Classification in the Presence of Background Domain Knowledge

João David Caires Vieira

Thesis to obtain the Master of Science Degree in Information Systems and Computer Engineering

Supervisor: Prof. Cláudia Martins Antunes

Examination Committee
Chairperson: Prof. Mário Rui Fonseca dos Santos Gomes
Supervisor: Prof. Cláudia Martins Antunes
Member of the Committee: Prof. Sara Alexandra Cordeiro Madeira

November 2014
Acknowledgements

I would like to thank to those who have been present and have in a way or another contributed to this thesis.

To my advisor, Prof. Cláudia Antunes, for all the guidance, patience and support.

To my friend, Liliana Fernandes, for all the patience and help with the drawing, and the redrawing, of plots and figures.

I would like to thank my parents for always being kind and supportive. I can not thank them enough for all the encouragement and dedication.

Finally, I would like to thank the taxpayers of Portugal for providing me with financial support, which was administered by the Portuguese national funding agency for science, research and technology (FCT) under project D2PM PTDC/EIA-EIA/110074/2009.
Resumo

Neste trabalho consideramos o problema de aprender a classificar um grupo de instâncias a partir de um conjunto de treino e de conhecimento de domínio existente. Tradicionalmente, duas abordagens muito diferentes têm dominado a área da classificação, uma baseada em representações lógicas, que faz uso da programação lógica para representar toda a informação conhecida, seja ela observações já classificadas ou conhecimento de domínio disponível; e outra que usa métodos puramente estatísticos e depende apenas de um conjunto de observações já classificadas para construir o modelo predictivo. A primeira abordagem lida melhor com a complexidade do mundo real enquanto a segunda se destaca pela boa capacidade em lidar com a incerteza, tantas vezes presentes nas aplicações práticas. Como tanto a complexidade, como a incerteza, fazem parte do mundo, vemos valor em explorar métodos que consigam lidar melhor com ambas as facetas. Propomos uma abordagem para adicionar conhecimento de domínio, representado numu linguagem de representação de conhecimento standard, para proporcionar às abordagens estatísticas a capacidade de melhor lidar com relações implícitas entre várias dimensões dos dados, que são parte do conhecimento de domínio das várias áreas de aplicação.

Palavras-Chave: Classificação, Árvores de decisão, Aspectos semânticos de data mining, Conhecimento de domínio, Ontologias
Abstract

We consider the problem of learning to classify a set of instances based on an available training set and on background domain knowledge. Traditionally, two very different approaches have dominated the area of classification, one based on logic representations that resorts to logic programming to represent the body of what is known, be it observed examples or available domain knowledge, and other that uses purely statistical methods, and only depends on a set of observed examples to build a prediction model. The first approach deals better with the complexity of the real world while the second excels on dealing with the uncertainty that is present in any practical application. As both complexity and uncertainty are part of the real world, we see value in exploring methods that can better deal with both facets. We propose an approach to add domain knowledge, represented in a standard knowledge representation language, to give statistical approaches the ability to better deal with the underlying relations between data that are known to exist in a given domain.

Keywords: Classification, Decision trees, Semantic aspects of data mining, Background Knowledge, Ontologies
Contents

1 Introduction 5

2 Literature Review 7
  2.1 Inductive Logic Programming 7
  2.2 Markov Logic Networks 11
  2.3 Decision Tree Learning 12
    2.3.1 Background Knowledge in Decision Tree Induction: EG2 13
    2.3.2 Ontology-driven induction of decision trees at multiple levels of abstraction 15
    2.3.3 Making Ontology-Based Knowledge and Decision Trees interact 15

3 Knowledge Representation in Machine Learning 18
  3.1 OWL 2: The Web Ontology Language 19
  3.2 Knowledge, Data and Uncertainty 21
  3.3 Summary and Discussion 23

4 Hierarchy-based Decision Tree Learner and Classifier 25
  4.1 Learning Decision Trees from Data 25
  4.2 HDTL: Hierarchy Based Decision Tree Learner 27
  4.3 Representing Feature Hierarchies 28
  4.4 Attribute Selection Criteria 29
  4.5 HDTC: Hierarchy Based Decision Tree Classifier 31
  4.6 Results 33
  4.7 Summary and Discussion 37

5 Hierarchy-based Naïve Bayes Learner and Classifier 38
  5.1 Building a Probabilistic Model from Data 38
  5.2 HNBL: Hierarchy Based Naïve Bayes Learner 40
  5.3 HNBC: Hierarchy Based Naïve Bayes Classifier 41
  5.4 Results 44
  5.5 Summary and Discussion 47

6 Ontology-based Decision Tree Learner and Classifier 49
  6.1 OWL 2 49
  6.2 OWL 2 EL 50
  6.3 ELK 51
  6.4 Structuring an Ontology to Support Classification Problems 52
List of Figures

3.1 Tree representation of the Mushrooms-Fungi hierarchy ........................................... 21
3.2 Knowledge states progression ................................................................. 22
3.3 Proposed knowledge model ................................................................. 23
4.1 Feature hierarchy for the attribute odor ....................................................... 28
4.2 Feature hierarchy with carried attribute values .............................................. 29
4.3 Representation of the various components and interactions of the hierarchy based decision tree learner, model and classifier ...................................................... 31
4.5 Example of a decision tree built by HDT ......................................................... 34
4.6 Mushrooms: Accuracy by size of training set in HDT, ID3 and C4.5 ....................... 34
4.7 Nursery: Accuracy by size of training set in HDT, ID3 and C4.5 ......................... 35
4.8 Mushrooms: Accuracy by percentage of abstract features using HDT, ID3 and C4.5 ..... 36
4.9 Nursery: Accuracy by percentage of abstract features using HDT, ID3 and C4.5 ...... 36
5.1 Mushrooms: Accuracy by size of training set in HNB and NB ............................... 45
5.2 Nursery: Accuracy by size of training set in HNB and NB ................................. 45
5.3 Mushrooms: Accuracy by percentage of abstract features using HNB and NB ...... 46
5.4 Nursery: Accuracy by percentage of abstract features using HNB and NB .......... 47
6.1 Representation of the various components and interactions of the ontology based decision tree learner, model and classifier ...................................................... 50
6.2 Cars: Accuracy by size of training set in OADT, ID3 and C4.5 ............................... 59
6.3 Cars: Accuracy by percentage of abstract features using OADT, ID3 and C4.5 .......... 60
6.4 Cars: Percentage of inferred nodes ........................................................... 60
6.5 Soybean: Accuracy by size of training set in OADT, ID3 and C4.5 ......................... 61
6.6 Soybean: Accuracy by percentage of abstract features using OADT, ID3 and C4.5 ...... 62
6.7 Soybean: Percentage of inferred nodes ....................................................... 62
List of Algorithms

<table>
<thead>
<tr>
<th>1</th>
<th>A generic standard algorithm for the induction of decision trees</th>
<th>27</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>A hierarchy based decision tree classifier</td>
<td>32</td>
</tr>
<tr>
<td>3</td>
<td>A generic standard algorithm for constructing the probability model</td>
<td>39</td>
</tr>
<tr>
<td>4</td>
<td>A hierarchy based naïve bayes classifier</td>
<td>43</td>
</tr>
<tr>
<td>5</td>
<td>Projects data set instances into the ontology as individuals</td>
<td>54</td>
</tr>
<tr>
<td>6</td>
<td>Obtains attribute values for the new generated attributes</td>
<td>55</td>
</tr>
<tr>
<td>7</td>
<td>An ontology based decision tree classifier</td>
<td>56</td>
</tr>
<tr>
<td>8</td>
<td>Translates instances from tabular to RDF triples</td>
<td>63</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

In spite of the great efforts that were made in the last decade in data mining algorithms, the problem of using existing domain knowledge to enrich and better focus the results on user expectations remains open to further developments [Cao 2010; Domingos 2003; Yang and Wu 2006].

While it is true that significant work has been done in some areas, namely pattern mining, to inject knowledge about the domain in the mining process, obtaining in this way a more sane number of results, better aligned with user expectations, it is also true that these ideas remain to be explored in the context of classification tasks.

Classification algorithms are supervised methods that look for and discover the hidden associations between the target class and the independent variables [Maimon and Rokach 2010]. Supervised learning algorithms allow tags to be assigned to the observations, so that unobserved data can be categorized based on the training data [Han et al. 2011]. In classification a model is a set of rules built from a group of already classified training data objects in order to forecast the classes of previously unseen data objects [Thabtah and Cowling 2007].

For domain experts to use this kind of models it is crucial that trust can be established. When the cost of making mistakes is very high, numerical validation is usually not enough. It is fundamental that they can understand the reasoning beyond the predictions and that this reasoning is aligned with their knowledge of the processes in the domain and their interactions. For this reason we will focus on human interpretable models of which decision trees are a well-known example.

One of the problems encountered in the automatic induction of classification rules from examples is the overfitting of the rules to the training data, in some cases resulting in excessively large models with low predictive power for unseen data [Bramer 2002]. Overfitting is the use of models that violate parsimony, i.e., that include more terms than are necessary or use more complicated approaches than are necessary. This is undesirable: adding irrelevant predictors can make predictions worse because the coefficients fitted to them add random variation to the subsequent predictions; the choice of model impacts its portability [Hawkins 2004]. Most current algorithms produce models that have a very low portability or are not portable at all: if a feature of some instance is changed by a synonym or is in a different language, chances are that a new model has to be learned.

Other problem is that classification, like other disciplines of data mining, suffers from a lack of focus on user expectations [Antunes and Silva 2014]. This is partly because most algorithms work on a purely statistical basis ignoring the semantics of the features, attributes and being unaware of the various relationships that can exist between them and that could otherwise be explored to produce less
complex, more actionable models.

It is our belief that the introduction of background domain knowledge is a key factor in the solution of the problems described above.

A body of formally represented knowledge is based on a conceptualization: the objects, concepts, and other entities that are assumed to exist in some area of interest and the relationships that hold among them [Genesereth and Nilsson, 1987]. A conceptualization is an abstract, simplified view of the world that we wish to represent. An ontology is an explicit specification of a conceptualization [Gruber, 1995].

In our opinion, the introduction of ontologies, as a means to formally represent existing background knowledge, in the learning process of classification algorithms will allow the production of more concise models by working at different levels of abstraction and exploring the relationship between concepts in the data set.

The main contributions of this work are:

1. a concept hierarchy guided decision tree learning algorithm, that is able to take advantage of user supplied feature (attribute value) hierarchies and learn a model that is able to deal with data specified at different levels of abstraction. We also describe how a classifier can be extended to be able to decide using decision trees built by our learner.

2. an extension to naïve Bayes that produces three dimensional conditional class probability tables, where the third dimension contains the different levels of abstraction for each feature. We describe how the classifier can be extended to use this new model to classify unseen instances with features at different levels of abstraction.

3. we extend our decision tree approach to make use of ontologies that go well beyond simple concept hierarchies. We make use of the favourable properties of the $\mathcal{EL}$ family of description logics to allow the use of ontologies that have enough expressive power to describe complex domains, while still allowing efficient (polynomial time) reasoning.

This document is organized as follows: chapter 2 reviews some of the work that has attempted to incorporate some form of domain knowledge in the process of learning classification models and using those models to classify unseen instances. Our overview goes from logical approaches like Inductive Logic Programming (ILP) to statistical ones like Decision Trees and Naïve Bayes.

Then, chapter 3 analyses some knowledge representation languages that might be suitable for the representation of domain knowledge in data mining, presenting and balancing the biggest tradeoffs that have to be made when selecting the right language for the problem at hand.

Next, chapter 4 describes an approach where we introduce a simple form of domain knowledge, feature hierarchies, in the induction of decision trees and present some results. We also apply this idea to Naïve Bayes in chapter 5 and show how it performs on some data sets.

In chapter 6 we go further and extend our ideas beyond feature hierarchies. We propose an approach that is able to incorporate domain knowledge that makes use of the full expressive power of $\mathcal{EL}$ family of description logics to define more complex facts about the domain.

Finally, chapter 7 concludes this work with some last thoughts and leaves some ideas for future work.
Chapter 2

Literature Review

Historically there have been two major approaches to research in artificial intelligence: one based on logic representations, and one focused on statistical ones. The first group includes approaches like logic programming, description logics, rule induction, etc. The second, more used in machine learning, includes Bayesian networks, hidden Markov models, neural networks, etc. Logical approaches tend to focus on handling the complexity of the real world, and statistical ones the uncertainty [Domingos et al., 2006] that is present in field applications.

This duality is clearly represented in classification where a lot of efforts where taken in the last decades in the research and development of algorithms that explored certain principles of statistics to build predictive models. Examples of algorithms following this approach include SVMs [Boser et al., 1992], back-propagation [Rumelhart et al., 2002], Naive Bayes, KNN [Altman, 1992], C4.5 [Quinlan, 1993], among others. These algorithms are usually very efficient in learning a model and the model produced yield good levels of accuracy for unseen data if the training set was properly balanced and sized. These kind of algorithms were the focus of most research in the last decades and saw wide adoption and acceptance by the industry.

On the other hand ILP is the most known representant of the logic approach to classification. In this kind of approach, in addition to the training set, an encoding of the known background knowledge is also provided. An ILP system will then derive a logic program as a hypothesis which entails all the positive and none of negative examples.

In section 2.1 we look at Inductive Logic Programming (ILP), the most traditional logic approach to classification, that since its early days has incorporated domain knowledge in the process. Then in section 2.2 we analyze Markov Logic Networks, a more recent approach that attempts to combine first-order logic and probabilistic models. In section 2.3 we review some attempts that have been made at introducing some forms of domain knowledge in the, traditionally, purely statistical approaches, namely on the induction of decision trees.

2.1 Inductive Logic Programming

Inductive Logic Programming (ILP) is one of the major approaches to the problem of classification which uses logic programming as a uniform representation for the existing training set, available background knowledge and model induced. In ILP the learned model is called a hypothesis and is expressed in first order predicate logic.
Formally the problem of classification in the context of ILP can be specified as follows [Blockeel and De Raedt, 1998]:

**Definition 2.1.1 (Classification with ILP).** Given: a set of classes $C$, a set of classified examples $E$ and a background theory $B$, find a hypothesis $H$ (a Prolog program) such that for all $e \in E$, $H \land e \land B \models c$, and $H \land e \land B \not\models c'$, where $c$ is the class of the example $e$ and $c' \in C - \{c\}$.

To counter the enormous complexity some restrictions are normally imposed, like not allowing recursion or limiting to Horn clauses. Refer to [Raedt, 1996] for a more detailed discussion on this.

Most ILP systems use the covering approach of rule induction [Muggleton and De Raedt, 1994]. For each iteration of a main loop a new clause is added to the hypothesis, explaining some of the positive examples. These examples are then removed and the loop continues until there are no positive examples remaining. At this point the hypothesis explains all positive examples. Meanwhile in an inner loop, individual clauses are created by searching the space of possible clauses that is structured according to a generalization or specialization operator. The search process is usually guided by some heuristic. An examples of such heuristic is to prefer clauses that cover many positive and few negative examples.

Usually the search starts with a clause with no conditions in the body and proceeds by adding them until it only covers positive examples. As this approach starts with a very general rule and iteratively adds literals to the clause to ensure that it covers only positive examples, i.e. is consistent, it is called a relative least general generalization, $rlgg$, [Plotkin, 1972] and is one of two common types of bottom-up search in the learning phase of ILP. It is however prone to be potentially infinite for arbitrary sets of background knowledge $B$. When $B$ consists of ground unit clauses only the relative least general generalization of two clauses is finite. Even so, the cardinality of the $rlgg$ of $m$ clauses relative to $n$ ground unit clauses can be $O(m^n)$ in the worst-case, making the production of such $rlggs$ not viable. Golem [Muggleton and Feng, 1992] is an example of an algorithm that follows this approach and avoids the problem of non-finite $rlggs$ by using extensional background knowledge. It does so by receiving a parameter $h$ and in at most $h$ resolution steps, it generates extensional background knowledge $B$ from intensional background knowledge $B'$ by generating all ground unit clauses derivable from $B'$ in such amount of steps. The $rlggs$ constructed by Golem were forced to have only a tractable number of literals by requiring $ij$- determinacy, which is equivalent to requiring that predicates in the background knowledge must represent functions. This condition is not met in a many real- world applications, including the learning of chemical properties from atom and bond descriptions.

Overcoming the determinacy limitation of Golem was one of the motivations of the ILP system Progol [Muggleton, 1995], a now well-known first order rules learner. Contrary to Golem, which is a specific-to-general learner, Progol uses general-to-specific search and combines it with inverse entailment.

FOIL [Quinlan, 1990] is an ILP system that learns Horn clauses from data expressed as relations. It explores some ideas that, at the time, had proved effective in attribute-value learning systems and extends them to a first-order logic. It is however restricted to rules expressible in function-free Horn clauses, is not incremental and requires that training set contains the target relation labeled with positive and negative tuples.

