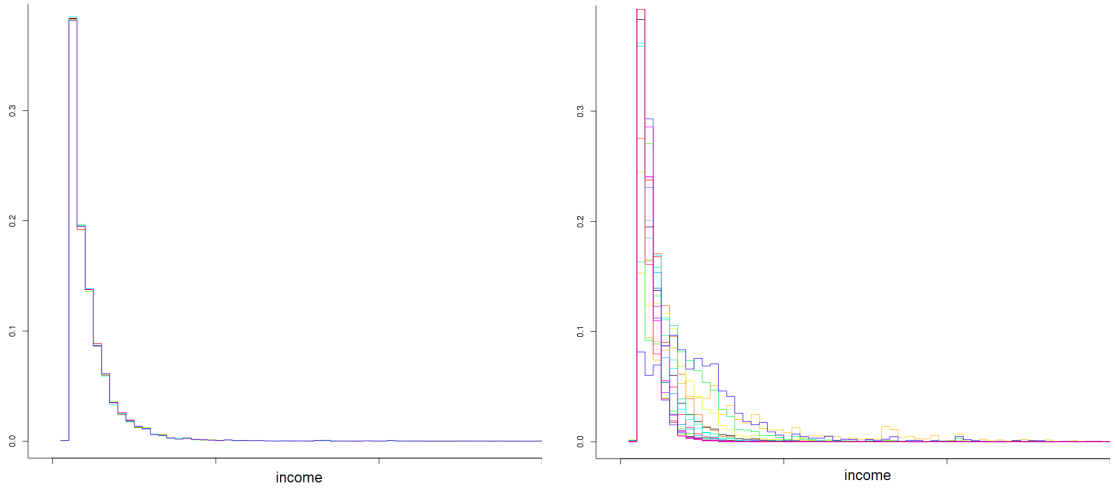
N/A (less than 3 years old)	High school graduate
No school completed	Some college, less than 1 year
Nursery school to grade 4	One or more years of college, no degree
Grade 5 or grade 6	Associate's degree
Grade 7 or grade 8	Bachelor's degree
Grade 9	Master's degree
Grade 10	Professional school degree
Grade 11	Doctorate degree
Grade 12 no diploma	

It's expected that the income distribution will be roughly the same for people born in any four year quarters, whereas for different education levels, e.g. Nursery school and Master's Degree, their income distribution are expected to be different from each other them and different from the overall income distribution.

Figure 3.4 shows the discrete probability distributions of income from each category's sub-population. On the left, we have the four income distributions of people born in Q1, Q2, Q3 and Q4, and on the right, the 17 different income distributions of people having each education level. As it is easily noticeable, for all the 4 year quarters, the income distribution is pretty much the same, while the education level, which has great influence on the income, the distributions are very different.

[1] http://www.rdfabout.com/demo/census/

Figure 3.4: Discrete probability distributions of income by year quarter (left), and by education level (right)



Based on this idea, we basically check how different categorical relations affect a numerical distribution by using distribution divergence measures. This 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.

3.5.1 Building the Lattice

We first start with the chosen numerical property, for example $hasIncome(X, Y)$, which should have at least two arguments, one being a joining variable X and the other the numerical attribute variable Y. Firstly, we query the examples distribution along the numerical attribute Y, arbitrarily choose a number of buckets k and discretize its Y's domain into buckets $b_1, b_2, \ldots, b_k$ using any unsupervised discretization method.

We then build a frequency histogram of the root node with the its overall population. We define the histogram of a node n with numerical attribute variable Y as a k-dimensional vector $h(n)$ where:

$$h(n) =< h_1(n), h_2(n), \ldots, h_k(n) > \tag{3.3}$$

where $h_i(n) = supp(n|Y \in b_i)$, and therefore:

$$|h(n)|_1 = \sum_{i=1}^{k} h_i(n) = supp(n) \tag{3.4}$$

Every node in the lattice is composed by conjunctive clause of non-negated literals and contains a frequency histogram of its population. For the sake of comparison, all histograms are built on the same buckets defined in the root node.

Subsequently, for the chosen set of categorical relations, we do the same with each sub-population of examples from each category obtained by joining the root with its corresponding categorical literal. For every category a node in the lattice is created with the associated frequency histogram. In addition, the edges of the lattice are created by setting the root as parent node, and adding the new nodes as children from the root.

