Decision Tree Learner in the Presence of Domain Knowledge

João Vieira and Cláudia Antunes

Instituto Superior Técnico, Universidade de Lisboa
{joao.c.vieira, claudia.antunes}@ist.utl.pt

Abstract. In the era of semantic web and big data, the need for machine learning algorithms able to exploit domain ontologies is undeniable. In the past, two divergent research lines were followed, but with background knowledge represented through domain ontologies, is now possible to develop new ontology-driven learning algorithms. In this paper, we propose a method that adds domain knowledge, represented in OWL 2, to a purely statistical decision tree learner. The new algorithm tries to find the best attributes to test in the decision tree, considering both existing attributes and new ones that can be infered from the ontology. By exploring the set of axioms in the ontology, the algorithm is then able to determine in run-time the best level of abstraction for each attribute, infer new attributes and decide the ones to be used in the tree. Our experimental results show that our method produces smaller and more accurate trees even on data sets where all features are concrete, but specially on those where some features are only specified at higher levels of abstraction. We also show that our method performs substantially better than traditional decision tree classifiers in cases where only a small number of labeled instances are available.

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

Topics: Data Mining, OLAP, and Knowledge Discovery; Semantic Web and Databases

1 Introduction

The automatic generation of simple and accurate classifiers from data is one of the major fields of data mining. Classification algorithms are supervised methods that look for and discover the hidden associations between the target class and the independent variables [12]. In a traditional supervised inductive learning scenario, the classifier is produced solely from a given set of already labeled instances or observations, ignoring any existing domain knowledge available.

However, with the advances of the semantic web, this knowledge is becoming increasingly available through domain ontologies, in many areas of application. Moreover, it is usual that the instances to be classified are described by features at different levels of abstraction. That is, the values of a particular attribute are specified at different levels of abstraction on different instances.

One of the most widely known data sets in classification is the Mushroom data set [3,11]. One might think that there is not much interest or knowledge about mushrooms beyond the circle of professional biologists or amateur witches but as it turns out there is a vibrant community of mushroom enthusiasts (and not only of the psychadelic kind) across the globe. As such, over time, there have been numerous guides and books about the subject and a great number of people have learned to correctly identify mushrooms through luck and experience, which is remarkable in an area where learning from one's mistakes might not be advisable.

One of the characteristics that are important when identifying mushrooms is its odor. Experts with gifted noses will tell you that an odor of anise or almonds is a good sign as is no odor at all. However if you smell creosote for example, it will probably be a good idea to have dinner somewhere else. But can we help amateurs with less sensitive noses avoid death by mushroom poisoning? Suppose that while some experts input an exact odor, other people can only input if it smells good or bad. One can certainly leverage existing domain knowledge to make an automated classifier aware of which odors are pleasant to humans and which are not and if possible classify the mushroom with this less precise information.

More generally if experts learn from guides and books in addition to examples and experience, why can't classifiers also make use of existing domain knowledge in addition to a set of training labeled instances?

If we start to read a book about mushrooms, we soon realize that if we want to survive a dinner of these dangerous delicacies, we will need more than knowing attribute values at different levels of abstraction. Often we are teached to infer new attributes, like species, from a combination of existing ones. For example, there is a common species of mushrooms called Parasol. Mushrooms in these species share, of course, the same basic characteristics, like cap surface and color, gill size and spacing and so on. However there is another species that share almost the same characteristics and their distinction is only possible by looking at a combination of two attributes, a bad smell and a greenish spore print color.

This is also very common in medical diagnosis, when some conditions can only be considered when certain specific combinations of symptoms occur. For example, weakeness and fatigue combined with weight loss and yellow discoloration of the skin points to liver problems. It is not enough to know the cause or the treatment but it helps narrow it down. The classifier we propose can, for example, use domain knowledge to determine probable liver problems and then use that extra information and a set of labeled instances to arrive at a concrete diagnosis.

For automated classification methods to be adopted in practice, it is crucial that a relationship of trust can be established between domain experts and the models generated. When the cost of making mistakes is very high, numerical validation is usually not enough. This is why it is so important that the generated models are simple, understandable, and somewhat aligned with certain facts that are known to be true in the domain.

Beside the inability of using anything other than labeled examples, one of the major problems faced during classifiers training is the *overfitting* of the learned model to the training data. Usually resulting in excessively complicated models, with low predictive power for unseen data [6]. Overfitting is then the production of models that include more terms or use more complicated rules than necessary, compromising the fundamental rule of machine learning – the *Occam's razor* principle.