There are three main techniques to specialize a logic program:

1. add literals to the body of a clause
2. remove a clause from the program

3. perform a substitution, i.e., replace some variables by terms

Conversely, there are three major ways in which a logic program can be generalized:

1. add a clause to the program

2. remove literals from the body of a clause

3. replace some terms in a clause by variables

In its early days, ILP focused on automated program synthesis from examples and background knowledge, formulated as a binary classification task but has broadened to cover a variety of data mining tasks, from classification and clustering to association analysis [Muggleton et al., 2012].

Up until 1997, when top-down induction of decision trees [Quinlan, 1986, 1993] was already one of the most popular data mining techniques, the approach had almost totally been ignored in the field of inductive logic programming. At the time, with the exception of [Boström, 1995] almost every ILP system used a covering approach. The main reason for this was the discrepancies between clausal representation employed in ILP and the structure underlying a decision tree which was more naturally constructed by divide-and-conquer algorithms.

The main contribution of TILDE [Blockeel and De Raedt, 1998] was allowing the introduction of a logical decision tree representation that corresponds to a clausal representation.

**Definition 2.1.2** (Logical decision tree). A logical decision tree (LDT) is a binary decision tree that fulfills the following constraints:

- every test is a conjunction of literals (in first-order logic)

- a variable that is introduced in some node (i.e., does not occur in higher nodes) can not occur in its right subtree

In short, the second requirement is necessary because newly introduced variables are quantified within the conjunction and the right subtree only matters when the conjunction fails: if the conjunction fails ("there is no such X") it does not make sense to speak of this X further down the tree. The algorithm is very similar in spirit to C4.5 [Quinlan, 1993] and most heuristics are in fact direct implementations (the gainratio, post-pruning algorithm, etc). It essentially differs in the computation of the tests to be placed in a node by employing a refinement operator under $\theta$-subsumption. Refer to [Muggleton and De Raedt, 1994] for an analysis of this technique. The algorithm that TILDE implements works, in short, as follow: it receives as arguments a set of examples $\varepsilon$, a pointer to a node $T$ and the query $Q$ associated with the node. The background knowledge $B$ is considered to be available. If $\varepsilon$ is homogeneous enough, then $T$ becomes a leaf with the value of the most frequent class in $\varepsilon$. Otherwise a heuristic is used to guess the best element of $\rho(Q)$ which becomes $Q_b$. $\rho$ is an operator mapping clauses into sets of clauses, such that for any clause $c$ and $\forall c' \in \rho(c), c\theta$-subsumes $c'$. It can, for example, add literals to the clause or unify several variables in it. Once $Q_b$ is found it then finds a $C'$ such that $Q_b \leftarrow Q, C'$ and it becomes the test of the current node $T$. The example set $\varepsilon$ is then partitioned is two subsets $\{E \in \varepsilon | E \cup B \models Q_b\}$ and $\{E \in \varepsilon | E \cup B \not\models Q_b\}$ which are then passed as arguments of the algorithm for the construction of left and right subtree respectively.
Note that for this to work the set of examples \( \varepsilon \) is made to be a Prolog knowledge base (or an equivalent relational database), i.e., an example consists of multiple relations and each example can have multiple tuples for these relations. This is known as learning from interpretations.

Although TILDE learns significantly faster than Progol, both are much slower than C4.5. Some comparisons show a difference of two orders of magnitude and marginally worse accuracy [Roberts et al. 1998]. ILP systems are known to be much slower while learning in classification problems, so the first result is to be expected. The worse accuracy, however, is not and might be explained by the propositional nature of the data. Other comparisons [Dzeroski et al., 1998] making use of data with implicit relations between attributes, a setting that favours ILP systems, show these systems having better accuracies than C4.5 but confirm significantly worse performance while learning.

Although ILP systems benefit from relevant background knowledge to construct simple and accurate theories more quickly [Srinivasan et al., 1999], background knowledge that contains large amounts of information that is irrelevant to the problem being considered can, and have been shown to, hinder the search for a correct explanation of the data [Quinlan and Cameron-Jones, 1993]. Further, traditional ILP is unable to cope with the uncertainty of real-world applications such as missing or noisy information, a known drawback when compared to the statistical approach.

To overcome this, the ILP community is now focusing on combining the expressive knowledge representation formalisms traditionally used in logic programming, such as relational and first-order logic, with principled probabilistic and statistical approaches to inference and learning. This new area of research goes usually under the name of probabilistic inductive logic programming but is also referred to as statistical relational learning and aims to deal explicitly with uncertainty making it more powerful than ILP [De Raedt and Kersting, 2008].

Probabilistic ILP representations introduce essentially two changes:

1. clauses are annotated with probabilistic information such as conditional probabilities
2. the covers relation becomes probabilistic

A probabilistic covers relation softens the hard covers relation used in ILP and can be defined as the probability of an example given the hypothesis and background knowledge [De Raedt and Kersting, 2008].

Definition 2.1.3 (Probabilistic covers relation). A probabilistic covers relation takes as arguments an example \( e \), a hypothesis \( H \) and possibly the background theory \( B \), and returns the probability value \( P(e|H, B) \) between 0 and 1 of the example \( e \) given \( H \) and \( B \), i.e., \( \text{covers}(e, H, B) = P(e|H, B) \).

A simplistic attempt at defining the probabilistic ILP learning problem is the following:

Definition 2.1.4 (Probabilistic ILP Learning Problem). Given a probabilistic-logical language \( \mathcal{L}_H \) and a set \( E \) of examples over some language \( \mathcal{L}_E \), find the hypothesis \( H^* \) in \( \mathcal{L}_H \) that maximizes \( P(E|H^*, B) \).

[De Raedt and Kersting, 2008] further refine this definition and present three learning settings, inspired by the existing classical approaches.

Probabilistic learning from interpretations makes use of Bayesian networks to assign probabilities to interpretations. The Bayesian network has two components: the directed acyclic graph and the conditional probability distributions. Together they specify the joint probability distribution. The basic idea is to induce this Bayesian network from a Bayesian logic program together with a background
theory. The idea underlying Bayesian logic programs is to view ground atoms as random variables that are defined by the underlying definite clause programs. Two types of predicates exist: probabilistic or Bayesian and deterministic or logical. A set of Bayesian definite clauses, each of them in the form \( A | A_1, \ldots, A_n \) with \( A \) being a Bayesian atom and \( A_1, \ldots, A_n, n \geq 0 \) being Bayesian and logical atoms, constitute a Bayesian logic program. Also each Bayesian clause \( c \) is annotated with its conditional probability distribution to quantify the probabilistic dependency among ground instances in the clause, \( \text{cpd}(c) = P(A | A_1, \ldots, A_n) \).

Probabilistic learning from entailment integrates probabilities in the entailment setting by assigning probabilities to facts for a single predicate. As far as the author knows, it remains an open problem as how to formulate more general frameworks for working with entailment.

Probabilistic proofs attach probabilities to facts and treat them as stochastic choices within resolution. Logical hidden Markov models and relational Markov models, which I will briefly review in the next section, can be viewed as a simple fragment of them, where heads and bodies of clauses are singletons only, also known as iterative clauses.

Although probabilistic ILP takes a step further in terms of dealing with uncertainty it does not perform consistently better than equivalent statistical approaches in terms of accuracy. The computational complexity of the learning phase is also much higher. Even on relational data, typically a stronghold of ILP approaches, a new kind of techniques have been proposed, known as propositionalization techniques [Kramer et al., 2001], that transform structured data mining problems into a simpler format, typically a vector of features or an attribute-value representation which can then be directly input into standard data mining algorithms.

There has been surprisingly little work on probabilistic learning with datasets described using formal ontologies [Muggleton et al., 2012]. Ontologies are crucial to deal with semantic interoperability and with heterogeneous data sets.

## 2.2 Markov Logic Networks

A Markov logic network (MLN) [Richardson and Domingos, 2006] is an approach that combines first-order logic and probabilistic models in a single representation. It consists in a first-order knowledge base with a weight attached to each formula or clause. Like probabilistic ILP, it tries to bring together the ability of probabilistic models to efficiently handle uncertainty and the expressive power of first-order logic.

MLNs provide to the statistical approach a compact language to specify large Markov networks and the ability to incorporate into them a wide range of background knowledge. On the other hand, MLNs add to the first-order logic the ability to deal with uncertainty.

A Markov network or Markov random field is a model for the joint distribution of a set of variables \( X = (X_1, X_2, \ldots, X_n) \in \Phi \). While a Bayesian network is a directed acyclic graph whose arrows represent causal influences or class-property relationships, a Markov network is an undirected graph whose links represent symmetrical probabilistic dependencies [Pearl, 1988]. The graph has a node for each variable, and the model has a potential function for each clique in the graph. A potential function is a non-negative real function of the state of the respective clique.

In general, exact inference in Markov networks require a sum over the whole network [Gilks et al., 1996]. As in Bayesian networks, the conditional distribution of a set of nodes \( V' = \{v_1, \ldots, v_i\} \) given
values to another set of nodes \( T' = \{t_1, \cdots, t_i\} \) in the Markov network may be calculated by summing over all possible assignments to \( u \notin V', T' \). This is a \#P-complete problem, and as such computationally intractable in the general case. Approximate inference is more feasible and the most widely used method for this is Markov chain Monte Carlo [Gilks et al. 1996].

In traditional ILP a first-order KB is a set of hard constraints on the set of possible worlds: if a world violates even one formula, it has zero probability. MLNs soften these constraints so that when world violates one formula in KB it becomes less probable, but not impossible. A weight is associated to each formula and represents how strong a constraint it is. To know which of two worlds is more probable one analyzes the number of formulas each violate and the weight of these formulas. The greater is the height, the greater is the difference in log probability.

Formally, [Richardson and Domingos 2006]

**Definition 2.2.1** (Markov Logic Network). A Markov logic network \( L \) is a set of pairs \( (F_i, w_i) \), where \( F_i \) is a formula in first-order logic and \( w_i \) is a real number. Together with a finite set of constants \( C = \{c_1, c_2, \cdots, c_{|C|}\} \), it defines a Markov network \( M_{L,C} \) such that:

1. \( M_{L,C} \) contains one binary node for each possible grounding of each predicate appearing in \( L \). If the ground atom is false the value is 0, if it is true the value is 1.

2. \( M_{L,C} \) contains one feature for each possible grounding of each formula \( F_i \) in \( L \). The value of this feature is 1 if the ground formula is true, and is 0 if it is not. The weight of the feature is the \( w_i \) associated with \( F_i \) in \( L \).

Inference in Markov Logic Networks is a search where the goal is to find a stationary distribution of the system, or one that is close to it. This stationary distributions contains the most likely assignments of probabilities to the ground atoms of an interpretation (vertices in a graph).

Once this set of assignments is known, inference can be performed in the more traditional statistical sense of conditional probability, i.e., given a formula \( A \) and a formula \( B \) known to be true, find the probability \( P(A|B) \). Computing this over the whole network is, however, intractable since it subsumes logical inference which is NP-complete and probabilistic inference, known to be \#P-complete [Roth 1996].

The most widely used approximate solution to this problem is Markov chain Monte Carlo (MCMC) [Gilks et al. 1996] and in particular Gibbs sampling which samples each variable in turn given its neighbors in the graph (Markov blanket) and counts the fraction of samples that each variable is in each state. Even then, for any reasonably sized network, Gibbs sampling is too slow to be practical [Singla and Domingos 2005]. Other popular methods for inference in Markov networks include belief propagation [Yedidia et al. 2000] and approximation via pseudolikelihood.

### 2.3 Decision Tree Learning

On the other side of the spectrum, decision tree learning is traditionally a purely statistical approach to the classification problem. No explicit constraints are defined and there are no *a priori* formulas defining relationships between attributes. Nothing beyond a tabular representation of the raw data is used. The method uses a decision tree as a predictive model mapping observations about an item to conclusions about the target attribute value. The leaves of these trees represent the target attribute values and the
branches are conjunctions of features (other attributes values) that lead to those target attribute values. Once the decision tree is built, inference is thus trivial. Given a set of attributes $A = \{a_1, a_2, \ldots, a_n\}$ and a target attribute $T$ there are a set of features $F = \{f_1, f_2, \ldots, f_u\}$ where each element of the set $F$ is associated to exactly one element of $A$. Further it also exists a set $C = \{c_1, c_2, \ldots, c_u\}$ of classes where each class is associated to the target attribute $T$. An instance $I$ can then be seen as a injective function that to each element of $A$ maps one element of $F$. This injective function can also be total if there are no missing attributes.

Starting at the root of the tree, each node has an associated attribute from $A$ and a number of branches labeled with a feature from $F$. To classify the instance one must reach a leaf and to do so, at each node must choose the branch so that the feature $f_i$ associated with that branch is equal to $I(a_i)$ where $a_i$ is the attribute associated with the current node. This is both simple and computationally efficient.

The only challenge is thus to learn the model, i.e., build the decision tree from an already class-labeled training set. The quality of the tree produced is the determining factor in the accuracy of the predictions that will be made using it. Several algorithms exist, the most known are ID3\footnote{Iterative Dichotomiser 3} \cite{quinlan1986} and its evolution C4.5 \cite{quinlan1993} and CART\footnote{Classification And Regression Tree} \cite{breiman1993}.

The popularity of decision trees is related with its simplicity and ability to work with data that underwent little preparation, as described above, but also because it is a white box model, i.e, the model (tree) produced and used to make the predictions can be easily represented in a simple human understandable form. It also inherits the strengthes of the statistical approach, i.e., it is able to scale up relatively well and is robust, i.e., performs well even when its assumptions are somewhat violated by the true model from which the data is generated. This is related with the inherent ability of the statistical approach to deal with uncertainty.

Statistical approaches, however, ignore the complexities of the real world: it is not possible to express or make use of existing background or domain knowledge, to explicitly state relationships between attributes or hierarchies of features nor constraint the results by facts which are known to be true, even if not represented in the subset of data being fed to the learning algorithm.

Research in logic programming eventually started incorporating probabilities and adopting concepts and ideas that traditionally were only found in the statistical approaches in order to better deal with the inherent uncertainty of real world applications \cite{domingos2006}. These approximation must also be made from the other side of the spectrum, i.e., statistical approaches have to gain by adopting some of the ideas and concepts of logic programming in order to better deal with the complexity of the real world.

### 2.3.1 Background Knowledge in Decision Tree Induction: EG2

\cite{nunez1991} was one of first approaches to extend the ID3 decision tree learner to make use of background knowledge in order to explore various types of generalizations, reduce the complexity of the generated decision trees and reduce the classification cost. To better illustrate the concept of classification cost in this context, consider an expert in brain tumors that receives a patient with a headache. The expert does not recommend the Scanner as a first diagnostic test, although it is the most effective one, because he has in mind economic criteria. Thus the expert asks simple questions and

---

1 Iterative Dichotomiser 3
2 Classification And Regression Tree
orders other more economic tests in order to discriminate the simple cases and only recommends such
an expensive exam for the complex ones.

EG2 follows this approach. It is an inductive algorithm that generates a decision tree from a set of
examples, a user-defined IS-A hierarchy, the cost of measurement of each attribute and some data about
the degrees of economy and generalization. These data will influence directly the search space that
the algorithm must undertake. ID3 selects attributes at each level of the tree based on the Information
Gain $I$ of that attribute. EG2, however, uses another criterion, that the author called $ICF$, Information
Cost Function, which essentially tries to be a "cost/benefit" metric and is defined as the ratio between
cost and the discrimination efficiency of the attribute.

$$ICF_i = \frac{(cost_i + 1)^{\omega}}{2^{\Delta I_i} - 1}$$

where:

$\omega$ is a calibration variable of the economic criterion

$\Delta I_i$ is the information gain of attribute $i$.

$ICF$ is calculated for each attribute and EG2 then selects the attribute with the smallest $ICF$.

EG2 performs two types of generalizations beside the typical ‘dropping condition’ performed by
any top down induction of decision tree (TDIDT) algorithm. The first makes use of the ISA hierarchy to
climb the generalization tree [Michalski, 1983] and second tries to apply the union of symbolic values
if this union fulfills certain criteria of completeness and consistency.

Inconsistency is a state where a leaf has at least two examples in the subset of examples that are
described equally but have different classes and thus cannot terminate.

Incompleteness is detected by measuring the proportion of its observable values in all leaves of
the subtree below the said generalization. To be considered complete it should be greater than a
user-specified threshold. It is 1 when each leaf contains each of the values of the generalization.

$$CF = \frac{\sum_{m=1}^{m} \sum_{n=1}^{n} \text{leaf j with i-th value of generalization}}{m \text{ leaves} \times n \text{ values of generalization}}$$

The algorithm then works as follows: using the economic metric in Equation 2.1 it selects the best
scoring attribute. A list $L$ maintains the more general abstract values and those observable values that
do not have abstract values associated. EG2 selects one abstract or observable value according to the
following criteria:

1. Abstract values with more observable attributes (more general)

2. If there is a tie or if there are only observable values, then partition the set of examples into
   subsets, each one according to each possible generalization and measure the entropy of the class
   of each subset of examples. Select the generalization that produced less entropy. The goal of this
   step is to choose the generalization that best classifies the examples.