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

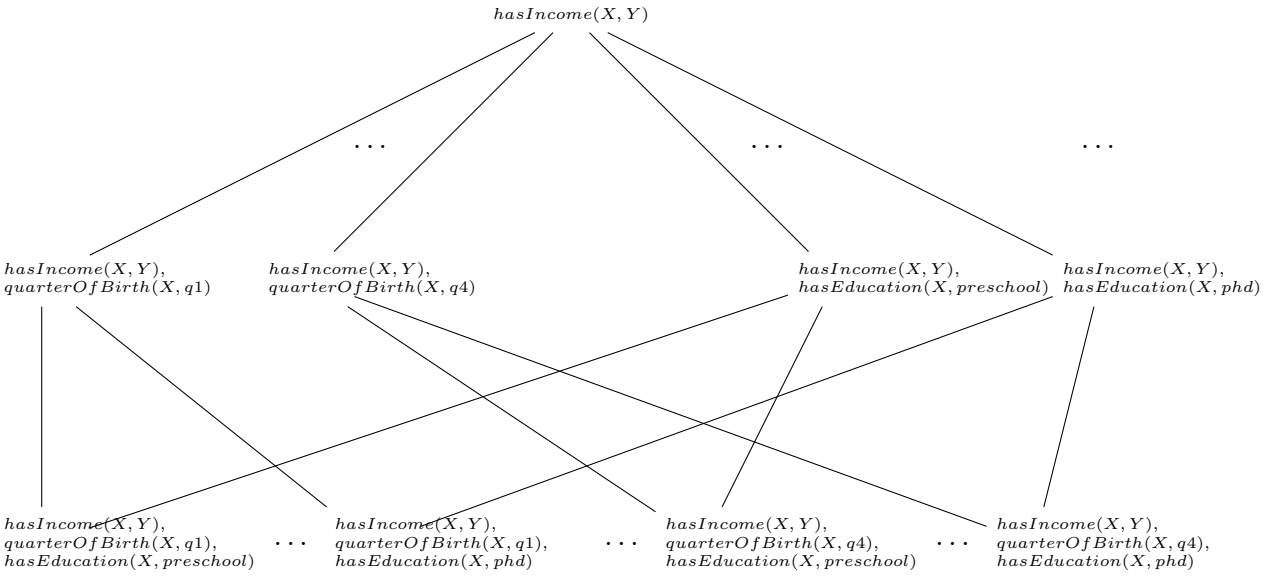
$$
\begin{array}{c}
hasIncome(X,Y), quarterOfBirth(X,\text{“Q1”}), hasEducation(X,\text{“Preschool”}) \\
hasIncome(X,Y), quarterOfBirth(X,\text{“Q1”}), hasEducation(X,\text{“Kindergarten”}) \\
\vdots \\
hasIncome(X,Y), quarterOfBirth(X,\text{“Q1”}), hasEducation(X,\text{“Doctorate Degree”}) \\
\\
\vdots \\
\\
hasIncome(X,Y), quarterOfBirth(X,\text{“Q4”}), hasEducation(X,\text{“Preschool”}) \\
hasIncome(X,Y), quarterOfBirth(X,\text{“Q4”}), hasEducation(X,\text{“Kindergarten”}) \\
\vdots \\
hasIncome(X,Y), quarterOfBirth(X,\text{“Q4”}), hasEducation(X,\text{“Doctorate Degree”})
\end{array}
$$

This process works just like in an itemset lattice until either a predetermined maximum level is reached or until all the possible combinations are exhausted. Figure 3.6 shows how a correlation lattice looks like.

3.5.2 Safe 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 the lattice size to a feasible level, later, in Section 3.5.4, we will also discuss possible heuristic pruning techniques.

Figure 3.5: Combination of relations *hasEducation* and *bornInMonth*



3.5.2.1 Support

As described in [2], 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. Hence, support is a monotonically decreasing measure so we can safely prune a node that does not reach the minimum support threshold. Therefore, we can use the apriori-style pruning, just like in the core-ILP and in the association rule mining.

In this thesis, we define support as the absolute frequency of supporting facts. So the support of a node n is defined as:

$$supp(n) = |x|x \in n|$$

3.5.2.2 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.

