Feature-Based Learners for Description Logics

Daniel Kudenko
Department of Computer Science
University of York
Heslington, York YO10 5DD
UK
kudenko@cs.york.ac.uk

Haym Hirsh
Department of Computer Science
Rutgers University
Piscataway, NJ08855
USA
hirsh@cs.rutgers.edu

Abstract

Inductive learning algorithms that have been applied to learning in description logics (DL) have not been as well studied and optimized as the more general class of feature-based learning algorithms. This paper proposes a way to apply feature-based learners to DL learning tasks by presenting a method to compute a feature-vector representation for DL instances. The representation is based on concepts computed by a DL learning algorithm and by a feature generation method that has previously been applied to sequence categorization tasks. We show encouraging empirical test results using the feature-based learning systems Ripper, C5.0, and Naive Bayes.

1 Introduction

Description logics (DL) are a well-studied formalism for the representation of knowledge, for which inductive learning problems have been defined in the past [3, 5]. When presented with a set of labeled examples (i.e., DL individuals), the learning algorithms compute a hypothesis to predict the label of new, previously unseen examples.

Even though description logics offer the advantage of expressiveness of the representation language, the learning algorithms that have been applied to DLs are not as well-studied and well-optimized as the more general feature-based approaches. Up until now it was not clear how to apply feature-based learners to inductive learning in the DL domain.

Feature-based learning algorithms such as Ripper [2] and C5.0 [8] require the input data to be represented in a feature-vector format, i.e. as a collection of feature-value pairs. A feature is a mapping function from the set of examples to a set of feature values (the *feature value domain*). For example, the weather at a certain time point could be represented as a feature-vector using the three features temperature (T), humidity (H), and pressure (P). The feature value domain for T and H can be defined as integers, while the feature value domain for P can be defined as the set {low, middle, high}. Thus, an example could be represented as the feature-vector `((T 50) (H 65) (P low))`.

In this paper we propose a method how DL individuals can be represented as a feature-vector, and how such a representation can be automatically computed from the training data. We also show encouraging results of experiments using this feature generation method.

We would like to thank Alex Borgida and William Cohen for many helpful discussions. Furthermore, we thank William Cohen for providing LCSLearnDisj and some of the datasets used in this research.

2 Application of feature-based learners to DL learning problems

The main problem in the application of feature-based learners to any domain is the question how to represent examples as feature-vectors. This is especially true for DL individuals, where there is no straight-forward way to represent them in terms of feature-value pairs suitable for machine learning.

We propose to represent DL individuals as a Boolean feature vector, where feature values are either *True* or *False* and each feature denotes the membership of the individual in a specific concept. More formally, for a given set of concepts C_1, C_2, ..., C_n we define a set of features F_1, F_2, ..., F_n, where for an individual I the feature F_i is true if I is a member of C_i, and F_i is false otherwise.

We propose two methods to compute the set of concepts on which the feature representation of individuals is based. The first method is based on the learning algorithm for description logics proposed by Cohen and Hirsh [3]. The second method is based on the FGENERAL feature generation algorithm, originally developed for sequence categorization tasks [7]. In the next section both methods are described in more detail.

3 Computation of Feature Representation for DL Individuals

As stated above, the computation of a representation of DL individuals as feature vectors is based on the computation of a collection of concepts, each of which denotes a Boolean feature. We propose two methods to compute such a set of concepts from the set of training examples.

3.1 Approach 1: LCSLearnDisj

LCSLearnDisj is a machine learning algorithm developed by Cohen and Hirsh [3] for two-class learning tasks in description logics. It computes a set of concepts for one of the classes (the so-called *foreground class*) from the set of training examples. The resulting set of concepts is interpreted as a disjunctive description for the foreground class, i.e., if an individual is a member of one of the concepts then that individual is labeled as belonging to the foreground class.

The computation of a concept is started by computing the least general concept (or an approximation) that subsumes a pair of individuals that have been greedily chosen from the set of training examples of the foreground class (the so-called *seed instances*). The greedy choice is based on the number of training examples from the foreground class that are subsumed by the *seed concept*. This concept is generalized step by step until it subsumes one or more individuals from the background class (i.e., the set of individuals that do not belong to the foreground class), at which point the concept is appended to the resulting list. The generalization steps are based on the computation of the least common subsumer. All individuals that have been subsumed by this concept are removed from the set of training examples, and the above process of greedily choosing a seed and generalizing it is repeated until there are no training examples from the foreground class left. Since this algorithm has been published before we will not discuss any further details but refer the reader to [3].