The algorithm then generates a subtree according to this abstract or observable value. If the subtree is
consistent and complete, it is saved. EG2 then tries to get a better generalization than the one saved, if
possible.

To get a better generalization the algorithm tries the best union of abstract values and observable
values. EG2 attempts the former valid value and other abstract and observable values and builds a
subtree according to this union. It iterates until an inconsistency or incompleteness is found at which point the last saved subtree is used. In cases where an observable value cannot be generalized, a subtree is generated according to this observable value. In this case there is no difference to ID3. The process for each selected attribute stops when there are no more values in the list \( L \).

EG2 focuses mainly on economy of resources and its main contribution was to include in the learning process this part of the common-sense reasoning. However no standard way of representing this knowledge was presented which makes it unsuitable for representing other types of background knowledge. It is also limited to a IS-A hierarchy and does not make use of other logic primitives that would allow the definition of more complex relations.

### 2.3.2 Ontology-driven induction of decision trees at multiple levels of abstraction

More recently, [Zhang et al., 2002] described an ontology-driven decision tree learning algorithm to learn classification rules at multiple levels of abstraction. Although called ontology-driven, what the proposed solution really uses is a taxonomy, i.e., a set of ISA relations associated with each attribute. It consists in a top-down concept hierarchy guided search in a hypothesis space of decision trees.

Traditionally, decision tree learning algorithms recursively select at each step, an attribute form a set of candidates attributes based on an information gain criterion. Each node in a partially constructed decision tree has, thus, associated with it a set of candidate attributes to choose from, for growing the tree rooted at that node.

In the algorithm proposed by [Zhang et al., 2002], each attribute has associated with it a hierarchically structured taxonomy over possible values of the attribute. At each step the algorithm chooses, not only a particular attribute but also an appropriate level of abstraction in the ISA hierarchy.

It starts with abstract attributes, i.e, groupings of attribute values that correspond to nodes that appear at higher levels of a hierarchically structured taxonomy. Each node of a partially constructed decision tree has associated with it a set of candidate attributes drawn from the taxonomy associated with each of the individual attributes. For each node, a set of nodes on the frontier is maintained and the information gain for the corresponding attributes is computed. It then selects, from the set of candidates under consideration, the one with the largest information gain.

The described algorithm can be seen as a best-first search through the hypothesis space of decision trees defined with respect to a set of attribute taxonomies.

This approach suffers from some of the same problems of EG2, described earlier, as the authors never specify a standard format to represent the domain knowledge and the knowledge that can be represented is restricted to ISA relations, a small subset of an ontology. It is however a tentative step in a meaningul direction, facilitating the discovery of classifiers at different levels of abstraction.

### 2.3.3 Making Ontology-Based Knowledge and Decision Trees interact

Other approach is to promote interaction with domain experts during the process, giving them the ability to guide the algorithm. [Johnson et al., 2010] proposes a generic interactive procedure, relying on an ontology to model qualitative knowledge and on decision trees as a data-driven learning method. Domain knowledge from experts and literature is formalised by using an ontology to specify a set of concepts and the relations linking them, giving a structure that facilitates the interaction with domain experts. In the proposed procedure an ontology \( \Omega \) is defined as a tuple \( \Omega = \{ \mathcal{C}, \mathcal{R} \} \) where \( \mathcal{C} \) is a set of
concepts and $\mathcal{R}$ is a set of relations.

Given a dataset $\mathcal{D}$ containing $K$ attributes and $I$ instances, each attribute $A_k, k = 1, \cdots, K$ is a concept $c \in \mathcal{C}$ in the ontology $\Omega$.

Each attribute, represented in the ontology as a concept $c$, may be associated with a definition domain which can be numeric, i.e., a closed interval $[\text{min}_c, \text{max}_c]$; flat, i.e., a non-hierarchized set of constants or hierarchized symbolic, i.e., a set of partially ordered constants that are themselves concepts in $\mathcal{C}$.

In this approach, the other constituent of an ontology $\Omega$ is the set of relations $\mathcal{R}$ that is composed of:

1. the subsumption relation, denoted by $\preceq$, defines a partial order over $\mathcal{C}$. For a $c \in \mathcal{C}$, $\mathcal{C}_c$ denotes the set of subconcepts of $c$ such that $\mathcal{C}_c = \{c' \in \mathcal{C}|c' \preceq c\}$.

2. the functional dependencies express the constraints between two sets of attributes and is represented, in the ontology, as a relation between two sets of concepts of $\mathcal{C}$.

Let $X = \{X_{k1}, \cdots, X_{k2}\} \subseteq \mathcal{C}, 1 \leq k_1 \leq k_2 \leq K$ and $Y = \{Y_{k3}, \cdots, Y_{k4}\} \subseteq \mathcal{C}, 1 \leq k_3 \leq k_4 \leq K$ be two disjoint subsets of concepts. $X$ is said to functionally determine $Y$ iff there is a function $f$ such that $f : \text{Range}(X_{k1}) \times \cdots \times \text{Range}(X_{k2}) \leftarrow \text{Range}(Y_{k3}) \times \cdots \times \text{Range}(Y_{k4})$. Two instances of such functional dependencies are required in [Johnson et al., 2010] approach:

- a property relation $\mathcal{P} : \mathcal{C} \leftarrow 2^{[\mathcal{C}]}$ that maps a single concept to a set of other concepts that represent associated properties.
- a determines relation $\mathcal{D} : 2^{[\mathcal{C}]} \leftarrow \mathcal{C}$ which specifies a subset of concepts whose values entirely determine the value taken by another concept.

Note that in this approach the ontology is not a mere taxonomical hierarchy and has enough expressive power for, e.g., discretize continuos variables into categories according to knowledge provided by field experts.

The idea is, then, to use this ontology to apply certain transformations to the dataset before the decision tree algorithm runs. The authors propose three kinds of transformations:

1. Replacement of a variable by new ones. This transformation consists of substituting a certain attribute by some of its more relevant properties which become new attributes. Consider, for instance, the attribute vitamin. If $\mathcal{P}(\text{vitamin}) = \{\text{solubility, thermosensitivity}\}$ then solubility and thermosensitivity may become new attributes. For a given instance, where VitaminA was a feature, now two features exist in its place: Liposoluble and Thermolability.

2. Merging of variables in order to create a new one. This transformation is useful to facilitate the interpretation, as less variables are considered it is likely that simpler model is produced, and to avoid considering as significant single variables that are only significant together. As an example, consider that in a given domain an expert is interested in cholesterol as one of various predictors for a given disease but the available dataset has, among others, HDL, LDL and VHDL which, divided, are of no particular interest to the domain experts. Therefore, it makes sense to replace HDL, LDL and VHDL by a new attribute called, e.g., cholesterol level. The features of this new attribute can then be defined in the ontology as a combination of the previous.
3. Grouping the modalities of a variable using common properties. In this transformation, rather than considering the modalities themselves, the subsets of modalities corresponding to a particular feature are considered.

Suppose, as an example, that we have an attribute \textit{water}, that, in the ontology, has, among others, a property \textit{pH} and that we are interested in the types of water in each instance but would prefer to group them by \textit{pH}. If for some instance we have a feature \textit{Tap water} for the attribute \textit{water} and the ontology defines that the value of the property \textit{pH} for \textit{Tap Water} has \textit{BasicpH} and \textit{NeutralpH} for all kinds of \textit{Water}. Then the new attribute \textit{Water’} will have only two features, \{TapWater\} and \{Deionized water, Distilled water, Distillied deionized water\}.

The proposed procedure may be described as an interactive approach that starts with an initial ontology $\Omega_0 = \{C_0, R_0\}$ that can be empty or obtained from domain experts. It is also assumed that the attributes in the provided dataset $D_0$ coincide with the concepts defined in the ontology. Then, at each step $i$:

1. Build a model $M_i$ (e.g., a decision tree) from the data set $D_i$.
2. Calculate the numerical accuracy of $M_i$ and discuss significance with domain expert.
   (a) If the domain expert is satisfied, stop the process.
   (b) Else elicit from the expert a set of transformations, as described above, to be applied to the ontology $\Omega$. Let $\Omega_{i+1}$ be the resulting ontology.
3. Apply the transformations in $\Omega_{i+1}$ to the data set $D_i$ to obtain a new data set $D_{i+1}$.
4. Let $i := i + 1$ and repeat.

The authors propose the following two ways of evaluating the method described:

- \textit{subjective} expert evaluation, assessing their confidence in the obtained results, and identifying possible inconsistencies in the model.
- \textit{objective} numerical evaluation where the results and stability of the induced models are measured:
  - The misclassification rate, $Ec = \frac{MC}{N}$, where $MC$ is the number of misclassified items and $N$ is the data set size, computed with a cross validation procedure or on the whole set.
  - Tree complexity, $N\text{rules} + N\text{nodes}/N\text{rules}$, where $N\text{rules}$ is the number of leaves (equivalent to the number of rules), and $N\text{nodes}$ is the total number of nodes in the tree.

[Johnson et al., 2010] makes a great contribution by formalizing the structure of an ontology that is not a mere taxonomical hierarchy in the context of classification problems and also identifying a small subset of the huge expressive power of ontologies, in the form of the transformations described previously. However the whole process requires the time and attention of domain experts at every step. Also, although the structure of the ontology is clearly specified in theory, the authors do not propose a standard way of writing them in practice.

We believe that there is room for improvement by enlarging the subset of transformations to include some others that are useful in the context of classification, to automatise the whole process (or, at least, more parts of it) so that domain experts are not required at every step, e.g., defining new tree evaluation criteria that would allow an algorithm to consider all transformations and make a choice that produces a simpler and more accurate model.
Chapter 3

Knowledge Representation in Machine Learning

Knowledge representation is a field of artificial intelligence focused on the design of computer representations that capture information about a certain domain so it can later be used to help tackle the complexity of real world problems.

A key trade-off in the development of a knowledge representation language is that between expressive power and the computational complexity involved in reasoning about said knowledge. First order logic sits in the extreme regarding expressive power and has become a de facto standard in mathematics and some areas of philosophy to formally define general propositions.

Unsurprisingly, the first approaches to classification that made use of existing knowledge in addition to the set of labeled instances expressed this knowledge in first order logic. Perhaps a bit more surprising is the fact that the set of classified observations was also defined in a subset of first order logic [Quinlan and Cameron-Jones 1993; Muggleton and Feng 1992]. These strategies, that go under the name of Inductive Logic Programming (ILP), have however suffered the consequences of using such an expressive knowledge representation formalism and have always lagged behind statistical approaches in terms of performance. This is not surprising: if the background knowledge is not restricted an ILP problem may not be decidable and even restricting it to determinate Horn-clauses still yields a problem that is PSPACE – hard [Kietz 1993] (note that $P \subseteq NP \subseteq PH \subseteq PSPACE$).

On the other hand, the few statistical approaches that attempted incorporating background knowledge in the learning and classification process have so far used ad hoc methods to define it. It is clear that not much thought is given to these representations as often they are not even formally presented and are used only to define very specific hierarchical relations between concepts, lacking any meaningful expressive power. Not surprisingly they are normally used in a specific approach and later forgotten. Subsequent attempts at similar problems develop their own incompatible formalisms, rewrite all the domain knowledge in an equally constrained manner and fall themselves into oblivion not long after.

There is a significant body of work in the area of knowledge representation and reasoning, specifically in ontology engineering. A number of ontology languages with well studied properties have been proposed like CycL [Lenat 1995], KL-ONE [Brachman and Schmolze 1985], OWL [Motik et al. 2009] among many others and are often accompanied by reasoning or inference engines [Hayes-Roth et al. 1983].
We believe that making use of existing knowledge representation formalisms instead of developing another *ad hoc* language will allow existing knowledge to be reused and shared, will enable us to take advantage of well studied properties to reach the right tradeoff between expressive power and practicality, and will permit the use of automated reasoning engines when doing so allow us to better explain the underlying relations between the data.

With this in mind we went on to find the right ontology language for representing background domain knowledge in our approach. The three main factors driving this decision were:

1. It should be a standard or at least a *de facto* standard, so existing domain knowledge can be reused and ontologies written today can be shared and reused by others in the future.

2. Strike the right balance between the expressive power and the computational complexity involved in reasoning in such a language. We would like to have at least existential quantification, intersections, concept inclusion (allows the construction of concept hierarchies), equivalence, disjointness and assertions, domain and range restrictions.

3. There is at least a reasoner that can perform realization in \( P \), i.e., compute the implied instance/-type relationships between named instances and concepts, in polynomial time.

First order logic would satisfy the first constraint and offers more than enough expressive power but as a consequence can give rise to undecidable ontologies. Many Description Logic (DL) models are built around the decidable fragments of first order logic and although more expressive than propositional logic still have more efficient decision problems than first-order logic.

Among Description Logic models, the Web Ontology Language (OWL 2) has seen wide adoption, propelled perhaps by the rise of the semantic web, and benefits from an enthusiastic community. A great deal of research in automatic reasoning and inference has been focusing in OWL 2 [Shearer et al., 2008; Sirin et al., 2007; Tsarkov and Horrocks, 2006]. Particularly, ELK [Kazakov et al., 2014] runs in polynomial time making OWL 2 meet our three main requirements.

### 3.1 OWL 2: The Web Ontology Language

An ontology is a set of precise descriptive statements about some subset of the world that constitutes the *domain of interest*. OWL 2 is a knowledge representation language developed to formulate, share and allow reasoning with knowledge about a domain of interest. It is not a programming language, as it only describes a state of affairs in a logical way.

Reasoners are tools that can be used to infer further information about that state of affairs and although the manner in which these inferences are realized algorithmically is not part of the specification, the correct answer to any such question is predetermined by the formal semantics.

The three main components of an ontology in OWL 2 are:

- **Entities** are elements that refer to objects in the domain
- **Axioms** are the basic propositions expressed by an ontology
- **Expressions** are combinations of entities that form more complex constructs from basic ones. An expression might also be composed by other expressions.
To formulate explicit knowledge, it is useful to assume that it consists of elementary pieces that are often called propositions or statements. Propositions like “creosote is a bad smell” or “all mushrooms are fungi” are examples of such basic statements. In fact, every OWL 2 ontology can be seen as just a collection of such basic “fragments of knowledge”. Propositions that appear in an ontology are called axioms in OWL 2, and the ontology asserts that its axioms are true.

These propositions are often composite constructs, formed by more than one type of components e.g. stating that an object is part of a category “Green is a light colour” or declaring what characteristics objects of the world must have in order to belong to a certain category “mushrooms in the Agaricus family have chocolate spore print colour and a smooth cap surface”.

All basic components of propositions, be they objects (e.g. Green), categories (e.g. Agaricus) or relations (e.g. spore print colour, cap surface) are called entities. In OWL 2, objects are called individuals, categories are classes and relations are known as properties. Two types of properties exist. Object properties relate objects to objects (like a mushroom to its spore print colour), while datatype properties assign data values to objects (like an age to a mushroom).

Entities can be combined into expressions. As an example, the atomic classes “mushroom” and “medicinal” could be combined conjunctively to describe the class of mushrooms that can be used for medicinal purposes. The resulting class expression could then be used in propositions or in other expressions. As such, expressions are essentially a special kind of entity defined by their structure.

Axioms are the constructs that allow these combinations. Several axioms exist and they can be used to combine entities and expressions. These combinations are themselves expressions that can also be combined with other expressions or entities with axioms.

We previously stated that “all mushrooms are fungi”. This means that whenever we know some individual to be a mushroom, that individual must also be a fungus. This relation cannot, however, be derived solely from the labels “mushroom” and “fungi” but is part of the existing domain knowledge in biology. In order to enable an automated system to draw the desired conclusions, this relation must be made explicit. In OWL 2 this can be done by using a subclass axiom, as it is done in OWL fragment 3.1.

As a rule of thumb, a subclass relationship between two classes $A$ and $B$ can be specified, if the phrase “every $A$ is a $B$” makes sense and is correct. It is common in the construction of ontologies to use subclass axioms not only for sporadically declaring this kind of dependencies, but also to build whole class hierarchies by specifying the generalization relationships of all classes in the domain of interest.

**OWL fragment 3.1**: Definition of a simple class hierarchy for the genera *Agaricus* and *Lepiota*. It follows that all individuals in these genera will also belong to the class Mushroom and all individuals in the class Mushroom will also be part of the class Fungus.

```
1 Class: Agaricus
2 SubClassOf: Mushroom
3 Class: Lepiota
4 SubClassOf: Mushroom
5 Class: Mushroom
6 SubClassOf: Fungus
7 Class: Fungus
```

The semantics of some knowledge representation languages are defined in a way that presumes that
what is not currently known to be true must be false. This is known as the closed world assumption. In such a language, given OWL fragment 3.1 and an individual $i_1$ that is known not to be of the genus *Agaricus*, then it is possible to conclude that $i_1$ must belong to the genus *Lepiota*.