Figure 3.6: Correlation Lattice example

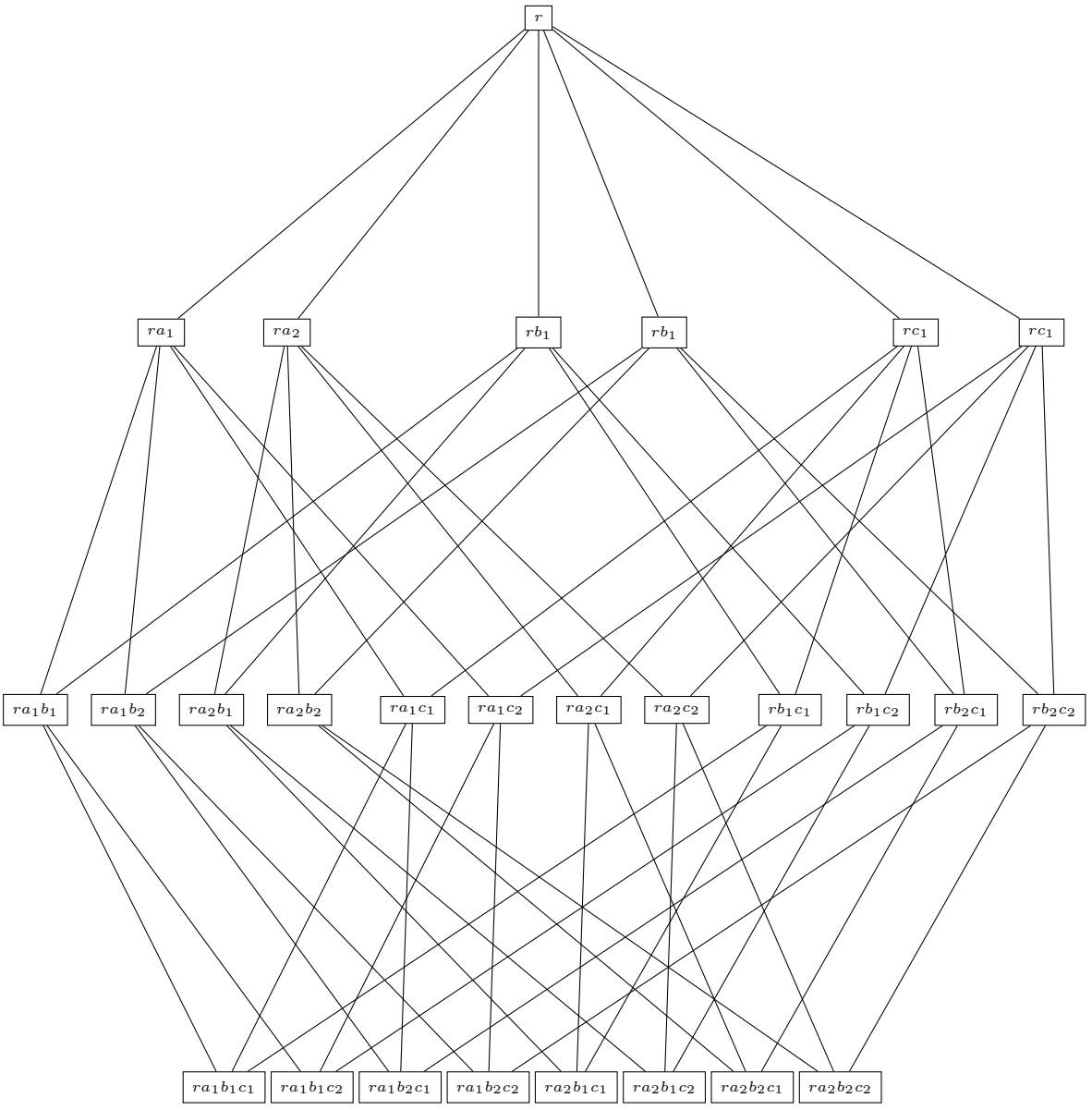
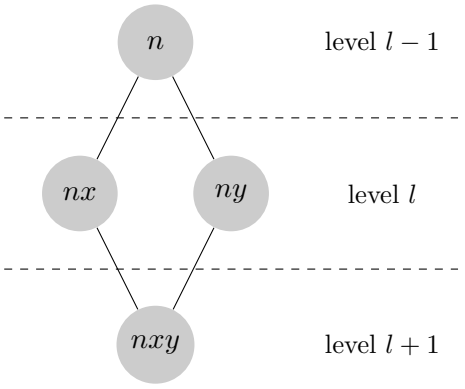


Figure 3.7: Node join example for independence test



Let's say we have the following join case:

In the example shown in Figure 3.7, x and y are literals and n is a node with clause c_n, such that $x \neq y$ and $x, y \notin c_n$. The nodes nx and ny are formed by adding x and y to n, so $c_{nx} = \{c_n, x\}$ and $c_{ny} = \{c_n, y\}$. With the histogram from these three nodes, we can calculate the conditional probability $p_i(x|n)$ and $p_i(y|n)$, then calculate $\hat{h_i}(nxy)$, which is the estimation of $h_i(nxy)$ assuming that x and y are independent given common parent n.

$$p_i(x|ny) = p_i(x|n) \tag{3.5}$$

$$= \frac{h_i(nx)}{h_i(n)} \tag{3.6}$$

$$p_i(y|nx) = p_i(y|p) \tag{3.7}$$

$$= \frac{h_i(ny)}{h_i(n)} \tag{3.8}$$

$$\tag{3.9}$$

$$\hat{h_i}(nxy) = p_i(x|ny)p_i(y|n) * h_i(n) \tag{3.10}$$

$$= p_i(x|p)h_i(yp) \tag{3.11}$$

$$\hat{h_i}(nxy) = p_i(y|nx)p_i(x|n) * h_i(n) \tag{3.12}$$

$$= p_i(y|n)h_i(xn) \tag{3.13}$$

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

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

$$df = k - 1$$

We calculate the critical value χ^2:

$$\chi^2 = \sum_{i=1}^{k} \frac{(h_i - \hat{h_i})^2}{\hat{h_i}} \tag{3.14}$$

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 3.13 as the following rules being equivalents, with similar confidence distributions:

$$x\text{:-}p, y \equiv x\text{:-}p \quad \text{and} \quad y\text{:-}p, x \equiv y\text{:-}p$$

Therefore, we know that if we have x fixed as the head of clauses in the core ILP, and we currently have x:-p joining the node px with py to obtain the rule x:-p, y does not add any valuable information. The same applies for having y, and obtaining y:-p, x from y:-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 3.7.