Adding irrelevant predictors can make models perform worse because the coefficients fitted to them add random variation to the subsequent predictions.

By ignoring the relationship between attribute values, e.g., the fact that *anise* and *pleasant* odors are not two unrelated features, but the same at different levels of abstraction, most current algorithms produce models that have very low portability [9].

We believe that adding domain knowledge is the key to the solution of these problems. The introduction of ontologies, as a means to formally represent existing background knowledge, in the learning process will give rise to the production of simpler and more accurate models. Since, through ontologies will be possible to consider different levels of abstraction and explore the relationship between the concepts instantiated in the data set.

In this paper, we overview the related work and propose an approach that introduces domain knowledge, represented as an ontology written in a standard format, OWL 2, in the context of classification. We propose a method that given a set of examples and an ontology expressing existing domain knowledge will automatically classify instances making use of the available domain knowledge when, and in the extent that, it helps produce simpler, more accurate decision trees and more probable classifications.

2 Literature Review

Two major approaches have dominated 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 [7] that is present in field applications.

This duality is clearly represented in classification, where a lot of efforts were 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 [5], back-propagation [20], Naive Bayes, KNN [1], C4.5 [18], among others. These algorithms are usually very efficient in learning a model. Moreover, the models produced yield good levels of accuracy for unseen data, if training sets were 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, Inductive Logic Programming (ILP) is the most known representant of the logic approach to classification [4]. 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.

Although ILP systems benefit from relevant background knowledge to construct simple and accurate theories more quickly [21], 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 [17]. Experimental results [19,8] also show that performance (in terms of time complexity) is much worse than statistical approaches, like C4.5. Further, traditional ILP is unable to cope with the uncertainty of real-world applications such as missing or noisy data, a known drawback when compared to the statistical approach.

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.

There has been surprisingly little work on probabilistic learning with datasets described using formal ontologies [14]. Ontologies are crucial to deal with semantic interoperability and with heterogeneous data sets.The strenghts of purely statistical methods are related with their relative simplicity and ability to work with data that underwent less preparation than required by the logic approach (where data and existing knowledge must be represented or transformed to first order logic). It is also 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.

However, it ignores the complexities of the real world. First, it is not possible to express or make use of existing domain knowledge, to explicitate relationships between attributes or hierarchies of features. And second, it doesn't allow for constraining the results according to facts which are known to be true, even if not represented in the subset of data being fed to the learning algorithm.

EG2 [15] was one of first approaches to extend a purely statistical method, the ID3 decision tree learner, to make use of background knowledge in order to reduce the *classification cost*.

More recently, an ontology-driven decision tree learning algorithm was proposed [24] 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 IS-A relations associated with each attribute. It consists in a top-down concept hierarchy guided search in a hypothesis space of decision trees.

A variation of the Naïve Bayes Learner making use of attribute-value taxonomies (AVT-NBL) was proposed [23]. It starts with the Naïve Bayes classifier based on the most abstract value for each attribute, and successively moves in the direction of the more concrete values, i.e., the ones appearing in the data set. The idea is to stop somewhere in between, in order to achieve a balance between the complexity of the resulting classifier and its classification accuracy.

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

3 Preliminaries

As far as the authors know 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.

We need a standard way of expressing domain knowledge, so it can be shared and reused. The Web Ontology Language, OWL 2 [13] 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 and with the Manchester syntax. Nonetheless we briefly review the main components of an OWL 2 ontology.

3.1 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 has 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 OWL DL ontology is a problem in NEXPTIME which is highly undesirable for our intended application.

3.2 OWL 2 EL

Fortunately a subset exists that trades off some aspects of the OWL DL expressive power in return for PTIME complexity in all the standard reasoning tasks. From now on, whenever we refer to OWL or to ontologies, keep in mind that we are talking about OWL 2 EL.

We wish to have a set of axioms that define class membership and then 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.

3.3 ELK

We use Elk [10] 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.