**Definition 3.1.1 (Closed world assumption).** Given a class $A$, two individuals $i$ and $j$, and the statement $A(i)$, then the statement $\neg A(j)$ is true.

OWL 2 is not one of these languages, making instead the open world assumption. This essentially codifies in the language the belief that in general no single observer or agent has complete knowledge of the domain and therefore cannot make the closed world assumption. Looking back at OWL fragment 3.1 and given an individual $i_1$ known not to be of the genus *Agaricus* nothing can be said about its class. It can certainly be part of the class *Lepiota* but it can also be part of any other unknown class.

**Definition 3.1.2 (Open world assumption).** Given a class $A$, two individuals $i$ and $j$, and the statement $A(i)$, then it is not possible to know if $\neg A(j)$ is true.

![Figure 3.1: Tree representation of OWL fragment 3.1](image)

Now that contrary to what this illustration might suggest Fungus and Mushroom are not equivalent, it is only known that all mushrooms are fungi, but there might be fungi that are not mushrooms.

### 3.2 Knowledge, Data and Uncertainty

Not all knowledge is created equal. Rather it is a continuum of representations with varying levels of value and actionability. These levels or states form a progression from the lowest level, where usability is marginal or potential to higher levels where usability is clearer and more immediate [Holsapple, 2004]. Through various kinds of knowledge processing one may progress from lower to higher states, increasing the relevance of knowledge with respect to accomplishing some concrete task. The highest state, a decision, is knowledge indicating a commitment to take some action and results from the processing of knowledge at other levels. Figure 3.2 shows a possible set of knowledge states and possible operations to jump from one state to another. The number of states or the concrete operations used to go from one specific state to another are not important for the point being made, just that a set of states with varying degrees of usability or actionability exist and that it is possible to progress to a higher state by executing some operations on the knowledge at lower states.

These ideas translate well to a classification problem. Observations are data, a low state with potential but no immediate actionability. Classification algorithms, at a very high level, essentially apply a set of processing steps to these labeled observations and, hopefully, produce a model capable of making decisions about the class of previously unseen instances. This model is then at the highest knowledge state, its actionability is clear and immediate.
Figure 3.2: The progression from lower knowledge states with marginal usability to higher knowledge states with immediate usability. Note that is possible to progress from one state to another by applying some knowledge processing technique.

Where does domain knowledge sit in this progression? It does not have the immediate actionability of a decision, otherwise everything needed to classify new instances would already be known and no learning process would take place. On the other hand, as it is a formalization of knowledge provided by domain experts it is reasonable to assume that it is at least more structured and has better usability than mere observations. That is because experts already partially processed these knowledge by gathering, selecting and analyzing data from multiple sources and experiences in the domain. That is how they become experts.

ILP systems do not, traditionally, make this distinction and as such both observations and domain knowledge contribute equally to the hypothesis being generated, that is, the hypothesis has to satisfy the domain knowledge, all the positive observations and none of the negative. This assumes that we are absolutely sure about the label of all instances, which is seldom the case, and about the relevancy of every statement in the domain knowledge for the problem at hand, which does not always happen.

The approaches we propose attempt to capture in their structure the idea that labeled instances and domain knowledge are at different knowledge states and should contribute in different ways to the model being generated. With this in mind we use automatic logical inference on domain knowledge and the new propositions that are generated go back into the body of domain knowledge. This is reasonable because this kind of knowledge, by its nature, was already selected and analyzed by a domain expert and is not expected to be noisy or false. It can, however, be irrelevant to the problem at hand.

To deal with the possibility that some propositions in the domain knowledge are irrelevant to the classification problem under consideration we avoid the use of logical inference to construct the model from the domain knowledge, i.e., we allow and use logical inference inside the existing domain
knowledge but avoid this kind of strong inference when building the model. Consider for example that the following propositions are part of the existing domain knowledge: “Lepiota have white gills, white spores and have rings on the stems”, “MushroomX has white gills and white spores”, “MushroomX has rings on the stems”. From the later two assertions about “MushroomX” and the first proposition about “Lepiota” one can logically infer that “MushroomX” belongs to the class “Lepiota”. This new proposition will be added to the domain knowledge but may or may not be used in the model being built.

Domain knowledge can add extra dimensions to the existing labeled instances, like the species of a mushroom, but whether or not this dimension will be part of the model depends on how it helps explain the underlying relations between features and the value of the target attribute. In essence this means that although the decision to add a new dimension is driven by logical inference, the decision to incorporate that extra dimension in the model is driven by statistical inference.

### 3.3 Summary and Discussion

We made the case for using the Web Ontology Language (OWL 2) as our knowledge representation language for the existing background domain knowledge. We briefly review the main assumptions that went into the design of the language, its structure and design trade-offs. We showed that it achieves a reasonable balance between expressive power and the computational complexity involved in doing reasoning with ontologies written in this language.

We also presented our knowledge model and made the case for using logical inference to generate more propositions from the existing domain knowledge but using statistical inference when deciding which of these propositions will influence the model and which attributes will have more, if any, weight.

Regarding the kind of processing we do inside the existing domain knowledge note that using statistical inference would not be practical at all, each proposition appears usually only once, whether one uses it to generate new domain knowledge or not has no statistical basis. We don’t have multiple observations of that proposition to draw any statistical significant conclusions. However we make the
case that, compared to the labeled instances in the training set, there is much less noise and uncertainty in the domain knowledge as the former are mere observations while the last is knowledge that was already selected, analyzed and processed by domain experts.

On the other hand using logical inference and blindly incorporating this domain knowledge in the model being generated would force this knowledge to not only be true but also relevant to the classification problem at hand. This is not reasonable. First, it would require an expert to at least partially rebuild the background knowledge for each different problem. There are multiple distinct problems in any given domain and what is relevant to one of them may not be relevant to the others. Second, it would require some kind of insight not only about the domain but about the problem itself, i.e., an expert that already knows that some combination of features will be important in predicting the target attribute. To avoid these requirements we use statistical inference instead when picking which propositions to use when building the model. This is only possible by looking at these propositions and at the training set simultaneously and checking which of them help better explain the data.

Perhaps the easiest point we make is in distancing ourselves from traditional ILP regarding the kind of inference that is used to build the model from the training set. A logical approach would force the model to explain all positive observations and none of the negative. This is clearly not reasonable when dealing with the kind of noise that is present in the training set and would leave no room to deal with the uncertainty that is present in real world applications.
Chapter 4

Hierarchy-based Decision Tree Learner and Classifier

In this chapter, we start by reviewing standard purely statistical approaches to decision tree learning. We observe that these approaches are made of three distinct but connected components: a learner, that given a set of labeled instances produces a model; the model itself, which is a tree where each node is an attribute, each edge is a possible attribute value and each leaf corresponds to the value of the target attribute; and a classifier which uses the model learned to classify previously unseen instances.

We extend these components and present HDTL, a hierarchy guided decision tree learning algorithm, that is able to take advantage of user supplied feature (or attribute value) hierarchies and learn a model that is able to deal with data specified at different levels of abstraction. We also describe how a classifier can be extended to be able to decide using the decision trees built by our learner.

We evaluate the performance by presenting experimental results of our approach and comparing it to a standard decision tree learning algorithm. We end with a summary and a brief discussion.

4.1 Learning Decision Trees from Data

Decision tree learning is a supervised classification learning approach and is widely known as a purely statistical machine learning method. The induced tree is learned from a training data set and once constructed can be used to classify, previously unseen, unlabeled instances.

Consider that each attribute $A_i$ assumes a value from a finite set of values $F(A_i)$. We call these values features. An instance $X_p$ to be classified is a tuple of features, i.e., $(f_{1p}, f_{2p}, \ldots, f_{np})$ such that each $f_{ip} \in F(A_i)$. An instance in the training set is, additionally, labeled with a class $c_j \in C$. A decision tree learner algorithm starts with the original set of labeled instances $T$ as the root node and on each iteration calculates the information gain $I_E(A_i)$ for each attribute $A_i \in T$. The attributed $A_i$ with the largest information gain is then used to split $T$ in various partitions, each corresponding to a feature $f_i \in F(A_i)$ and containing the instances of the training set $T$ where the value of the attribute $A_i$ is the feature $f_i$. The attribute $A_i$ is not present in these partitions. The algorithm continues recursively on each subset until every element belongs to the same class; there are no attributes left; or there are no training instances left.
The information gain $I_E(A_i)$ of an attribute $A_i$ is calculated as follows:

$$I_E(A_i) = H(T) - \sum_{f\in F(A_i)} p(f)H(t_{A_i=f})$$

(4.1)

where $H(T)$ is the entropy of the training set and $H(t_{A_i=f})$ is the entropy of a subset of the training set formed by the instances of $T$ where the value of attribute $A_i$ is $f$. The entropy of a set $T_i$ is given by:

$$H(T_i) = -\sum_{c_j\in C} p(c_j) \log_2 p(c_j)$$

(4.2)

Not all decision tree learning algorithms use the same metric. For instance, CART [Breiman, 1993] uses Gini impurity instead of Information gain. As it measures the impurity of the set, the attribute minimizing this value should be chosen, instead of the one maximizing the metric as it is done when using Information gain. Gini index can be calculated as follows:

$$I_G(A_i) = \sum_{f_j\in F(A_i)} \frac{t_{A_i=f_j}G(t_{A_i=f_j})}{t_{A_i}}$$

(4.3)

where $t_{A_i}$ is the entire subset and $t_{A_i=f_j}$ is the fraction of this subset where the attribute $A_i$ assumes the value $f_j$ and $G(A_i)$ can be computed as follows:

$$G(T_i) = 1 - \sum_{c_j\in C} p(c_j)^2$$

(4.4)

Although these two metrics or split criteria are the most widely known, many others do exist. In section 4.2 we will show the need and propose other metric to deal with attribute values at different levels of abstraction.

Despite the existence of various different split criteria, traditional decision tree learners are, at heart, very similar. Algorithm 1 is a simple implementation of a decision tree learner. It is interesting to observe in line 9 that at each recursive step a single decoupled function is responsible for picking the best attribute and thus has an enormous influence in the shape of the tree being built.

It has been shown [Clarke, 1992] that a decision tree can be seen as a probability model for the training data. The entropy based selection criterion essentially maximizes the deviance for the probability of the model being built. As such, the decision tree learning process using an entropy impurity measurement can be seen as making a maximum likelihood estimation over the training set.

It is not uncommon for decision tree learning algorithms to construct models that violate Ockham’s razor principle of parsimony or succinctness, i.e., that build larger and more complicated trees than those needed to explain the true underlying relation between input and output attributes. This is undesirable both because it makes the tree harder to understand and because it is usually an indication that the model is overfitted to the training set. Although various decision tree pruning techniques exist that can be applied to an already built tree to help alleviate these problems with varying degrees of effectiveness, we believe that it is possible, in some extent, to avoid building an overcomplicated tree in the first place by making use of existing domain knowledge.

Our other motivation is the fact that in real world applications it is often the case that it is impossible or too expensive to always collect the exact attribute value for all instances and all attributes. Most existing approaches can deal with missing attribute values, e.g., by replacing missing values according to some probability distribution when this is known or can be determined from existing data; by grouping missing values under a new “unknown” attribute value or even by training a model with
Algorithm 1 A generic standard algorithm for the induction of decision trees

1 function BuildDecisionTree(instances, attributes, target)
2 if all instances have some target attribute value \( v \) then
3 return node(\( v \))
4 end if
5 if attributes = \( \emptyset \) then
6 \( v \leftarrow \) mostCommonValue(instances, target)
7 return node(\( v \))
8 end if
9 \( A \leftarrow \) best(instances, attributes, target) \>
\( \triangleright \) Pick best attribute according to some metric e.g. \( I_E \)
10 root \( \leftarrow \) node(\( A \))
11 for all \( v \in \text{values}(A) \) do
12 \( b \leftarrow \text{addBranch}(\text{root}, v) \)
13 instances_\( v \) \( \leftarrow \) \( \emptyset \)
14 for all \( i \in \text{instances} \) do
15 if value(\( i, A \)) = \( v \) then
16 instances_\( v \) \( \leftarrow \) instances_\( v \) \( \cup \) \( i \)
17 end if
18 end for
19 if instances_\( v \) = \( \emptyset \) then
20 \( v \leftarrow \) mostCommonValue(instances, target)
21 addNode(\( b, \) node(\( v \))
22 else
23 addNode(\( b, \) BuildDecisionTree(instances_\( v \), attributes \(-\) \( A \), target)
24 end if
25 end for
26 return root
27 end function

existing values and trying to then use it to predict missing ones. Nonetheless, we believe that when
the exact attribute value is not known but a less accurate, more abstract version of it is available, we
can leverage domain knowledge to build a tree that makes use of all available information.

Given this context, we propose HDTL, an extension of standard decision tree learning algorithms
for building decision tree models from a training set with attribute values possibly at different levels
of abstraction and from a set of OWL 2 axioms that establish a concept hierarchy. We will also describe
a classifier that given a model produced by our learning algorithm, can classify new instances with
attribute values defined at different levels of detail.

4.2 HDTL: Hierarchy Based Decision Tree Learner

In everyday life we constantly use, without thinking, inductive and deductive reasoning to learn about
the world and to make decisions based on what we know or what we think we know.

A child, after seeing a reasonably large number of dogs from different races bark concludes by
inductive reasoning that, in general dogs bark. If later presented with a dog, from a previously unseen race, chances are that the child will predict, by deductive reasoning, that this animal barks, because it is a dog, and in general, dogs bark.

Usually when learning something we try to generalize, i.e., find the largest set of elements to which what we have learned applies. It allows us to construct simpler, more compact mental models that can later be applied to a greater range of situations. When making decisions, we frequently start from general, sometimes abstract, rules and work our way down to the specific situation at hand.

To incorporate these ideas in decision trees, both the learner and the classifier have to be modified. The learner has to build a model that balances between choosing more abstract attribute values and maximizing information gain. The classifier has to be able to use this more abstract model and still classify instances composed by concrete features.

**Definition 4.2.1 (Hierarchy Based Decision Tree Learner).** An hierarchy based decision tree learner is an algorithm that given a set of labeled instances, also known as a training set, and a set of axioms describing a concept hierarchy, a decision tree model where each node is an attribute at a possible level of abstraction, each branch is a feature of that attribute at that same level of abstraction and each leaf is the class being predicted.

### 4.3 Representing Feature Hierarchies

Let \( A = \{A_1, \ldots, A_n\} \) be the set of attributes and \( F(A_i) \) denote the set of features of attribute \( A_i \). We call features to the values of the domain of the attribute \( A_i \).

Let \( H_i \) be a feature hierarchy defined over the set of features \( F(A_i) \). This hierarchy forms a tree where each leaf is a concrete feature and the other nodes are abstractions of these concrete features. Each node may have a maximum of one parent. Figure 4.1 shows an example of a feature hierarchy over the set of possible values for the attribute odor.

We further define an operation \( \text{depth}(H_i, f_a(A_i)) \) that returns the length of the path from feature \( f_a(A_i) \) to the concrete version of itself. We call this the level of abstraction of feature \( f_a(A_i) \). OWL 2 provides two axioms that allow the construction of this kind of hierarchies, \( \text{SubClassOf}(CE_1, CE_2) \) and \( \text{SuperClassOf}(CE_2, CE_1) \). Either of them allows for the construction of hierarchies and we support both. However we impose two restrictions in the way these axioms are used. First, only entities may be used meaning that complex class expressions are not allowed. Second one might notice that these axioms impose a subsumption relation between \( CE_1 \) and \( CE_2 \) such that \( CE_1 \sqsubseteq CE_2 \). So, if \( CE_1 \) holds, then \( CE_2 \) also holds. If not, \( CE_2 \) may or may not be true. This happens to be the same as
CE₁ \iff CE₂. So in OWL 2 is possible to assert that two concepts or classes are equivalent by declaring \texttt{SubClassOf}(CE₁, CE₂) and \texttt{SuperClassOf}(CE₁, CE₂), i.e., \( CE₁ \implies CE₂ \land CE₂ \implies CE₁ \) or \( CE₁ \iff CE₂ \). This is the same as the \texttt{EquivalentTo}(CE₁, CE₂) axiom. However this kind of construct requires some form of reasoning to take place and as such is not supported in our first approach. We give up on these extra expressive power to be able to process the ontology in linear time.

![Figure 4.2: Example of a hierarchy over a set of possible values for the attribute odor where the concrete feature None has no level of abstraction and as such is carried.](image)

We do not, however, require every feature to have the same number of levels of abstraction. If some concrete feature \( f₁ \in Aᵢ \) has less levels of abstraction than other feature \( f₂ \in Aᵢ \) then the highest level of abstraction of \( f₁ \) is carried to fill the higher levels of abstraction for which no abstraction exists. As such the attribute \( Aᵢ \) itself will have as many abstractions as the feature \( f_j \in Aᵢ \) such that \( f_j \) is the feature with the most levels of abstraction in \( Aᵢ \). Figure 4.2 shows an example of a feature hierarchy where the concrete value \( \text{None} \) has no higher levels of abstraction and as such is carried up. As such, the first level of abstraction for attribute odor is composed by the features \( \text{Bad}, \text{Pleasant} \) and \( \text{None} \).