This idea is very similar to the idea of the lift measure, presented in Section 2.3, of detecting dependency. However, as we also take into account the support distribution, and we use a more sophisticated technique that compares the observed and expected distribution assuming independence.

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.

3.5.3 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 the 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 [10]:

$$D_{KL}(P||Q) = \sum_i \ln\left(\frac{P(i)}{Q(i)}\right) * P(i) \tag{3.15}$$

- Chi-squared (χ^2):

$$D_{\chi^2}(P||Q) = \sum_i \frac{(P(i) - Q(i))^2}{P(i)} \tag{3.16}$$

- Jensen-Shannon [11]:

$$D_{JS}(P||Q) = \frac{1}{2}D_{KL}(P||M) + \frac{1}{2}D_{KL}(Q||M), \tag{3.17}$$

Where P and Q are discrete probability 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 [11], 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 not be uniform and therefore potentially be interesting for searching for refined-rules with numerical intervals.

3.5.4 Heuristics

As seen before, the number of nodes in a correlation lattice grows 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:

$$\sum_{i=1}^{l} \binom{nm}{i} \tag{3.18}$$

if $l = nm$:

$$\sum_{i=1}^{mn} \binom{nm}{i} = 2^{nm} \tag{3.19}$$

Pruning by support is usually not sufficient to make an exhaustive search 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 non-overlapping categorical relations $a(X, Z)$ with constants z_1 and z_2 and $b(X, W)$ with constants w_1 and w_2. For simplicity, let's assume that the numerical variable Y is divided into two buckets and root r has an uniform distribution frequency histogram $h(r) =< 2, 2 >$ which results in the probability distribution $< 0.5, 0.5 >$.

It is possible join r with a and b obtaining the following uniform frequency histograms:

$$h(ra_1) =< 1, 1 >$$
$$h(ra_2) =< 1, 1 >$$
$$h(rb_1) =< 1, 1 >$$
$$h(rb_2) =< 1, 1 >$$

Although all of the nodes have the exact same probability distribution as r, which results in a divergence measure of zero, they could not be safely pruned, as it is possible to combine r a b with totally different distributions that maximize the divergence measure:

$$h(ra_1b_1) =< 1, 0 >$$
$$h(ra_1b_2) =< 0, 1 >$$
$$h(ra_2b_1) =< 0, 1 >$$
$$h(ra_2b_2) =< 1, 0 >$$

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. That happens due to the sampling error, which says that smaller samples are more likely to have a distribution more different to the original distribution from which it was drawn. Consequently, the exclusive use of 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.

As seen in Section 3.5.2.2, checking for conditional independence of categories is very insightful and can detect equivalent rules like in Figure 3.7 where we know that x:-$p, y \equiv x$:-p and y:-$p, x \equiv y$:-p. Nevertheless, we cannot prune the node pxy from the

lattice as x and y might not be independent given pz, and for instance, a rule x:-p, y, z might not be equivalent to x:-p, y.

3.6 Bucketing the 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 whole correlation lattice should 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.

Since we do not consider any labels for the examples, we use one of the unsupervised discretization methods presented in Section 2.4: equal frequencies or equal width. We choose the method according to the domain characteristics and define the bucket boundaries based on the overall population distribution.

3.7 Incorporating Correlation Lattice into the Core ILP Algorithm

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

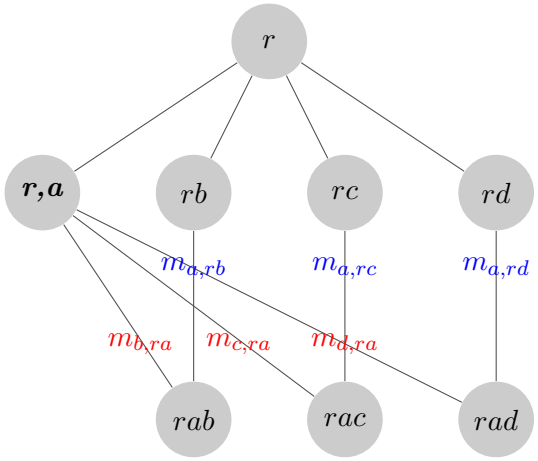
3.7.1 Querying the Correlation Lattice

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:-r, r is the root of a correlation lattice and 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 3.8: Interestingness of adding a literal to the body of a clause



As shown in Figure 3.8, if we can add b, c or d to ra, we are interested about the measures $m_{a,rb}$, $m_{a,rc}$ and $m_{a,rd}$ (shown in blue), instead of $m_{b,ra}$, m_c, ra and $m_{d,ra}$ (shown in red). Nevertheless, searching for values of $m_{a,rb}$, $m_{a,rc}$ 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 suggestions map when lattice is built. 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 and associated measure $m_{b,ra}$. This is shown more clearly in Algorithm 7.