4 Structuring an Ontology to Support Classification Problems

Until now we have presented 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 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 1.1. What mushrooms odors smell bad?

```
Class: BadSmell

Individual: c
    Types: BadSmell
Individual: y
    Types: BadSmell
Individual: f
    Types: BadSmell
Individual: m
    Types: BadSmell
Individual: p
    Types: BadSmell
Individual: s
    Types: BadSmell
```

OWL fragment 1.2. Green spore print colors make greenish mushrooms

```
Class: Greenish

Individual: r
    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 1.3. Definition of the attributes *odor* and *spore print color* in the ontology, allowing the definition of axioms that use these attributes

```
ObjectProperty: odor
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 this 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 1.4. Attribute hierarchy showing a new class *species* and an higher level of abstraction *smell* for the attribute *odor*

```
Class: Odor
Class: Smell
    SubClassOf: Odor
Class: Species
Class: Attribute
```

```
6       SuperClassOf: Odor
7       SuperClassOf: Species
8
9   Class: BadSmell
10      SubClassOf: Smell
```

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 1.5. 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
```

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 1 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 all instances are classified. 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.

Also note that by leveraging incremental reasoning the inner loop does not trigger a full re-computation. This step can be completed in PTIME.

In Algorithm 2, for each new attribute (as defined in the ontology), we fetch the individuals for each possible attribute value. After this step we can proceed

Algorithm 1 Projects data set instances into the ontology as individuals

```
1  procedure ProjectToOntology(instances, ontology)
2      for all i ∈ instances do
3          j ← individual(i)                          ▷ create individual for instance
4          ontology ← ontology + j                    ▷ add individual to ontology
5          for all a ∈ attributes(i) do
6              v ← value(i, a)
7              if hasProperty(a, ontology) ∧ hasIndividual(v, ontology) then
8                  objectPropertyAssertion(j, a, v, ontology)
9              end if
10         end for
11     end for
12 end procedure
```

Algorithm 2 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 for
15 end procedure
```

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.

5.1 Attribute Selection Criterion

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

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

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:

$$H(T) = -\sum_{c_j \in C} p(c_j) \log_2 p(c_j) \tag{2}$$

One might think that as ID3 recursively selects the attribute with the highest information gain it would naturally pick attributes at higher levels of abstraction if those led to simpler, more accurate trees.

This is, however, not the case. The information gain metric is biased through attributes with a greater number of features [22].

Proposition 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 pratical 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 [16,22] 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.

This is highly undesirable when dealing with attributes at different levels of abstraction. 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. Using information gain this attribute will never be selected.

The gain ratio attribute selection measure [16] minimizes this bias and can be calculated as follows:

$$\frac{IG(A_i)}{-\sum_{f \in F(A_i)} p(f) \log_2 p(f)} \tag{3}$$

5.2 Ontology Aware Decision Tree Classifier

The model produced can be used to classify instances where most of the attributes are missing 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.

Given an instance I with some possibly missing values we compute new attribute values using a method analogous to the one described in the previous section and obtain an extended version of I. We use this extended version to navigate the decision tree as usually.

6 Results

In order to execute some experiments and compare the performance of the proposed algorithm with standard ID3 decision tree algorithm a `Java` implementation was developed, as part of the D2PM framework [2].

In spite of data with values specified at different levels of abstraction being commmon in many domains of application there are few standard benchmark data sets with these characteristics and with an associated ontology. We selected the *Mushroom* data set from the UCI Machine Learning Repository as a starting point.

Domain knowledge obtained from the book "The Mushroom Hunter's Field Guide" was made explicit in an OWL 2 ontology.

Table 1. Accuracy and tree size of ID3 and OADT on the original data

	ID3 Accuracy	OADT Accuracy	ID3 Tree Size	OADT Tree Size
S1	0.785	0.999	5	4
S2	0.999	1.000	7	4
S3	0.999	1.000	7	4
S4	0.993	0.999	5	4
S5	1.000	1.000	7	4
Avg.	0.9552	0.9996	6.2	4

Three sets of experiments were then executed. The first compares the accuracy of the proposed Ontology Aware Decision Tree with the standard ID3 algorithm on the original data, where all values are concrete. We also look at the complexity of the produced decision trees. Figure 1 shows an example of a decision tree generated by OADT for randomly selected small 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**.

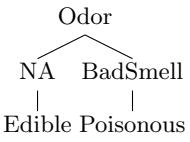

Fig. 1. Example of a decision tree generated by OADT from a small training set (< 50 instances)

The second shows how the accuracy of both algorithms changes with increansingly smaller 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. The results are shown in Figure 2.