### 4.4 Attribute Selection Criteria

The criterion that determines which attribute is picked at each step of the tree building process ends up determining much of the tree structure and consequently its complexity, and can boost or hinder the accuracy of the model.

Intuition, and the ockham’s razor principle of parsimony, would make one prefer attributes with less possible values as these reduce the branching factor and provide a simpler explanation of the data (as long, of course, they are enough to explain the data as well or almost as well as attributes with more possible values). Unfourtunately this is not the case with the popular metrics we introduced in section 4.1. These metrics are biased towards attributes with more attribute values [White and Liu, 1994]. As an attribute at a higher level of abstraction will always have fewer attribute values than its more concrete counterpart, this characteristic is highly undesirable in a metric for our approach.

**Proposition 4.4.1.** Given an attribute \( Aᵢ \) and an attribute \( Aᵢ₊₁ \) where at least one feature of \( Aᵢ₊₁ \) is at a higher level of abstraction and all others are at least at the same level, i.e., \( Aᵢ₊₁ \) is a parent of \( Aᵢ \), then \( I_E(Aᵢ) \geq I_E(Aᵢ₊₁) \).

**Proof.** The case where exactly one feature from \( Aᵢ \) appears in \( Aᵢ₊₁ \) at an higher level of abstraction and all other remain the same is a mere renaming of one feature in pratical terms and is trivial to observe that no counts change because of it and consequently, in this case, \( I_E(Aᵢ) = I_E(Aᵢ₊₁) \).
Now consider the case where \( n \) features \( f_1, \ldots, f_n \) from \( A_i \) are represented by a common ancestor \( f_a \) in \( A_{1i} \) and all other features remain the same. Equivalently we might say that \( A_i \) can be obtained from \( A_{1i} \) by splitting \( f_a \) in \( n \) features. This is exactly the case where it has been shown by Quinlan [1986], White and Liu [1994] that the information gain of the attribute with more features is greater than or equal to the attribute with less features even if the features of the later are already sufficiently fine for the induction task at hand.

As we climb up in the feature hierarchy, more features will be aggregated and the attribute representing that level of abstraction will consequently have less possible values. By using information gain this attribute will never be preferred. It has also been shown that Gini index only disagrees with Information Gain in about 2% of the cases [Raileanu and Stoffel, 2004]. We implemented both metrics and in practice they behaved as poorly as the theory led us to believe.

As an example, consider that we affectionately name each mushroom in our data set with such care that no two mushrooms share the same name. Our choice of names although affectionate is ultimately random and has no significant correlation with any other attribute, including the one we are trying to predict. Despite this, information gain will pick this attribute to the root of the tree against intuition and good sense.

The rationale is that as the names do not repeat they unequivocally identify every mushroom in the training set. As such each name is either of a poisonous mushroom or of an edible one but never both. In other words, once the name is known one can tell if the mushroom is edible or poisonous. However, once this model is used to classify unseen mushrooms things quickly go sideways. Either the new mushrooms have different names and cannot be classified or happen to have some repeated names but as the names are random they can be of either class, although the model will put them in the same class as the previously seen mushroom that happened to have the same name.

Although this is an extreme case it also applies to more reasonable examples. Consider for instance the attribute odor and all its nine different attribute values (not all of which are depicted in the previous illustrations). Consider that at a higher level of abstraction they can be grouped under two values, a pleasant smell and a bad smell. If this new attribute has the same predictive power as the attribute odor but uses less different features it stands to reason that it should be preferred. First, because it gives rise to a simpler tree; second, because the model is more statistically significant. In a small training set we can only have few observations of each odor and although they might explain the class we are trying to predict perfectly in the training set it may happen by pure chance. If they are grouped under two values, pleasant and bad smell, there will be more observations for each possible value and the probability that the correlations happens by pure chance is reduced. Third, the model becomes more robust. If a new odor appears in some future instance that we wish to classify, as long as we know if it is a pleasant or bad smell, we can still predict its class, even though that specific odor never appeared in the training set.

For our approach to work, we need a metric that effectively picks more abstract attributes in the place of concrete ones when doing so allows a simpler model to be built, that also explains the data. A first approach might be to pick the attribute with fewer possible values whenever there is a tie. However, there are a significant number of cases where this is not enough. For instance, when the training set is not very large and there is an attribute with a lot of attribute values it is very likely that each possible value has only a very limited set of observations and, that by pure chance, these observations happen to belong to the same class. In these cases it is likely that the more concrete
attribute has a greater score than the more abstract one even though it is not more likely to be a good predictor. However, we cannot blindly choose the attribute with fewer values. While some abstractions may help build simpler and more accurate models, others are simply too generic to be good predictors.

What we would like is to still calculate the score but penalize attributes with a lot of distinct values. In this way an attribute with a slightly worse score may still be picked if it achieves such a score with significantly less different values. However if the score is too bad to begin with, it will not be picked. This is the idea behind Gain Ratio \cite{Quinlan1986}:

$$I_R = \frac{I_E(A_i)}{-\sum_{f \in F(A_i)} p(f) \log_2 p(f)}$$

(4.5)

Although Gain Ratio is still a bit biased towards attributes with more values it is clearly less so than Information Gain or Gini Index, and unlike these will pick the more abstract attributes at least in the more obvious cases.

Now, that we have added extra dimensions (or attributes) corresponding to the different levels of abstraction available for each variable and have a metric that will pick the best attribute at best level of abstraction, we are able to build a model where each node is at the right level of abstraction. However, the classifier will need to be modified to be able to use such a model.

### 4.5 HDTC: Hierarchy Based Decision Tree Classifier

Commonly the distinction between learner, model and classifier is somewhat nebulous. Once the model is built the nodes in the tree correspond directly to the attributes of every new instance that one might have to classify and it is common to call the model itself, a classifier.

In our approach it is convenient to make the distinction clear because the decision tree model itself may not hold all the information needed to classify all the instances that we can. This happens because each node represents an attribute at a certain level of abstraction but we do not require the instances we wish to classify to have these attributes at the same level of abstraction. They can be formed only by concrete attribute values or by a mix between concrete and abstract features.
Algorithm 2 A hierarchy based decision tree classifier

```plaintext
1 function Classify(instance, decisionTree, domainKnowledge)
2     r ← root(decisionTree)
3     if children(r) = ∅ then
4         return r
5     end if
6     f ← feature(instance, r)
7     if f = ∅ then
8         f ← feature(instance, concrete(r, domainKnowledge))
9         while f /∈ features(r) do
10            f ← superClassOf(f, domainKnowledge)
11         end while
12     end if
13     subTree ← child(r, f)  \Tree rooted by the node obtained by following the branch f of node r.
14     return Classify(instance, subTree, domainKnowledge)
15 end function
```

Figure 4.3 shows the various components (learner, model and classifier) and how they interact with each other in our hierarchy based approach.

Definition 4.5.1 (Hierarchy Based Decision Tree Model). A hierarchy based decision tree is a model produced by a decision tree learner that is aware of the different levels of abstraction that may exist for some or all attributes. In this model, each node represents an attribute at a certain level of abstraction and each branch of these nodes represent the possible attribute values at that level of abstraction. The leafs of the tree represent the class being predicted.

Definition 4.5.2 (Hierarchy Based Decision Tree Classifier). Given a hierarchy based decision tree model, some domain knowledge and an instance to be classified, a hierarchy based decision tree classifier will start at the root node and if the node is at a higher level of abstraction than the instance’s corresponding attribute value, it will use the domain knowledge to climb up the abstraction tree until the right level of abstraction is reached. It will then follow the appropriate branch. This is done recursively until a leaf is reached at which point the class is returned.

The intuition is the following: when the learner tries to build a model it attempts to build the more generic model that still explains the concrete data observed in the training set. Doing so produces a more robust classifier as explained in section 4.2 that is able to better predict a wider range of possible new instances. However, as the model is now more generic than the instances, we need a way to apply the abstract knowledge embedded in the model to the particular instances we want to classify. To do so we find, for each feature, the level of abstraction that matches the relevant node of the model, using the available domain knowledge. Note that any particular instance to be classified may have a mix of concrete and abstract attribute values as long the model is generic enough to support the level of abstraction of all features, i.e., the level of abstraction of the features must be equal or lower than the corresponding decision tree nodes. As an example, if a certain node of a decision tree uses the smell of a mushroom and consequently has two branches, PleasantSmell and BadSmell, then we can classify instances that either specify the exact odor (like creosote, musty, and so on) or just the smell (pleasant
or bad). However, if instead of the smell the decision tree node uses the exact odor then we can only classify instances that specify the exact odor.

### 4.6 Results

In order to execute some experiments and compare the performance of the proposed algorithm with the standard ID3 and C4.5 decision tree algorithms a Java implementation was developed, as part of the D2PM framework [Antunes, 2011]. The standard ID3 version used to compare was written by the authors and results were compared against Weka’s implementation [Hall et al., 2009] to ensure no mistakes were made. The C4.5 implementation used was the J48 implementation in the Weka library with pruning and subtree raising enabled.

In spite of data with values specified at different levels of abstraction being common in many domains of application there are few standard benchmark data sets with these characteristics and with an associated ontology. We selected the Mushroom and the Nursery data set from the UCI Machine Learning Repository [Bache and Lichman, 2013].

The nursery data set corresponds to 12960 observations with 8 attributes and a target attribute with 5 possible values. Three of these five classes dominate the data set, with each having about 32% of the universe of instances. The two remaining classes are represented by fewer than 3% of all instances.

The mushroom data set includes descriptions of 8124 samples corresponding to 23 species of gilled mushrooms in the Agaricus and Lepiota family (although no information is present about the species of each observation). There are 22 attributes and the target attribute has two possible values: poisonous or edible. The observations are nearly evenly distributed between these two classes.

Domain knowledge obtained from the book “The Mushroom Hunter’s Field Guide” and from [Zhang et al., 2002] was made explicit in an OWL 2 ontology.

Three sets of experiments were then executed. The first compares the accuracy of the proposed Hierarchy Decision Tree with the standard ID3 and C4.5 algorithm on the original data, where all values are concrete. We also look at the complexity of the produced decision trees. Figure 4.5 shows an example of a decision tree generated by HDT for randomly selected small subsets (about 50 instances) of the data set as training sets. This simple tree has an accuracy over the entire data set of 0.914 while the standard ID3 algorithm, for the same training set, generates a tree that has an accuracy of only 0.549, almost as bad as randomly picking a class.

The second set of tests shows how the accuracy of all algorithms evolve with the size of the training...
sets. A subset with 1000 instances was randomly selected from the original data set to serve as a test set. Six subsets were randomly selected from the remaining instances of the original data set, with sizes of 700, 300, 70, 50, 20 and 15 instances to be used as training sets.

To assess the accuracy of the three algorithms, we used cross-validation by repeated random sub-sampling. Five disjoint subsets were randomly selected and each was divided in two disjoint subsets, a training set and a test set. We display the mean accuracies.

The results obtained (displayed in Figure 4.6 and Figure 4.7) show that our approach outperforms both ID3 and C4.5 in all the tested subsets of both data sets. The difference is more pronounced in the smaller training sets, becoming less noticeable as the size of the training set increases.

On smaller training sets it is more likely that not all possible attribute values are present. As HDT tries to build a more general model, with some nodes corresponding to abstract attributes, it is still able to predict the class of instances containing features that were not present in the training set, while ID3 and C4.5 fail. When the size of the training set grows and all attribute values become present, ID3 and C4.5 catch up.

The last set of tests studies how the accuracy of the algorithms changes with an increasing number of values being abstract, e.g., not knowing the exact odor of a mushroom but being able to tell if it has a pleasant or bad smell. Starting from a data set with no abstract values, six data sets were then generated with an approximate percentage of abstract attribute values of 5%, 10%, 15%, 20%, 25% and 50%.
Figure 4.7: The influence of training set size on the accuracy of ID3, C4.5 and HDT in the Nursery data set.

The results of these tests show that HDT is able to maintain its accuracy better than ID3 and C4.5 when, instead of the exact attribute values, only more abstract versions of the features are available. This serves as evidence that HDT is able to make use of feature hierarchies to build more robust models that maintain good predictive power when some information exists about the attribute value, but is insufficient to exactly determine its value. Figure 4.8 shows these results.

HDT’s ability to maintain its predictive power in the face of less precise attribute values depends on the quality of the available feature hierarchies and, in some extent, on the nature of the classification problem. For some classification tasks it might very well be the case that for some attributes, the exact value is needed to predict the right class. In these cases HDT will pick the concrete version of the attribute when building the model but the robustness of such a model is negatively affected, although still better than traditional approaches. Figure 4.9 shows this.

These results are in line with our expectations. First, even on data where all values are concrete, domain knowledge can help build models that perform as good or better while being considerably simpler. This difference in accuracy is more pronounced with smaller training sets.

Second, when the specific concrete values are unknown but a more abstract version is available, HDT maintains its performance remarkably well while the performance of traditional ID3 and C4.5 decreases as more values are expressed at higher levels of abstraction.
Figure 4.8: Accuracy of ID3, C4.5 and HDT in the Mushrooms data set with an increasing number of abstract values.

Figure 4.9: Accuracy of ID3, C4.5 and HDT in the Nursery data set with an increasing number of abstract values.
4.7 Summary and Discussion

We proposed an approach that makes use of some existing domain knowledge, in the form of a hierarchy of attribute values, to build and use a more robust and simpler decision tree model. We made the case that by using more abstract values in some nodes of the decision tree we are able to better deal with new unseen instances, even with those where some attribute values were never seen before.

We reviewed some popular attribute selection criteria and showed why they are inept to be used in an approach where we have to choose the right level of abstraction for an attribute, in addition to picking the best attribute. In this case, the bias that Information Gain and Gini Index have towards attributes with more distinct values severely hinders their ability to pick attributes at higher levels of abstraction even when doing so seems obvious. We look instead at Gain Ratio, an attribute selection metric based on Information Gain but that penalizes attributes with a lot of different values, making it less biased and giving a chance for attributes at higher levels of abstraction to be picked.

In our approach the learner, model and classifier are three distinct entities and we make that distinction clear. In traditional approaches a decision tree model can be almost directly applied to a new instance when trying to predict its class. This happens because there is a direct match between the features in the instance and the branches in the decision tree. As we wish to classify instances that may not have all features at the same level of abstraction of the corresponding nodes and branches in the model we need a classifier that can use the available domain knowledge to climb up in each feature hierarchy until a match exists and a branch can be chosen. We propose and describe such a classifier. It allows for applying more generic models to concrete observations and still be able to classify them.

The results show that the method we propose is able to perform considerably better than traditional classifiers even with small training sets, achieving levels of performance that require more traditional approaches to be trained with much larger sets. They also show that as we decrease the percentage of concrete features, replacing them with less precise (more abstract) ones our method is able to maintain its performance while the accuracy of traditional approaches decreases significantly.

Despite these strengths this method still suffers from the very limited range of axioms (SubClassOf and SuperClassOf) that are supported. Although these are enough to build hierarchies, they are clearly not enough to define more interesting rules that would allow us to define new dimensions from existing attributes, e.g., all mushrooms that have a bad smell and white spores are of the species false parasol.
Chapter 5

Hierarchy-based Naïve Bayes Learner and Classifier

We start this chapter with a review of the standard naïve bayes approach, a purely statistical classification method. We note that, just like with decision trees, it is possible to identify three distinct but connected components: a learner, that given a set of labeled instances produces a model; the model itself, which is a class conditional probability table from which the most probable class can be computed; and a classifier which uses the model built to classify previously unseen instances.

We extend these components and present HNB, a hierarchy aware naïve bayes learning algorithm, that is able to take advantage of user supplied feature (or attribute value) hierarchies and learn from data specified at different levels of abstraction. We also describe how a classifier can be extended to be able decide using a three dimensional probability table built by our learner, where the third dimension holds a layer for each possible level of abstraction and the other two hold the class and the possible attribute values, as usual.

We evaluate the performance by presenting experimental results of our approach and comparing it to a standard naïve bayes learning algorithm. We end with a summary and a brief discussion.

5.1 Building a Probabilistic Model from Data

The Bayes theorem allows for the computation of the probability of the occurrence of some particular event given some observations. Bayesian inference derives the posterior probability as a consequence of two antecedents, a prior probability and a likelihood function that can be determined from a probability model for the observed data. Bayesian inference computes the posterior probability (or hypothesis in the context of classification) according to Bayes’ rule:

\[ P(h|T) = \frac{P(T|h)P(h)}{P(T)} \]  

(5.1)

where \( h \) is the hypothesis being tested, and \( T \) is some labeled observation. Consequently \( P(T|h) \) is the likelihood of the observation \( T \) given the hypothesis \( h \). \( P(h) \) is the prior probability of \( h \), such that \( \sum_{h \in H} P(h) = 1 \) and \( \forall h \in H : P(h) \geq 0 \).