In the example of Figure 3.8, 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 3.6 to better illustrate it:

Algorithm 7: Suggestion map build when joining nodes during lattice build

Input: **Input:** nx, ny: Joining nodes, n: Common parent node
$\hat{h}(nxy) \leftarrow h(nx)h(ny)/h(n)$;
$h(nxy) \leftarrow$`queryFrequencyHistogram(`nxy`)`;
$\chi^2 \leftarrow \sum_{i=1}^{k}(h_i(nxy) - \hat{h}_i(nxy))^2/(\hat{h}_i(nxy))$;
$dof \leftarrow k - 1$;
$pValue =$`criticalValue(`χ^2, dof`)`;
if $pValue \geq minPValue$ **then**
  $m_{x,ny} \leftarrow$`interestingness(`$h(nxy), h(ny)$`)`;
  $m_{y,nx} \leftarrow$`interestingness(`$h(nxy), h(nx)$`)`;
  $nx.suggestionsMap(x) \leftarrow (y, m_{x,ny})$;
  $ny.suggestionsMap(y) \leftarrow (x, m_{y,nx})$;

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 the search for numerical intervals as a possible refinement in the top-down ILP refinement step. Such a refinement would be possible only if the body contains a literal which is the root of a correlation lattice and whose numerical attribute variable is free, as shown in Algorithm 8. If so, then we check whether the clause is interesting by calling the function `isClauseInteresting`, which is shown in Algorithm 10.

Algorithm 8: Extension to the 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, *interestingnessThreshold*: Minimum interestingness value allowed
$r_{new} \leftarrow lit_{new}.relation$;
if $arg_{new} = 1$ ***and*** r_{new} *is numerical* ***and*** $\exists\ lattice(r_{new})$ **then**
  $root \leftarrow lattice(r_{new}).root$;
  if `isClauseInteresting(`*clause,root,interestingnessThreshold*`)` **then**
    `checkNumericalRanges(`*clause*,lit_{new},*node*`)`;

Once we find out that that the clause is interesting, we can perform the numerical interval search refinement shown in Algorithm 8. It basically queries the support and confidence distribution of the base-rule along the numerical attribute and searches for

Algorithm 9: `checkNumericalRanges`

Input:
Output:
$v \leftarrow$ Numerical variable from lit_{new};
$acc \leftarrow$ `queryConfidenceDistribution(`$clause$,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 ILP's refinement graph;
$suggestions \leftarrow node.getSuggestions(clause.head)$;
foreach $lit_i \in suggestions$ ***and*** $i \leq k$ **do**
    $node_i \leftarrow node$.`search(`lit_i`)`;
    `checkNumericalRanges(`$clause \cup lit_i$,lit_{new},$node_i$`)`;

Algorithm 10: `isClauseInteresting`

Input: $clause$: Clause with numerical literal, $root$: Correlation lattice root node, $interestingnessThreshold$: Minimum interestingness value allowed $bodyNode \leftarrow root$;
foreach $lit_i \in clause.body$ **do**
    $bodyNode \leftarrow bodyNode$.`search(`$lit_i$`)`;
$headNode \leftarrow bodyNode$.`search(`$clause.head$`)`;
if $headNode \neq bodyNode$ **then**
    $interestingness \leftarrow$`measureInterestingness(`$headNode.distribution$,$bodyNode.distribution$`)`;
    if $interestingness \geq interestingnessThreshold$ **then**
        return true;
else
    return false;

optimal intervals using techniques, such as [9], presented in Section 2.5, and which is represented by in 8 by function `searchInterestingIntervals`. If any relevant interval is found, we add the required the arithmetic literals to restrict the numerical variable domain and add the clause to the ILP's refinement graph.

After that, the algorithm also queries the correlation lattice for suggestion of new literals to be added to the clause

Chapter 4

Algorithmic Framework

This thesis was implemented in Java, using the code from Teflioudi [14] as base for the core-ILP algorithm, on which we added the changes in the refinement step mentioned at Section 3.7. Also, we change the knowledge base back-end to RDF3X, in order to improve the SPARQL queries performance.

4.1 Knowledge Base Back-end

For storing and retrieving RDF data we use RDF3X [15]. 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*.