The third studies how the accuracy of both algorithms changes with an increasing number of values being abstract, e.g., not knowing the exact *odor* of

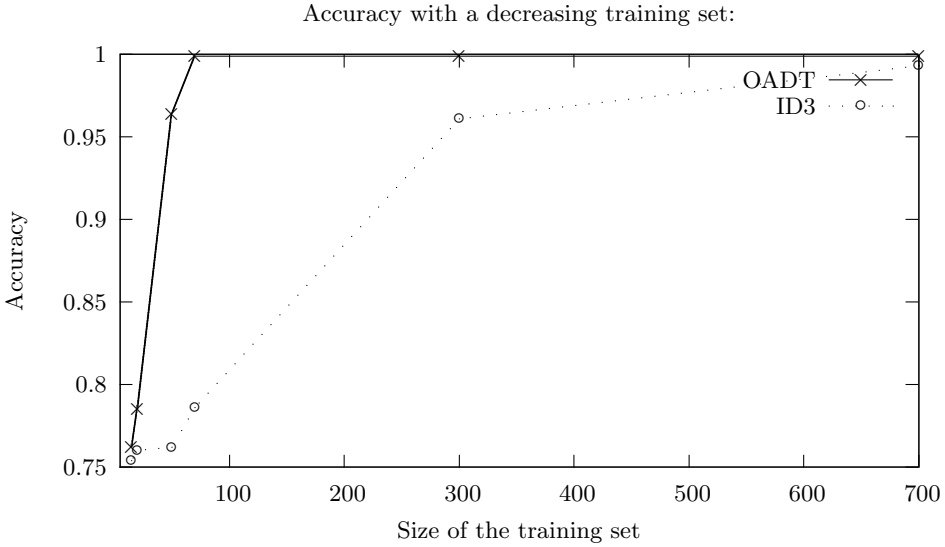

Fig. 2. The influence of training set size on the accuracy of ID3 and OADT.

a mushroom but being able to tell if it has a pleasant or bad smell. Starting from a data set with no abstract values, five data sets were then generated with an approximate percentage of abstract values of $5\%, 10\%, 15\%, 20\%$ and 25%. Figure 3 shows these results.

To assess the accuracy of the two algorithms on the original data set, 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. Table 1 shows these results.

These results are in line with our expectations. First, even on data were 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, OADT maintains its performance remarkably well while the performance of traditional ID3 decreases as more values are expressed at higher levels of abstraction.

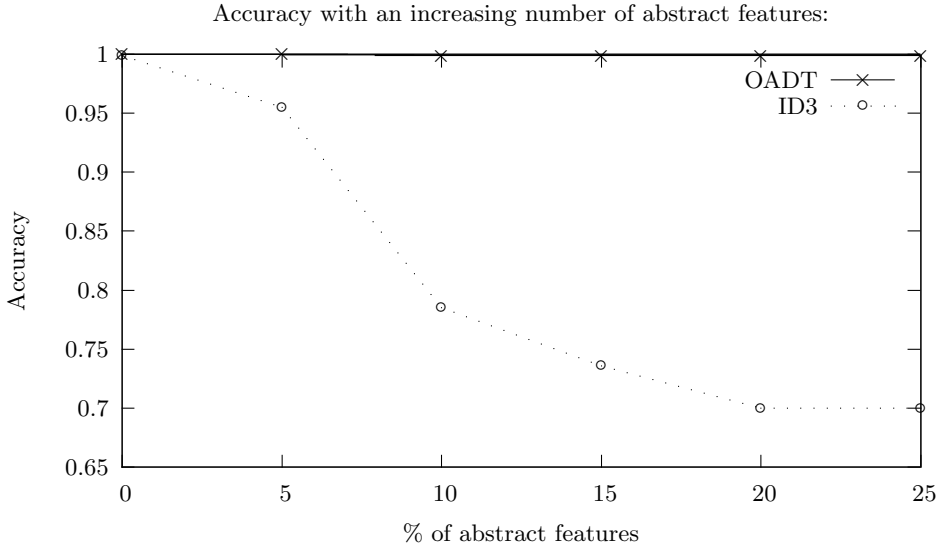

Fig. 3. Accuracy of ID3 and OADT in a data set with an increasing number of abstract values.

7 Summary

In this paper, we have described an approach for learning from ontologies, represented in a standard knowledge representation language, OWL 2 EL, and from a set of training data.

We show that it is possible to produce simpler decision trees that perform as good or better than traditional approaches. We also show that domain knowledge helps produce classifiers that perform very well in small training sets comparatively to traditional algorithms.