Bayesian learning is the process of learning the likelihood distribution \( P(T|h) \) and the prior probability distribution \( P(h) \) from a set of instances labeled with \( h_1, h_2, \ldots, h_n \) known as the training set. In
other words we wish to produce a maximum a posteriori hypothesis $h$ that maximizes $P(h|D)$, i.e.,

$$h_{\text{MAP}} = \arg \max_{h \in H} P(h|T) = P(h)P(T|h)$$ (5.2)

Naïve Bayes is, then, a supervised classification learning approach and is widely known as a purely statistical machine learning method. The probability model is calculated from a training data set and once constructed can be used to classify previously unseen, unlabeled instances. Although it makes the naïve assumption that the attribute values of an instance are conditionally independent given the class, it has been shown that accuracy is only very slightly affected by this \[Zhang, 2004\].

The goal of the learner is to calculate class conditional probabilities $p(f_k|c_j)$ and class probabilities $p(c_j)$ for every $c_j \in C$ and $f_k \in F(A_i)$ from the provided training set $T$. These probabilities are sufficient to completely specify a Naïve Bayes model and can be stored in $n + 1$ tables, where $n$ is the number of attributes. One of the tables has a single dimension and holds $p(c_j)$ for all $c_j \in C$. The other $n$ tables have two dimensions, one for all $c_j \in C$ and other for all $f_{ik} \in F(A_i)$. Each cell of these $n$ tables holds a class conditional probability $p(f_{ik}|c_j)$ with $1 \leq k \leq |F(A_i)|$.

Algorithm 3 is a simple implementation of a naïve bayes learner.

As in the previous chapter, our motivation is the fact that in real world applications it is often the case that it is impossible or too expensive to always collect the exact attribute value for all instances and all attributes. Most existing approaches can deal with missing attribute values, e.g., by replacing missing values according to some probability distribution when this is known or can be determined from existing data; by grouping missing values under a new “unknown” attribute value or even by training a model with existing values and trying to then use it to predict missing ones. Nonetheless, we believe that when the exact attribute value is not known but a less accurate, more abstract version of it is available, we can leverage domain knowledge to build a probability model that makes use of all available information.
Given this context, we propose HNB, an extension of the standard naïve bayes algorithm for building probability models from a training set with attribute values possibly at different levels of abstraction and from a set of OWL 2 axioms that establish a concept hierarchy. We will also describe a classifier that given a model produced by our learning algorithm can classify new instances with attribute values defined at different levels of detail.

5.2 HNBL: Hierarchy Based Naïve Bayes Learner

We now introduce HNBL, an algorithm for learning multi-layered probability tables from a feature hierarchy and a training set. The layers represent the different levels of abstraction for the features of each attribute.

Consider that each attribute \( A_i \) assumes a value from a finite set of values \( F(A_i) \). We call these values features. An instance \( I_p \) to be classified is a tuple of features, i.e., \( (f_{1p}, f_{2p}, \cdots, f_{np}) \) such that each \( f_{ip} \in F(A_i) \). Given a set of hierarchies \( H = H_1, \cdots, H_n \) and a training set \( T = (I_p, c_I) \) where \( c_I \in C \) and is known to be the class of the instance \( I_p \), the goal of the learner is to compute, for each level of abstraction \( l \), for each feature \( f_{ki} \), and for each class \( c \) the conditional class probability \( p(f_{ki}|c) \) and class probabilities \( p(c_j) \) for every \( c_j \in C \).

The traditional version presented in section 5.1 requires \( n+1 \) tables, where \( n \) is the number of attributes. One of the tables has a single dimension and holds \( p(c_j) \) for all \( c_j \in C \). The other \( n \) have two dimensions, one for all \( c_j \in C \) and other for all \( f_{ki} \in F(A_i) \). Each position of these tables hold a class conditional probability \( p(f_{ki}|c_j) \).

To be able to deal with features at multiple levels of abstraction the following changes are needed:

- The first table stays unchanged, holding \( p(c_j) \) for all \( c_j \in C \).
- The other \( n \) tables will have an extra dimension. The size of this dimension will be the number of levels of abstraction for the feature \( f_{ki} \in F(A_i) \) in accordance to its corresponding feature hierarchy \( H_i \).
- Each position of these multi-layered tables will hold the class conditional probability of a given feature at a certain level of abstraction \( l \) divided by the probability of feature \( f_{ki} \) occurring in the training set, i.e., \( p(f_{ki}|c_j) / p(f_{ki}) \) for the \( k \)th feature of attribute \( A_i \) at \( l \)th level of abstraction. Given Equation 5.3 one might wonder why the denominator \( p(f_{ki}) \) is needed. This will be clear when we show the modified version of the classifier in section 5.3. To calculate the probabilities to fill in each position of the table, the following steps are needed:

1. Starting from the concrete features fill each position of the first layer with \( \#(f_{ki}|c_j) \), i.e., the count of instances, in the subset of the training set that are of the class \( c_j \), where the value of attribute \( A_i \) is \( f_{ki} \).
2. Work our way up by aggregating the counts, e.g., for a feature at the first level of abstraction, sum the counts of all features that are its children. They will all be concrete and as such will have been counted directly; for a feature at the second level of abstraction, its children will all be the first level of abstraction and will have been counted previously.
3. Once we have all counts, we loop through each position of the table and calculate \( p(f_{ki}|c_j) / p(f_{ki}) \) based on the counts of that position. Note that aggregating the probabilities directly, instead of the counts will not work because of the denominator.
For each attribute it is possible that some features have more levels of abstraction than others in which case the number of layers for the attribute will be determined by the feature with the largest number of levels of abstraction. The features with fewer levels will simply have their highest level of abstraction propagated up.

Note that contrary to what happens with HDTL, described in section 4.2, where the model already embodies a decision about the best level of abstraction for each attribute, here no such decision is made while building the model, as this choice is delegated to the classifier. While this produces a larger, more complex model it allows the instances to be classified to have features at any level of abstraction.

5.3 HNBC: Hierarchy Based Naïve Bayes Classifier

We propose a variation of the Naïve Bayes classifier that makes use of the data structures described to choose the highest level of abstraction, for the features $f_{ij}$ of the attributes $A_i$ of the given data set, where $p(f_{ij}|c_k)$ is maximized for each $c_k \in C$.

**Proposition 5.3.1.** Given the conditional independence assumption, maximizing for a class $c_j$, $P(c_j) \prod_{i=0}^{n} \frac{p(f_{ij}|c_j)}{p(f_{ij})}$ can be done by selecting the maximum values for each $\frac{p(f_{ij}|c_j)}{p(f_{ij})}$ individually.

*Proof.* Given the conditional independence assumption, the choice of feature for a given attribute $A_i$, or equivalently, the choice of level of abstraction for a given feature of attribute $A_i$ bears no influence on the class conditional probability of features of any other attribute $A_j, j \neq i$. Therefore maximizing the product consists in maximizing each term of the product.

When, for a given $f_{ijl} \in F(A_i)$ and $c_j \in C$ there is a $f_{ij(l+1)} \in F(A_i)$ for the same $c_j \in C$ such that $P(f_{ij(l+1)}|c_j) \geq P(f_{ijl}|c_j)$ then the more abstract feature $ij(l+1)$ will be chosen instead of the more concrete feature $ijl$. Equation 5.3 makes the assumption that in Equation 5.3 for any class $c_j$ the denominator of the product will always be the same and as such can be removed without loss of order. This holds because for any given instance $I_p$ we are only trying to choose the best class. As such when computing Equation 5.3 we end up with a pool of hypotheses, one for each $c_j \in C$. The only thing changing from one hypothesis to the next is $c_j$ which does not affect the denominator $p(f_{ijp})$.

However when we also consider features at different levels of abstraction we end up with a larger pool, one in which both $c_j$ and $f_{ijp}$ can change. Consequently we end up with different denominators among hypotheses as the probability for a bad smell is certainly larger than the individual probabilities of any given odor that we consider a bad smell.

The fact that we can no longer ignore the denominator forces us to make another naïve assumption in order to be able to efficiently compute the denominator and that is the assumption that the attributes are independent.

**Proposition 5.3.2.** Given an instance $I_p$ composed by a set of features $f_{1p}, \cdots, f_{np}$ possibly at different levels of abstraction the most probable class can be given by the following equation assuming that all attributes are independent and conditionally independent given the class:

$$\arg\max_{c_j \in C} p(c_j) \prod_{i} \frac{p(f_{ijp}|c_j)}{p(f_{ijp})}$$

(5.3)
Proof. We wish to know what class $c$ maximizes $p(c|f_1, \ldots, f_n)$.

$$p(c|f_1, \ldots, f_n) = \frac{p(c)p(f_1, \ldots, f_n|c)}{p(f_1, \ldots, f_n)} = \frac{p(c)p(f_1|c)p(f_2, \ldots, f_n|c, f_1)}{p(f_1)p(f_2, \ldots, f_n|f_1)} = \frac{p(c)p(f_1|c)p(f_2, c, f_1)p(f_3, \ldots, f_n|f_1, f_2)}{p(f_1)p(f_2|f_1)p(f_3, \ldots, f_n|f_1, f_2)} = \frac{p(c)p(f_1|c)p(f_2|c, f_1) \ldots p(f_n|c, f_1, \ldots, f_{n-1})}{p(f_1)p(f_2|f_1) \ldots p(f_n|f_1, f_2)} = \frac{p(c)p(f_1|c)p(f_2|c) \ldots p(f_n|c)}{p(f_1)p(f_2) \ldots p(f_n)} = p(c) \prod_{i=0}^n \frac{p(f_i|c)}{p(f_i)}$$

Note that we use the chain rule of probability, the independence assumption of Naïve Bayes and the assumption that all attributes are independent.

As features can have multiple levels of abstraction, given a class $c_j \in C$ and a feature $f_{in} \in F(A_i)$ we still have to pick the right level of abstraction. To do this, go to the position of the table for the feature $f_{in}$ and the class $c_j$. That position will contain a list where each value corresponds to $p(f_{in}|c_j)/p(f_{in})$ for a different level of abstraction of the feature $f_{in}$. We compute the maximum of this list to obtain the best level of abstraction for the current attribute.

It is important to show that the class conditional probability of a given feature can be higher or lower at different levels of abstraction, i.e., if climbing always improved the probability, then we would climb until all features converged in a single abstract feature and the model produced would be meaningless. On the other hand if climbing always produced worse probabilities we would never climb and the model produced would be equal to the traditional Naïve Bayes.

**Proposition 5.3.3.** For any given feature $f_{in} \in F(A_i)$, a more abstract version of $f_{in}$ can yield a higher or lower probability $p(f_{in}|c_j)/p(f_{in})$.

Proof. Let $f_k$ be any concrete feature and $c_j$ be the winning class, i.e., the class for which $p(f_k|c_j)/p(f_k)$ is larger than any other, with $l$ being any possible level of abstraction. We will show that using a more abstract feature can be better without always being better and can be worse without always being worse.

Let $f_a$ be an abstract feature such that $f_a = f_k \cup f_c; f_e \cap f_k = \emptyset$. We may assume, without loss of generality, that $f_c$ appears once in the training set. Thus we want to show that is possible that:

$$\frac{p(f_k|c_j)}{p(f_k)} \geq \frac{p(f_a|c_j)}{p(f_a)} \quad \frac{p(f_k|c_j)}{p(f_k)} \geq \frac{p(f_k \cup f_e|c_j)}{p(f_k \cup f_e)} \quad \frac{p(f_k|c_j)}{p(f_k)} \geq \frac{p(f_k|c_j) + p(f_e|c_j)}{p(f_k) + p(f_e)}$$

If the instance containing $f_c$ is not of the class $c_j$ then $p(f_c|c_j) = 0$ and the inequality is verified. If, on the other hand, the the instance containing the feature $f_c$ is of the class $c_j$ then we can show that the
opposite occurs:

\[
\begin{align*}
\frac{p(f_k|c_j)}{p(f_k)} & \leq \frac{p(f_a|c_j)}{p(f_a)} \\
p(f_k|c_j) & \leq \frac{p(f_k \cup f_e|c_j)}{p(f_k \cup f_e)} \\
p(f_k|c_j) & \leq \frac{p(f_k|c_j) + p(f_e|c_j)}{p(f_k) + p(f_e)} \\
\frac{\#(f_k|c_j)}{\#c_j} & \leq \frac{\#(f_k|c_j) + 1}{\#c_j} \\
\frac{\#f_k}{\sum \#c_i} & \leq \frac{\#f_k}{\sum \#c_i} \\
\#(f_k|c_j) & \leq \#f_k
\end{align*}
\]

Consider that we have an instance \(I_p = (f_1p, \ldots, f_{np})\) to classify. For each possible class \(c_j\) and for each feature \(f_{ip}\):

1. Find in what level of abstraction the feature \(f_{ip}\) was given via the feature hierarchy \(H_i\) obtained from the ontology.

2. If it is a previously seen feature go to the appropriate position in the table. Find the layer with the maximum value.

3. If the feature was not seen before, i.e., it did not appear in the training set, use the taxonomy from the ontology to find the closest ancestor. Repeat until an ancestor is found that has a corresponding layer in the table.

Algorithm 4 A hierarchy based naïve bayes classifier

```plaintext
1 function Classify(instance, cube)
2     max_p ← 0
3     class ← ∅
4     for all pC ∈ classesProb(cube) do
5         p ← pC
6         for all attribute ∈ instance do
7             p ← p × max(cube, C, attribute, value(attribute, instance))
8         end for
9         if p ≥ max_p then
10            max_p ← p
11            class ← C
12        end if
13     end for
14     return class
15 end function
```

Use Equation 5.3 to find the most probable class. The rational behind the third point is the following: consider that in the training set, for attribute \(A_i\) the following features where seen \(f_1i, \ldots, f_{(n-1)i}\) and based in \(H_i\) we know some of these features are children of an abstract class \(f_{ai}\). We will then have
a position of the table holding the class conditional probability of this abstract feature. Now consider that an instance \( I_p \) is given, containing a feature \( f_{ni} \) never seen in training set. Suppose that we know, however, from \( H_i \) that \( f_{ai} \) is an ancestor of \( f_{ni} \). We can still classify the instance \( I_p \) using the class conditional probabilities of \( f_{ai} \) in place of \( f_{ni} \), which we do not have.

## 5.4 Results

In order to execute some experiments and compare the performance of the proposed algorithm with standard Naïve Bayes a Java implementation was developed, as part of the D2PM framework [Antunes 2011]. The standard Naïve Bayes version used to compare was written by the authors and the results obtained were compared with Weka’s implementation [Hall et al. 2009] to ensure no mistakes were made in our implementation.

As in the last chapter, we use the Mushroom and the Nursery data sets from the UCI Machine Learning Repository [Bache and Lichman 2013] to test the accuracy of our approach and analyse how it compares to standard naïve Bayes.

We have also reused the feature hierarchies written in the last chapter as OWL 2 ontologies. Two sets of experiments were then executed. The first compares how the accuracy of the proposed Hierarchy based naïve Bayes and the standard naïve Bayes evolve with the size of the training sets. Six subsets were randomly selected from the original data sets, with sizes of 700, 300, 70, 50, 20 and 15 instances to be used as training sets. The remaining instances were used as part of the test set. The entire process was repeated five times and the mean accuracy was calculated.

The results obtained show that our approach outperforms standard naïve Bayes in all the tested subsets of both data sets. The difference is more significant in the smaller training sets, becoming less pronounced as the size of the training set increases.

On smaller training sets it is more likely that not all possible features are present. As HNB tries to build a more robust model, with some layers corresponding to abstract attributes, it is still able to predict the class of instances containing features that were not present in the training set, while NB treats these features as missing values. When the size of the training set grows and all attribute values become present, standard NB catches up. These results are shown in Figure 5.1 and Figure 5.2.

The second experiment studies how the accuracy of both algorithms changes with an increasing number of features being abstract, e.g., not knowing the exact odor of a mushroom but being able to tell if it has a pleasant or bad smell. Starting from a data set with no abstract features, six data sets were then generated with an approximate percentage of abstract features of 5%, 10%, 15%, 20%, 25% and 50%.

The results of these tests showed that HNB is able to maintain it’s accuracy better than standard naïve Bayes when, instead of the exact features, only more abstract attribute values are available. This supports the hypothesis that HNB is able to make use of feature hierarchies to build more robust models that maintain good predictive power when some information exists about the attribute value, but is insufficient to exactly determine it’s value. Figure 5.3 shows these results.

HNB’s ability to maintain it’s predictive power in the face of less precise attribute values depends on the quality of the available feature hierarchies and, in some extent, on the nature of the classification problem. For some classification tasks it might very well be the case that for some attributes, the exact value is essential to predict the right class. In these cases HNB will still have a layer for the
Figure 5.1: The influence of training set size on the accuracy of NB and HNB in the Mushrooms data set.