All the facts are stored in a single triples table, with no schema, over which the indices are created. Moreover, one important characteristic of the indices is that they feature count-aggregated variants in each of the one- and two-dimensional projections.

The query processor explores the exhaustive indexation by relying mostly on merge-joins over the sorted index lists, and building operator trees that preserve a convenient ordering which allows further merge-joins. When it is impossible, RDF3X simply switches to hash-join.

RDF3X supports SPARQL queries, nevertheless, in the version 0.3.7 which was used in this thesis, many features of the query language were not implement. For example, optional patterns are not supported, and sorting and filtering of numerical attributes is not implemented, with numbers being dealt as strings. Therefore the expression $31 > 4$, for example, will be evaluated as the comparison of strings "31" > "4", which returns false instead of true.

As the evaluation of numerical expressions plays a key role in this thesis, we had to make changes to the original code in order to support this feature.

The RDF3X process is accessed by the Java application via the JDBC interface.

4.2 Preprocessing

4.2.1 Correlation Lattice Data Structure

We structure the correlation lattice data in two main classes: *CorrelationLattice* and *CorrelationLatticeNode*.

4.2.1.1 The CorrelationLatticeNode class

Basically, in the *CorrelationLattice* class, we store attributes related to the lattice as a whole, such as the root property, root node, maximum number of levels, as well as attributes consistent through all the lattice nodes, such as discretization boundaries, support and interestingness thresholds. Moreover, it maintains a hash set with all the lattice nodes in order to avoid duplicate nodes.

4.2.1.2 The CorrelationLatticeNode class

In a *CorrelationLatticeNode* we simply store the histogram data obtained with the discretization on root's numerical attribute, a set of pointers to the child nodes

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

4.2.1.3 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, we can extract a histogram for the node *hasIncome(x,y)hasEducation(x,z)* and its correspondent constants.

For the further levels, we simply add a pattern correspondent to each literal in the node.

4.2.1.4 Searching Rules in Correlation Lattice

In the lattice itself, 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 difference literal as head. Below, we show an example using the notation defined at 3.2.4:

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:

$$
\begin{aligned}
Rule_1: \quad & a_1 \text{ :- } b_1, c_1, r \\
& supp(Rule_1) = h(ra_1b_1c_1) \\
& acc(Rule_1) = \frac{h(ra_1b_1c_1)}{h(rb_1c_1)}
\end{aligned}
$$

$$
\begin{aligned}
Rule_2: \quad & b_1 \text{ :- } a_1, c_1, r \\
& supp(Rule_2) = h(ra_1b_1c_1) \\
& acc(Rule_2) = \frac{h_i(ra_1b_1c_1)}{h(ra_1c_1)}
\end{aligned}
$$

$$
\begin{aligned}
Rule_3: \quad & c_1 \text{ :- } a_1, b_1, r \\
& supp(Rule_3) = h(ra_1b_1c_1) \\
& acc(Rule_3) = \frac{h(ra_1b_1c_1)}{h(ra_1b_1)}
\end{aligned}
$$

Therefore, in order to search for rules in the correlation lattice, we can simply perform a breadth-first traversal in the lattice, and do such evaluation in each node.

Subsequently we can analyze the frequency and confidence distributions and determine whether any of the rules are interesting, using any of the techniques such as Brin et al. [9], discussed in 2.5, in order to search for optimal intervals.

Chapter 5

Experiments

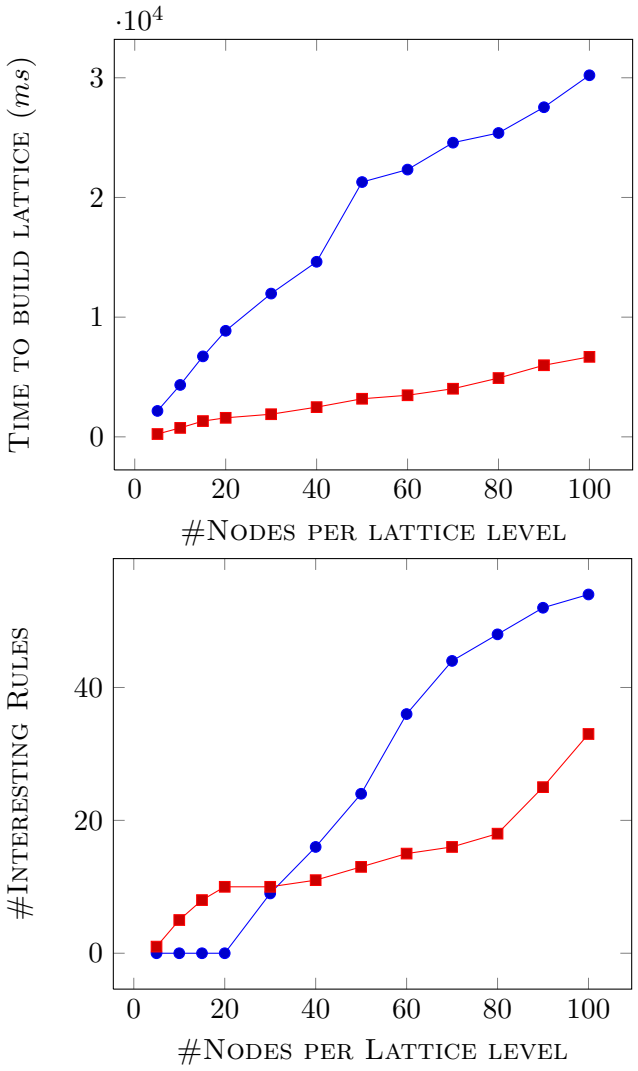
***Roughly explaining the experiments I did so far, we want to evaluate whether the heuristics are good. So we use US Census data to build a lattice (for income property) with the limitation of having up to n nodes at each level. So far I compared the KL-divergence*Support with support only as measures. 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