We also describe how to change the classifier to leverage available knowledge to classify instances with values not seen in the training set, by infering values from the ontology. This is useful to classify instances where the concrete value of some attributes is not known but the abstract class is, e.g., not knowing the exact phylum of a bacteria, but knowing it is Gram positive.

References

1. Altman, N.: An introduction to kernel and nearest-neighbor nonparametric regression. The American Statistician 46(3), 175–185 (1992)
2. Antunes, C.: D2pm: Domain driven pattern mining. Tech. rep., project report, Tech. Report 1530, IST, Lisboa (2011)
3. Bache, K., Lichman, M.: UCI machine learning repository (2013), `http://archive.ics.uci.edu/ml`

4. Blockeel, H., De Raedt, L.: Top-down induction of first-order logical decision trees. Artificial intelligence 101(1), 285–297 (1998)
5. Boser, B.E., Guyon, I.M., Vapnik, V.N.: A training algorithm for optimal margin classifiers. In: Proceedings of the fifth annual workshop on Computational learning theory. pp. 144–152. ACM (1992)
6. Bramer, M.: Using j-pruning to reduce overfitting in classification trees. Knowledge-Based Systems 15(5), 301–308 (2002)
7. Domingos, P., Kok, S., Poon, H., Richardson, M., Singla, P.: Unifying logical and statistical ai. In: AAAI (2006)
8. Dzeroski, S., Jacobs, N., Molina, M., Moure, C., Muggleton, S., Laer, W.V.: Detecting traffic problems with ilp. In: Proceedings of the 8th International Workshop on Inductive Logic Programming. pp. 281–290. Springer-Verlag (1998)
9. Hawkins, D.M.: The problem of overfitting. Journal of chemical information and computer sciences 44(1), 1–12 (2004)
10. Kazakov, Y., Krtzsch, M., Simank, F.: The incredible elk. Journal of Automated Reasoning 53(1), 1–61 (2014), http://dx.doi.org/10.1007/s10817-013-9296-3
11. Lincoff, G., Nehring, C.: National Audubon Society Field Guide to North American Mushrooms. Knopf (1997)
12. Maimon, O., Rokach, L. (eds.): Data Mining and Knowledge Discovery Handbook, 2nd ed. Springer (2010)
13. Motik, B., Patel-Schneider, P.F., Parsia, B., Bock, C., Fokoue, A., Haase, P., Hoekstra, R., Horrocks, I., Ruttenberg, A., Sattler, U., et al.: Owl 2 web ontology language: Structural specification and functional-style syntax. W3C recommendation 27, 17 (2009)
14. Muggleton, S., De Raedt, L., Poole, D., Bratko, I., Flach, P., Inoue, K., Srinivasan, A.: Ilp turns 20. Machine Learning 86(1), 3–23 (2012)
15. Núñez, M.: The use of background knowledge in decision tree induction. Machine learning 6(3), 231–250 (1991)
16. Quinlan, J.R.: Induction of decision trees. Machine learning 1(1), 81–106 (1986)
17. Quinlan, J.R., Cameron-Jones, R.M.: Foil: A midterm report. In: Machine Learning: ECML-93. pp. 1–20. Springer (1993)
18. Quinlan, J.R.: C4. 5: programs for machine learning, vol. 1. Morgan kaufmann (1993)
19. Roberts, S., Jacobs, N., Muggleton, S., Broughton, J., et al.: A comparison of ilp and propositional systems on propositional traffic data. In: Inductive Logic Programming, pp. 291–299. Springer (1998)
20. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning representations by back-propagating errors. Cognitive modeling 1, 213 (2002)
21. Srinivasan, A., King, R.D., Muggleton, S.: The role of background knowledge: using a problem from chemistry to examine the performance of an ilp program. Transactions on Knowledge and Data Engineering (1999)
22. White, A.P., Liu, W.Z.: Technical note: Bias in information-based measures in decision tree induction. Machine Learning 15(3), 321–329 (1994)
23. Zhang, J., Kang, D.K., Silvescu, A., Honavar, V.: Learning accurate and concise naïve bayes classifiers from attribute value taxonomies and data. Knowledge and Information Systems 9(2), 157–179 (2006)
24. Zhang, J., Silvescu, A., Honavar, V.: Ontology-driven induction of decision trees at multiple levels of abstraction. In: Abstraction, reformulation, and approximation, pp. 316–323. Springer (2002)