Figure 5.2: The influence of training set size on the accuracy of NB and HNB in the Nursery data set.
Figure 5.3: Accuracy of Naïve Bayes and Hierarchy based Naïve Bayes in the Mushrooms data set with an increasing number of abstract features.

more abstract version of the attribute and will still be able to make a prediction but the accuracy will be hindered by the lack of the more precise attribute value, although still better than traditional approaches. Figure 5.4 shows these results.

The results obtained are consistent with our two hypotheses. First, even on data where all features are concrete, domain knowledge can help build models that perform better when trained with very small training sets and slightly better on normally sized training sets.

Second, when the specific concrete features are unknown but a more abstract version is available, the Hierarchy based Naïve Bayes maintains its performance while the performance of traditional Naïve Bayes decreases as more features are expressed at higher levels of abstraction. These results point to HNB being a more robust approach to classification when domain knowledge is available, when compared to standard NB.
5.5 Summary and Discussion

In this chapter we proposed an approach that makes use of some existing domain knowledge, in the form of a hierarchy of attribute values.

This approach builds and uses an extension of the naïve Bayes class conditional probability table that allows features at different levels of abstraction by essentially having a layer for each level of abstraction.

We showed that removing the denominator of Equation 5.3 is no longer a viable option. In addition to assuming that attributes are conditionally independent given the class we also assume that they are independent (note that neither of these assumptions implies the other). Although both are naïve assumptions our experimental results show that in practice our method outperforms standard naïve bayes.

Our hierarchy based naïve Bayes learner and classifier perform well enough to show that the basic idea of using OWL 2 SubClassOf axioms to represent some form of domain knowledge as hierarchies can be applied to different classification algorithms, although with some fundamental differences. In decision trees our work revolved mainly around the attribute selection metric and the classifier. The divide and conquer nature of the learning algorithm translates nicely to problems where we have features at different levels of abstraction. Beyond the attribute selection criteria no special care has to be taken when some features happen to be generalizations of others. We simply add a new attribute $A_{n+1}$ that contains the features of $A_n$ at an higher level of abstraction, and let our attribute selection criteria pick among all the one that explains most of the data. At once, not only do we pick the best attribute but also the best level of abstraction. With naïve bayes things do not translate as nicely, which forces us to add an extra dimension to the model with the purpose of holding the class conditional probabilities at different levels of abstraction for each feature and class combination. Simply adding a new attribute as we do in the decision tree learning method would of course violate the assumption that attributes are independent given the class.

The approach we proposed in this chapter is enough to work around this assumption when dealing
with feature hierarchies because in these cases the dependencies are clear and exist among values of the same attribute. For any feature $f_{ip}$ in Equation 5.3 there will not be another feature $f_{jp}$ such that both belong to the same hierarchy.

But what if instead of the SubClassOf axioms we needed more complex constructs involving different attributes and giving rise to new ones? Say, for the sake of argument, that we wish to express the idea that if a mushroom has green gills and green spore print then it is of the species *molybdites*. The color of the gills and the color of the spore print are existing attributes but the species is a new one defined by the specified combination. Then we can of course define a new axiom that uses the new attribute species and some existing ones to give rise to another new attribute, creating, in the process, a very complex network of dependencies between attributes that becomes harder and harder to hide from the algorithm.
Chapter 6

Ontology-based Decision Tree Learner and Classifier

The approaches we have presented until now and, as far as the authors know, other existing approaches to introducing some form of domain knowledge in the classification process deal only with taxonomies which are a fraction of the expressive power of true ontologies.

In this chapter we present a decision tree learner and classifier that is able to make use of knowledge expressed in ontologies that go well beyond simple feature hierarchies. With greater expressive power comes greater complexity. To be able to deal with this extra complexity we will introduce an additional component in our approach with the mission of inferring logical consequences from a set of asserted domain facts expressed as axioms in the ontology. This component is commonly known as a semantic reasoner or simply as a reasoner. The logical consequences inferred by the reasoner will be considered by the learner while building the model.

We need a standard way of expressing domain knowledge, so it can be shared and reused. The Web Ontology Language, OWL 2 [Motik et al., 2009] satisfies this criterion and offers plenty expressive power to use in the context of classification. We assume that the reader is somewhat familiar with OWL 2, described in section 3.1 and with the Manchester syntax. Nonetheless, we briefly review the main components of an OWL 2 ontology.

Figure 6.1 shows how the reasoner is incorporated in our approach and how it interacts with the learner and classifier.

6.1 OWL 2

The main components of an OWL 2 ontology are axioms, classes, individuals and properties. Two types of properties exist: data properties have a literal as a range, and object properties have a class as range.

Note that classes in the ontology have nothing to do with class of the instance in a classification problem, i.e., the attribute value we are trying to predict.

Classes provide an abstraction mechanism for grouping resources with similar characteristics. When you think of the concept Parasol, for instance, you are not thinking of any concrete mushroom. Rather all the mushrooms that share the necessary characteristics to be considered of that species.
However if you embark in a mushroom hunting adventure you will probably find a mushroom of this species for dinner. That mushroom is an Individual of the class *Parasol*.

Characteristics are called properties and odor is an example of a property of the class *Parasol*.

So how can one define which individuals belong to the class *Parasol*? Using axioms. Axioms are the core of an OWL 2 ontology and are essentially statements that are truth in the domain. You can then say that mushrooms with white spore print color and not white gills are of the class *Parasol*. You can also say that *Parasol* is a subclass of *Mushroom*, i.e., all individuals in the class *Parasol* are also in the class *Mushroom*.

However reasoning in a full OWL DL ontology is a problem in NEXPTIME which is highly undesirable for our intended application.

### 6.2 OWL 2 EL

Fortunately a subset exists that trades off some aspects of the full OWL DL expressive power in return for PTIME complexity in standard reasoning tasks, namely ontology consistency, class expression subsumption and instance checking. One of the fragments of the Web Ontology Language that is known to have these properties is the OWL 2 EL profile [Hitzler et al., 2009].

Dedicated reasoning algorithms for this profile are available and have been demonstrated to be implementable in a highly scalable way [Baader et al., 2006]. The EL acronym reflects the profile’s basis in the $\mathcal{EL}$ family of description logics [Baader et al., 2005, 2008], logics that provide mainly Existential quantification of variables. Universal quantification on properties is disallowed and, therefore, propositions like “all children of a rich person are rich” cannot be directly stated in a $\mathcal{EL}$ logic. However statements like “all mushrooms are fungi” are supported. In the first example there is a class *Person*.
which has an object property hasChildren. The axiom states that all entities in the codomain of this property must be of the class Rich if the corresponding entity in the domain is also of the class Rich. The second example only states that all elements of the class Mushroom are also elements of the class Fungus which can be stated using class inclusion (SubClassOf axiom) that is allowed. Using class expressions on both sides of the SubClassOf is also permitted, so rules like “mushrooms with a bad smell and dark colours are unattractive” are valid in the EL profile.

Also valid are existential quantification restrictions to a class expression or a data range, to an individual and self-restriction; enumerations involving a single individual or a single literal and intersection of classes and data ranges.

As illustrated in the previous examples, class and object property inclusion is allowed as is class equivalence and disjointness. Property equivalence, transitive and reflexive object properties are also supported as are both domain and range restrictions. All types of assertions are allowed as long as they are formed by class expressions in EL.

As previously stated universal quantification to a class expression is not supported. Also unsupported are cardinality restrictions (not to be confused with range restrictions which are supported: range restrictions allows one to state that a teenager is a person with an age between thirteen and nineteen; cardinality restrictions would allow one to state that a person has exactly one age). Among other unsupported constructs are disjunction, class negation, irreflexive, inverse, symmetric and asymmetric object properties. As such there is no way of directly specifying that the object property childOf is the inverse of parentOf. You may have noticed that most restrictions affect properties, not classes. In section 6.4 we propose a design pattern that avoids most of these limitations by building class-expression oriented ontologies that can easily be used by classification algorithms.

6.3 ELK

Once we have a set of axioms that define class membership we will need to quickly compute which individuals belong to which classes. Although it might sound simple, it is a rather complex topic and an area of research in itself [Stoilos et al., 2010; Dentler et al., 2011; Shearer et al., 2008; Kazakov and Klinov, 2013, 2014] another hard problem hiding behind a simple formulation.

It turns out that EL is robustly tractable for the main reasoning problems, i.e., ontology classification and consistency checking, computing if a given complex concept is satisfiably and checking whether one complex concept is subsumed by another. Not only are these reasoning problems polynomially solvable for EL, but this holds even when the language is augmented with nominals (and thus ABox assertions).

We use ELK [Kazakov et al., 2014] – an open source, Java-based reasoner for OWL EL ontologies – to determine which individuals in the ontology belong to which classes. This is called ABox Realization.

Realization is the task of computing the implied instance/type relationships between all named individuals and named classes in an ontology. Only direct instance/type relations are returned in the result. In order to determine which instance/type relations are direct, one needs to know all subclass/superclass between named classes in the ontology. Therefore, ELK automatically triggers TBox classification before ABox realization.

TBox classification is the task of computing the implied subclass/superclass relationships between all named classes in an ontology. Besides finding out whether a class is subsumed by another one or
not, this task involves the transitive reduction of the computed class taxonomy: only direct subclass/superclass relations are returned in the result.

Although our implementation does use ELK, our approach does not depend on ELK specifically. Rather any reasoner that can deal with ontologies at least as expressive as OWL EL can be used instead. In truth, our approach does not depend on OWL EL either, a more expressive profile can be used instead. However in such case the polynomial time complexity can no longer be guaranteed.

EL reasoners, like ELK, have the advantage of dealing with a much simpler language. Unlike conventional tableau-based procedures [Horrocks et al., 1999], which test unknown subsumptions by attempting to construct counter-models, EL procedures derive new subsumptions explicitly making use of inference rules. Consequently the reasoner never inspects subsumptions that are not entailed by the ontology and all subsumptions can be computed at once in one pass.

6.4 Structuring an Ontology to Support Classification Problems

In the previous chapter we have looked at ontologies as a completely separated topic from the problem of learning to predict a target attribute from a set of labeled examples.

In this section we will make a bridge between the data set and the ontology, so the algorithm can leverage the available domain knowledge and the labeled instances in the data set to produce a more precise and compact model.

Attribute values in the data set that we want to use while defining axioms are added as Individuals to the ontology. Consider that we are interested in the following odors: creosote (c), fishy (y), foul (f), musty (m), pungent (p) and spicy (s) which are bad smells but not in almond (a), anise (l) or none (n). Also consider that we are only interested in green (r) spore-print-color.

OWL fragment 6.1: What mushrooms odors smell bad?

```
1 Class: BadSmell
2
3 Individual: c
4 Types: BadSmell
5 Individual: y
6 Types: BadSmell
7 Individual: f
8 Types: BadSmell
9 Individual: m
10 Types: BadSmell
11 Individual: p
12 Types: BadSmell
13 Individual: s
14 Types: BadSmell
```

OWL fragment 6.2: Green spore print colors make greenish mushrooms

```
1 Class: Greenish
2
3 Individual: r
4 Types: Greenish
```

Existing attributes (in the data set) that we wish to mention in our axioms are added as object properties. Suppose that we are interested in odor and in spore-print-color.
OWL fragment 6.3: Definition of the attributes odor and spore print color in the ontology, allowing the definition of axioms that use these attributes

1. ObjectProperty: odor
2. ObjectProperty: spore–print–color

A meta-class “Attribute” that can have two kinds of direct subclasses. New attributes have no corresponding object property and represent a new dimension in which instances in the data set can be considered. These kind of new attributes result from the application of a set of axioms to the existing attribute values or to an abstraction of them. In the next example we will add a new attribute called Species.

On the other hand, direct subclasses of the meta-class “Attribute” that have a corresponding object property represent attributes that already exist in the data set but will have multiple levels of abstraction. Each direct subclass of one of these attributes represent a new level of abstraction to be considered. In the next example we will add a higher level of abstraction to the attribute odor, called smell. The subclasses of smell are the possible attribute values of the new attribute smell.

OWL fragment 6.4: Attribute hierarchy showing a new class species and an higher level of abstraction smell for the attribute odor

1. Class: Odor
2. Class: Smell
3. SubClassOf: Odor
4. Class: Species
5. Class: Attribute
6. SuperClassOf: Odor
7. SuperClassOf: Species

Note that the subclasses of smell and species are the possible attribute values. We can have how many attributes values we want. However note that it is possible that some instances are not part if any of these attribute values because they are not part of any of the corresponding classes. As an example, consider any instance where the attribute value of the attribute odor is anise. This instance is not part of the BadSmell class and there are no other subclasses of odor. When this happens the attribute in question will have a new special attribute value “NA” that will have all instances that do not belong to any attribute value. One might be tempted to define a new class GoodSmell as the negation of the class BadSmell. This is a violation of the EL profile as it does not support class negation.

At last, suppose that we know that if some mushroom smells bad or has greenish spore print color it is of the species “FalseParasol”.

OWL fragment 6.5: What characteristics must a mushroom have in order to belong to the species False parasol?

1. Class: FalseParasol
2. SubClassOf: Species
3. SuperClassOf: odor some BadSmell
4. SuperClassOf: spore–print–color some Greenish
6.5 Ontology Aware Decision Tree Learner

Now that we have a bridge between instances in the data set and domain knowledge in the ontology we will enrich each instance in the data set with what we can infer from the ontology. Suppose that we have an instance with green spore print color and a poignant odor. From the ontology we know that the species of this instance is “FalseParasol” and that it smells bad.

Algorithm 5 creates an individual in the ontology for each instance in the data set and makes object property assertions corresponding to the instance attribute values. After it is run, the inferred attribute values will have been computed for all instances. Note that instances in the data set are projected into the ontology as individuals and only the attribute values that can influence class inference are added. In our example the individuals added to the ontology would have only two dimensions: odor and spore print color.

Algorithm 5 Projects data set instances into the ontology as individuals

\begin{algorithm}
\begin{algorithmic}[1]
\Procedure{ProjectToOntology}{instances, ontology}
\ForAll{$i \in \text{instances}$}
\State $j \leftarrow \text{individual}(i)$ \Comment create individual for instance
\State $\text{ontology} \leftarrow \text{ontology} + j$ \Comment add individual to ontology
\ForAll{$a \in \text{attributes}(i)$}
\State $v \leftarrow \text{value}(i, a)$
\If{$\text{hasProperty}(a, \text{ontology}) \land \text{hasIndividual}(v, \text{ontology})$}
\State $\text{objectPropertyAssertion}(j, a, v, \text{ontology})$
\EndIf
\EndFor
\EndFor
\EndProcedure
\end{algorithmic}
\end{algorithm}

Also note that by leveraging incremental reasoning [Kazakov and Klinov, 2013] the inner loop does not trigger a full re-computation. This step can be completed in PTIME.

In Algorithm 6, for each new attribute (as defined in the ontology), we fetch the individuals for each possible attribute value. After this step we can proceed to attribute selection as we usually would in a normal decision tree algorithm. In our implementation we use a simple version of the ID3 algorithm, as specified in Algorithm 1. The reasons stated in section 4.4 for abandoning Information Gain and adopting Gain Ratio as the attribute selection criterion for the Hierarchy based Decision Tree Learner also hold when using more expressive ontologies instead of simple hierarchies, as we will show in the next section.

6.6 Attribute Selection Criterion

The approach we describe in this chapter is not dependent on any particular attribute selection criteria. As this approach subsumes the Hierarchy based Decision Tree method described in Chapter 4 it is also hindered by strongly biased criteria, like Information Gain, when dealing with attributes that directly or indirectly abstract others.
Algorithm 6 Obtains attribute values for the new generated attributes

```
1 procedure GetAttributeValues(ontology)
2   for all a ∈ subClassOf('Attribute', ontology, direct = True) do
3      if hasProperty(a, ontology) then ▷ higher levels of abstraction for a
4         for all a_h ∈ subClassOf(a, ontology, direct = True) do
5            for all v ∈ subClassOf(a_h, ontology, direct = True) do
6               instances(a_h, v) ← individuals(v, ontology)
7            end for
8         end for
9      else ▷ a is not an abstraction of an existing attribute
10         for all v ∈ subClassOf(a, ontology, direct = True) do
11            instances(a, v) ← individuals(v, ontology)
12        end for
13     end if
14 end procedure
```

The information gain $IG(A_i)$ of an attribute $A_i$ is calculated as follows:

$$IG(A_i) = H(T) - \sum_{f \in F(A_i)} p(f)H(t_{A_i=f})$$  \hspace{1cm} (6.1)$$

where $H(T)$ is the entropy of the training set and $H(t_{A_i=f})$ is the entropy of a subset of the training set formed by the instances of $T$ where the value of attribute $A_i$ is $f$. The entropy of a set $T$ is given by:

$$H(T) = - \sum_{c_j \in C} p(c_j) \log_2 p(c_j)$$  \hspace{1cm} (6.2)$$

The main problem of using Information Gain with an ontology guided decision tree learner lies within concept hierarchies and the generation of attributes that are abstractions of other existing attributes. In general attributes with fewer values are at a disadvantage [White and Liu, 1994].