5.0.2 Example of Rules Learned

We selected a couple of interesting rules learned with our approach in order to illustrate the kind of results we obtain. On LinkedMDB, for example, we learned that films in English with Canadian director, and low budget are significantly more likely to be also Canadian than the ones with higher budget. The base rule shown below has confidence 0.36:

$country(X, canada)$:- $director(X, Z), bornIn(Z, canada), language(X, english), budget(X, Y)$

When we refine this rule, we find an interval with confidence 0.83:

$country(X, canada)$:- $director(X, Z), bornIn(Z, canada), language(X, english), budget(X, Y), Y \leq 80000$

Figure 5.2: Lattice with 4 levels

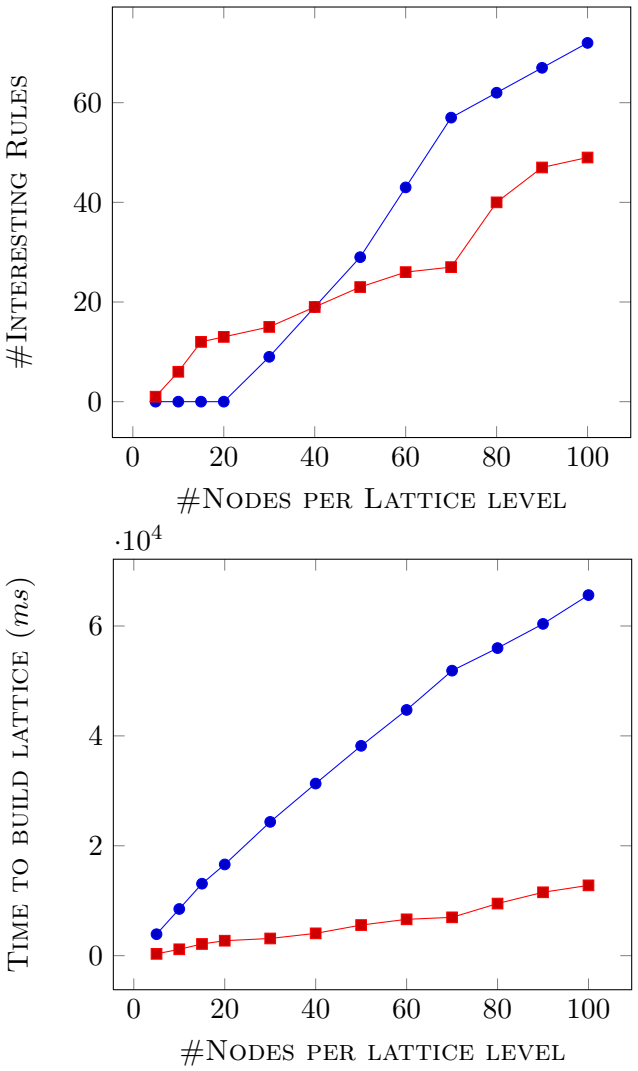


Figure 5.3: Recall-Accuracy

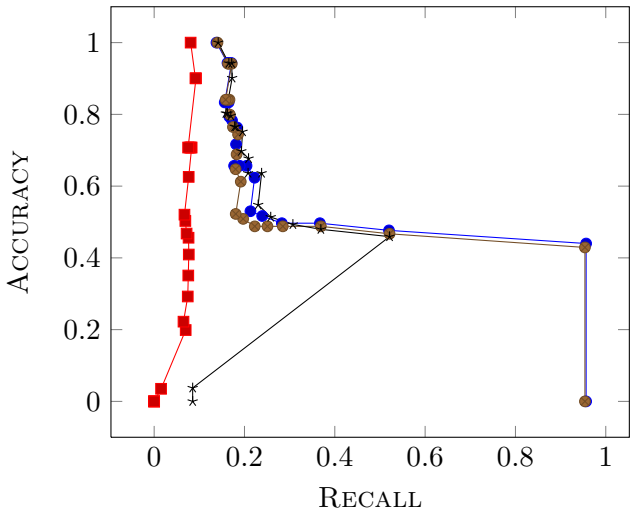
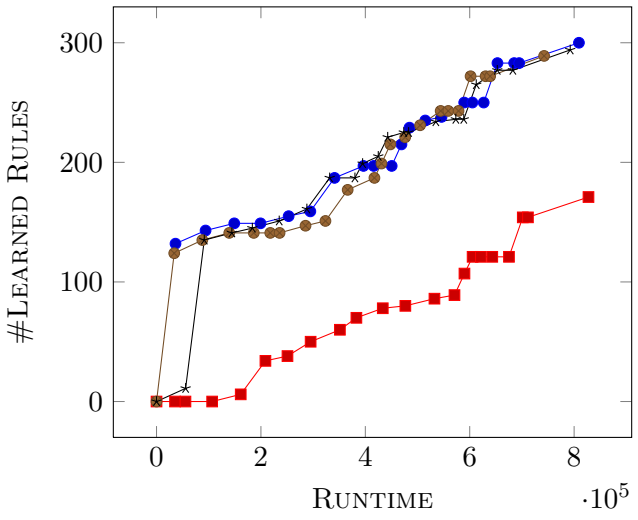


Figure 5.4: Runtime-LearnedRules



List of Figures

2.1 Part of the refinement graph for the family relations problem[3]. 12
2.2 Itemset Lattice example [7]. 16
2.3 Itemset Lattice generated by apriori algorithm for the example from Table 2.2 18
2.4 Example of buckets generated [9] 20
2.5 Execution trace of procedure shown in Algorithm 3. (a) Before first iteration, (b) after first iteration, (c) before second iteration, and (d) after second iteration[9]. 21
2.6 Semantic Web stack 25
2.7 Linking Open Data cloud diagram, by Richard Cyganiak and Anja Jentzsch. `http://lod-cloud.net` 26
2.8 Zoom of Figure 2.7 highlighting datasets relevant to this thesis 27

3.1 Example of frequency histograms from body support and positives support (left), and resulting confidence distribution (right) 30
3.2 Support distribution of body and positives from figure 3.1 31
3.3 Type hierarchy example 37
3.4 Discrete probability distributions of income by year quarter (left), and by education level (right) 41
3.5 Combination of relations $hasEducation$ and $bornInMonth$ 43
3.6 Correlation Lattice example 44
3.7 Node join example for independence test 44
3.8 Interestingness of adding a literal to the body of a clause 50

5.1 Lattices with 3 levels 58
5.2 Lattice with 4 levels 59

List of Tables

2.1 A simple ILP problem: learning the *daughter* relation [3] 10