**Proposition 6.6.1.** Given an attribute $A_i$ and an attribute $A_{1i}$ where at least one feature of $A_{1i}$ is at an higher level of abstraction and all others are at least at the same level then $IG(A_i) \geq IG(A_{1i})$.

**Proof.** The case where exactly one feature from $A_i$ appears in $A_{1i}$ at an higher level of abstraction and all other remain the same is a mere renaming of one feature in practical terms and is trivial to observe that no counts change because of it and consequently, in this case, $IG(A_i) = IG(A_{1i})$.

Now consider the case where $n$ features $f_1, \ldots, f_n$ from $A_i$ are represented by a common ancestor $f_a$ in $A_{1i}$ and all other features remain the same. Equivalently we might say that $A_i$ can be obtained from $A_{1i}$ by splitting $f_a$ in $n$ features. This is exactly the case where it has been shown [Quinlan, 1986; White and Liu, 1994] that the information gain of the attribute with more features is greater than or equal to the attribute with less features even if the features of the later are already sufficiently fine for the induction task at hand.

Note, however, that contrary to the hierarchy based approach described in chapter 4 which only produces attributes at higher levels of abstraction, the ontology based approach described here can produce new attributes with as many and sometimes more values than existing attributes. Without
loss of generality, consider that we have two attributes $A_1$ and $A_2$, and that both have $n$ possible values. A set of axioms can easily be written in a way that produces a new attribute $A_3$ that can have $n^2$ values.

Some existing approaches that deal with feature hierarchies attempt to modify Information Gain in such a way as to not only cancel it’s bias towards attributes with more values but also introduce a bias towards more abstract attributes [Zhang et al., 2002]. This kind of approach, valid as it might be for cases where only more abstract attributes are inferred, is of no particular advantage when dealing with true ontologies containing axioms capable of producing both attributes with more and less values than existing ones.

The gain ratio attribute selection measure [Quinlan, 1986] minimizes the bias towards attributes with more values (although it is still biased towards them [De Mántaras, 1991]) and can be calculated as follows:

$$IG(A_i) - \sum_{f \in F(A_i)} p(f) \log_2 p(f)$$  \hspace{1cm} (6.3)

Any attribute selection criteria that is as biased as, or less biased than, Gain Ratio is likely to yield good results. The distance-based attribute selection measure [De Mántaras, 1991] was specifically benchmarked against Gain Ratio and was shown to be less biased. Information Gain and standard Gini index should be avoided: both are almost equally biased and have been shown to differ in about 2% of the cases [Raileanu and Stoffel, 2004].

### 6.7 Ontology Aware Decision Tree Classifier

The model produced can be used to classify instances where most of the attributes are missing or only available at higher levels of abstraction, as long as there is enough information to infer the value of the new attributes and together with the existing ones they are enough to reach a leaf of the decision tree.

The classifier works as follows: given an instance $I$ with some possibly missing values, an ontology $O$ and a model $T$ we infer additional attribute values for $I$ by applying the axioms in $O$ to the existing attribute values of $I$. Let $I_E$ be this extended version of $I$. This extended instance $I_E$ is used to find

**Algorithm 7** An ontology based decision tree classifier

```plaintext
1 function CLASSIFY(instance, decisionTree, ontology)
2     O ← ProjectToOntology(instance, ontology)
3     I_E ← GetAttributeValues(O)
4     return GetLeaf(I_E, decisionTree)
5 end function

1 function GetLeaf(extendedInstance, decisionTree)
2     r ← root(decisionTree)
3     if children(r) = ∅ then
4         return r
5     end if
6     f ← feature(extendedInstance, r)
7     subTree ← child(r, f)
8     return GetLeaf(extendedInstance, subTree)
9 end function
```
the path to the tree leaf representing the predicted instance class. Note that the tree might contain some nodes that correspond to pre-existing attributes and others that correspond to attributes inferred by applying ontology axioms to the existing attribute values. This is a natural consequence of picking the best attribute at each step of the tree induction process from a pool containing both concrete and inferred attributes.

6.8 Results

As the approach we described here subsumes the one described in chapter 4, the results in section 4.6 also apply to this approach given the same domain knowledge. It would be reasonable to expect a comparison between HDT and OADT on those data sets. However, the performance of HDT on those sets left little space for improvement as we achieved almost maximum accuracy even on very small training sets, using only feature hierarchies. This will probably be the case in domains where feature hierarchies have enough expressive power to define the more interesting domain knowledge. Adding more complex rules did not produce any significant changes in the accuracy of OADT in neither of those two data sets. This result serves only the purpose of showing that adding irrelevant domain knowledge does not hinder the accuracy of our approach. Most of the rules introduced were not picked by the learner and as such the model produced was not influenced by them.

To assess the performance of OADT we had to look for data sets where some domain knowledge that involved more than one attribute could be materialized. This kind of domain knowledge cannot be expressed using feature hierarchies, because each feature hierarchy corresponds to only one attribute. Two data sets were selected that seemed to fit these requirements, the Car Evaluation data set and the Soybean data set from the UCI machine learning repository.

The Car Evaluation data set consists of 1728 instances with 6 attributes. The target attribute has 4 possible values that are not balanced: 70% of all instances fall in one of the classes and about 22% of the instances fall in a second class. The remaining two classes are represented by about 4% of the instances each.

The Soybean data set is composed by 683 observations with 35 attributes. The target attribute, the disease that affects the soybean, has 19 possible values.

The Car Evaluation data set has a concept structure that has some intermediate concepts that are not present in the data set. Because of this known underlying structure, this data set is particularly useful for testing our approach. The data set consists of six attributes: buying price, maintenance price, number of doors, capacity in terms of persons to carry, the size of luggage boot and the estimated safety of the car. The attributes price, technology and comfort do not exist in the data set but can possibly be inferred from existing attributes, by exploring the following known structure [Bache and Lichman, 2013]:

\[
\text{buying} \times \text{maint} \implies \text{price} \\
\text{doors} \times \text{persons} \times \text{lug\_boot} \implies \text{comfort} \\
\text{comfort} \times \text{safety} \implies \text{tech}
\]

A OWL2 ontology was written by exploring these known relations. [OWL fragment 6.6] is an example of the kind of axioms in this ontology.
Two set of experiments were then performed. The first studies the accuracy of ID3, C4.5 with pruning and subtree raising, and OADT by varying the size of the training set. A subset of instances was randomly selected from the original data set to serve as a test set. Six subsets were randomly selected from the remaining instances of the original data set, with sizes of 700, 300, 70, 50, 20 and 15 instances to be used as training sets.

The results of this first set of tests (displayed in Figure 6.2 and Figure 6.5) show that our approach outperforms both ID3 and C4.5 in all the tested subsets of both data sets. The difference is more pronounced in the smaller training sets and decreases as the number of instances in the training set increases.

On smaller training sets it is more likely that not all possible attribute values are present. As OADT tries to build a more general model by applying the axioms in the ontology to infer new attributes (potentially at a higher level of abstraction), the model produced can still be used to predict the class of instances with features that were not seen by the learner in the training set, while ID3 and C4.5 fail. When the size of the training set grows and all attribute values become present, ID3 and C4.5 start to catch up.

**OWL fragment 6.6:** A fragment of the car ontology written using the known underlying structure of the domain. Note that this is not the complete ontology and is presented only for illustration.

```plaintext
1 Class: HighPrice
2    SubClassOf: Price
3    SuperClassOf: buying value vhigh and maint value high
4 SuperClassOf: buying value high and maint value vhigh
5
6 Class: HighComfort
7    SubClassOf: Comfort
8    SuperClassOf: persons value four and lug_boot value big
9 SuperClassOf: persons value more and lug_boot value big
10 SuperClassOf: persons value more and lug_boot value med
11
12 Class: TechGood
13    SubClassOf: Tech
14    SuperClassOf: MediumComfort and safety value high
15
16 Class: TechExc
17    SubClassOf: Tech
18    SuperClassOf: HighComfort and safety value high
```

The second set of tests shows how the accuracy of all three algorithms change when some features are unknown or only partially known, e.g., not knowing the exact buying price of a car but having enough domain knowledge to infer from other attributes that, overall it is expensive.

Starting from a data set with only concrete values, six data sets were then generated with an approximate percentage of abstract and missing attribute values of 5%, 10%, 15%, 20%, 25% and 50%.

The results of these tests show that OADT is able to maintain its accuracy better than ID3 and C4.5 when, there are missing attribute values or instead of the exact attribute values, only more abstract versions of the features are available. This shows that OADT is able to make use of the existing domain knowledge to build more robust models that maintain good predictive power when some
Figure 6.2: The influence of training set size on the accuracy of ID3, C4.5 and OADT in the Car Evaluation data set.

information exists about the attribute value, but is insufficient to exactly determine it’s value and when the attribute value is not present in the data set but can some abstract version of it can be inferred from other attributes using the facts in the ontology. [Figure 6.3] shows these results.

OADT’s ability to maintain it’s predictive power in the face of less precise or missing attribute values depends on the quality of the available axioms in the ontology and, in some extent, on the nature of the classification problem.

In some classification tasks it might very well be the case that for some attributes, the exact value is needed to predict the right class. In these cases OADT will pick the concrete version of the attribute when building the model but the robustness of such a model is negatively affected, although still better than traditional approaches. [Figure 6.6] shows this.

For Soybean disease data set the underlying structure was based on the structure of the questionnaire that was filled to describe a diseased plant [Michalski and Chilausky, 1980]. The same article also derived some expert derived rules for predicting soybean diseases. We did not use these rules as they try to directly predict the class and we are only interested in providing some domain knowledge, not the entire set of rules to make a prediction.

The following equations illustrate the kind of structure that was used. Note that this is only a fragment. For the entire structure refer to [Michalski and Chilausky, 1980].

\[
\begin{align*}
  \text{date} \times \text{plant_stand} \times \text{precip} \times \text{temp} \times \text{hail} \\
  \times \text{crop_hist} \times \text{area\_damaged} & \implies \text{Environment} \\
  \ldots \\
  \text{mold\_growth} \times \text{seed\_discolor} \times \text{seed\_size} \times \text{shriveling} & \implies \text{Condition\_of\_Seed} \\
  \text{leafspots\_halo} \times \text{leafspots\_marg} \times \text{leafspot\_size} \\
  \times \text{leaf\_shread} \times \text{leaf\_malf} \times \text{leaf\_mild} & \implies \text{Condition\_of\_Leaves} \\
  \ldots \\
  \text{Condition\_of\_Leaves} \times \ldots \times \text{Condition\_of\_Seed} & \implies \text{PlantLocal\_Condition}
\end{align*}
\]
Figure 6.3: Accuracy of ID3, C4.5 and OADT in the Car Evaluation data set with an increasing number of abstract values.

Figure 6.4: The number of nodes in the tree that correspond to attributes derived from the ontology in the Car Evaluation data set.
Figure 6.5: The influence of training set size on the accuracy of ID3, C4.5 and OADT in the Soybean data set.
Figure 6.6: Accuracy of ID3, C4.5 and OADT in the Soybean data set with an increasing number of abstract values.

Figure 6.7: The number of nodes in the tree that correspond to attributes derived from the ontology in the Soybean data set.
6.9 Summary and Discussion

Ontology aware learning algorithms and classifiers are needed to explore dimensions of the data that are not explicitly defined but can be logically inferred from existing domain knowledge. By exploiting semantic reasoning and statistical methods we are able to better deal with the complexity of real world problems and with the uncertainty of practical applications.

The approach proposed in this chapter makes use of the favourable properties of the $\mathcal{EL}$ family of description logics to allow the use of ontologies that have enough expressive power to describe complex domains and still allow efficient (polynomial time) reasoning. The use of the standard OWL 2 EL profile ensures that knowledge can be shared and reused and that further advancements in reasoning techniques will automatically benefit our approach, like the improvements that were introduced in logic programming benefited ILP techniques in the past.

The structure proposed for the ontology bridges the gap between traditional data sets and the knowledge expressed in the ontology. The prescribed part of the ontology is very small and straightforward and does not limit the axioms that can be used or the way that they can be mixed and composed to express more complex domain facts. It is simply a pattern that we found effective when writing ontologies that are used in the context of classification problems.

We show that the idea of an extended instance $I_E$, first introduced in chapter 4 translates nicely from simple hierarchies, where new attributes are just existing ones at higher levels of abstraction, to full blown ontologies, where the inferred attributes can have less direct and more complex relationships with existing ones.

It is also interesting to observe that the algorithms proposed can easily deal with RDF triples instead of more common database tables or CSV files. Translating from one format to the other is a very simple operation that can be performed inline. Future work may consider learning from instances in RDF triples directly, by slightly changing the way features are counted and sets are split.

We also make the case for using unbiased splitting criteria instead of artificially biasing existing criteria to pick more abstract values. By doing so we can easily incorporate any developments that are made in this front even if not related with ontology aware decision tree learning.

We maintain the distinction made in previous chapters between learner, model and classifier and show how each of these components works in our ontology aware approach to decision tree classification.
Chapter 7

Conclusions and Future Work

The first few decades of AI were marked by a great divide between the logical and the statistical approaches to common problems like knowledge representation, automated reasoning, planning and, of course, machine learning. Logical approaches performed better when dealing with complex problems but lagged behind in domains with noise and uncertainty. On the other hand, the statistical approach excelled in these domains but suffered when there were complex relations between elements in the data.

As the real world is complex and uncertain this work tries to push for a middle ground and use logic to handle the complexity of existing domain knowledge and statistics to handle the uncertainty and noise in the observations.

7.1 Conclusions

The approach we propose uses a standard knowledge representation language with well studied properties. We recognize that complex facts and rules exist in most domains and as such logic approaches are better suited to define and reason with them. We tried to strike a balance between the expressive power provided by the knowledge representation language and the complexity of reasoning with such a language. The $\mathcal{EL}$ family of description logics was used mainly for providing reasoning with PTIME complexity while providing enough expressive power to describe known facts about most domains.

Traditional logical approaches to classification have relied on first order logic and suffered from problems related to the high computational complexity of reasoning in such a language. However, in fairness, it must be stated that some rules that can be expressed in first order logic are impossible to convey in our approach. It is our belief that the usefulness of those rules are limited to a small set of domains and applications for which logical approaches are a better fit. For most domains, however, the $\mathcal{EL}$ family of description logics presents a very good balance between complexity and expressivity and is able to describe ontologies as complex as the SNOMED CT medical ontology [Baader et al., 2005].

Our approach excels in problems where:

1. there is some domain knowledge available, even if the usefulness of such knowledge for the problem at hand is not known a priori. The inference of new knowledge from existing facts is a logical process but the decision of which of these knowledge will incorporate the model is a statistical one and as such having irrelevant domain knowledge does not hinder the learning algorithm.
2. The size of the training set is relatively small, i.e., there are not a lot of labeled instances readily available. This does not mean that our approach works better with small training sets, it just means that when the training set grows, traditional statistical approaches catch up in terms of accuracy. This is not surprising. With a big enough and well balanced training set, learning algorithms like C4.5 are able to infer by themselves most of the useful rules. However, in small training sets, our approach is able to significantly outperform C4.5.

3. A robust classifier is needed, i.e., we want to produce a model where instances with previously unseen features can still be accurately classified.

4. Simplicity is a goal in itself. Although simpler models do not necessarily lead to greater accuracy, simpler models are still preferable to complex ones if both have comparable performances. Simpler models are easier to understand, explain and share [Domingos, 1998].

The datasets we used favour existing approaches, like C4.5, because they have mutually influenced each other. On one hand some later optimizations to C4.5 have been tested against those data sets and it is fair to say that they are at least a bit influenced by them. On the other hand those data sets have been through some pre-processing (feature selection, balancing, etc) to perform well with existing classification algorithms. In spite of this our approach was able to outperform ID3 and C4.5 in those data sets.

7.2 Future Work

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

The first is extending the algorithms to better support numerical (continuous) attribute values. It is already possible to express numerical intervals in the ontology but as we extended from ID3 we inherited it’s poor ability to deal with continuous-valued attributes.

Further improvements may include extending the learner to directly support RDF triples without the need of translating them to an intermediate form. This would enable the algorithm to make a more efficient use of the semantic web stack [Berners-Lee et al., 2001] to obtain both the domain knowledge in the form of an ontology and the training set in the form of sets of RDF triples. Other related development would be enabling the use of SPARQL [Quilitz and Leser, 2008] to query distributed RDF data sources to be used, on demand, as training sets.

We have implemented some of the most widely known attribute splitting criteria like Information Gain, Gini Index and Gain Ratio. However some newer and less known approaches [De Mántaras, 1991; Strobl et al., 2007] less biased than Gain Ratio may yield even better results without any other changes to our approach.

Finally, the study and development of a more generic framework that is able to apply the ideas explored in this work to more classification algorithms without the need of making significant changes to every implementation.
Bibliography