2.2 Example of transaction database $\mathcal{D}$ [7] 16

3.1 Example data for categorical relation *hasSex* 31
3.2 Example of data linked to example from Table 3.1 33
3.3 Example of absence and presence of property as categories 34

Bibliography

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

[2] Nada Lavrac and Saso Dzeroski. *Inductive Logic Programming: Techniques and Applications*. Ellis Horwood, New York, 1994. ISBN 0134578708. URL http://www-ai.ijs.si/SasoDzeroski/ILPBook/.

[3] 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.

[4] Nada Lavrac and Saso Dzeroski. Background knowledge and declarative bias in inductive concept learning. In *AII*, pages 51–71, 1992.

[5] Georg Gottlob, Nicola Leone, and Francesco Scarcello. On the complexity of some inductive logic programming problems. *New Generation Comput.*, 17(1):53–75, 1999.

[6] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. *SIGMOD Rec.*, 22(2):207–216, June 1993. ISSN 0163-5808. doi: 10.1145/170036.170072. URL http://doi.acm.org/10.1145/170036.170072.

[7] Nicolas Pasquier, Yves, Yves Bastide, Rafik Taouil, and Lotfi Lakhal. Efficient mining of association rules using closed itemset lattices. *Information Systems*, 24: 25–46, 1999.

[8] 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.

[9] 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.

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

[11] 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.

[12] Nguyen Xuan Vinh, Julien Epps, and James Bailey. Information theoretic measures for clusterings comparison: Variants, properties, normalization and correction for chance. *J. Mach. Learn. Res.*, 9999:2837–2854, December 2010. ISSN 1532-4435. URL http://dl.acm.org/citation.cfm?id=1953011.1953024.

[13] S. Guiaşu. *Information theory with applications*. Advanced Book Program - McGraw-Hill Book Company. McGraw-Hill, 1977. ISBN 9780070251090. URL http://books.google.de/books?id=DohQAAAAMAAJ.

[14] Christina Teflioudi. Learning soft inference rules in large and uncertain knowledge bases. Master's thesis, Universität des Saarlandes, Saarbrücken, March 2011.

[15] 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.