3.2 Approach 2: FGENERAL

FGENERAL is the generalized version of the FGEN feature generation algorithm for sequence categorization tasks [7]. FGENERAL for description logics computes a set of concepts in a similar fashion to LCSLearnDisj, with the following three main differences: 1. A seed instance is selected greedily based on the number of instances from the foreground class subsumed by the resulting feature concept; 2. the generalization process is guided by a similarity heuristic and is stopped once the classification accuracy of the concept on a holdout set decreases; 3. each of the resulting concepts undergoes a pruning process, i.e., a generalization in small steps.

The input to FGENERAL are pre-classified training examples (i.e. individuals). The output is a set of concepts, which can be used to map each training and test example into a Boolean feature vector. FGENERAL generates a set of concepts (i.e., features) for each class separately, and returns the union of the sets.

The FGENERAL algorithm consists out of two main parts: concept construction (shown in Figure 1) and generalization (pruning).

Concepts are constructed incrementally, starting with the least general concept that subsumes a single individual of the foreground class (the seed around which a feature is generated).

A concept is constructed step by step by making it more general to subsume at least one additional individual from the target class in each iteration. The additional individual is one that requires the fewest generalizations of the current concept to subsume the additional individual. In other words, this additional individual is "most similar" to the concept.

After each iteration the new concept is heuristically evaluated. As a heuristic we use classification accuracy of the concept on a holdout set, which is one third of the training set (since concepts correspond to Boolean features they can be interpreted as two-class classifiers). The generalization steps are continued, until the resulting concept has a lower heuristic value than the concept in the previous step.

This feature generation algorithm creates features that tend to over-fit the training data. In order to solve this problem each computed concept is subsequently generalized (pruned) after it is created. The pruning operations of the concepts are: 1. removing a primitive superconcept restriction; 2. incrementing a maximum number restriction (up to a certain maximum); 3. decrementing a minimum number restriction; 4. removing a value restriction; 5. removing the one-of restriction; 6. removing a same-as restriction. The generalization is performed in a greedy hill-climbing fashion by trying to increase the heuristic value of the pruned concept as much as possible and stopping the generalization as soon as a local maximum is reached (i.e. the heuristic value of the current concept is lower than the value of the concept of the last iteration).

More details on the FGENERAL algorithm can be found in [6].

4 Empirical Evaluation

We evaluated the performance of feature-based learners on a number of DL learning tasks in the CLASSIC description logic [1]. These tasks have been used before in an empirical evaluation of LCSLearnDisj [3], which produced the best results compared to other DL learning algorithms. For the empirical evaluation we selected five datasets from three domains. The KRK, KBK, and KQK datasets are used to learn to detect

```
/* main function that computes the set of FGENERAL concepts.
   Bi union Hi are training examples (individuals) of class i.
   Hi is used as a holdout set for evaluation purposes
   (1/3 of the training set).  */
ConstructConcepts(B1,B2,H1,H2) {
  Seeds = B1;   /* set of potential seeds */
  Concepts = {};
  I-Sets = {};
  while (Seeds not empty) {
    For each s in Seeds do {
      Cs = ComputeConcept(s,B1,B2,H1,H2);  /* compute concept from
                                              seed example s */
      Is = {};
      For each s' in B1 do {
        if (Subsumes(Is,s')) {
          add s' to Is;
        }}
      I-Sets = I-Sets union {Is};
    }
    Let I be the biggest set in I-Sets, i.e., the biggest of the Is
             sets computed in the iterations above;
    Let C be the concept corresponding to I;
    Concepts = Concepts union {C};
    Seeds = Seeds - I;   /* remove all elements of I from Seeds */
  }
  return Concepts;
}


/* compute a concept corresponding to a cluster built around seed s */
ComputeConcept(s,B1,B2,H1,H2) {
  C = LeastGeneralSubsumer(s);
  OldValue = GetValue(C,H1,H2); /* compute heuristic that
                                   evaluates C */
  C-Set = {};
  repeat
    For each s' in B1 do {
      CG' = LCS(C,LeastGeneralSubsumer(s'));  /* generalize C to subsume s'*/
      C-Set = C-Set union {CG'};
    }
    Let CG be the element in C-Set with maximum concept description size computed
          in the iterations above (if there is a tie the choice is
          arbitrary);
    Let s' be the example that gave rise to the computation of CG;
    Remove s' from B1;
    NewValue = GetValue(CG,H1,H2);
    if (NewValue >= OldValue) then C = CG;
  until NewValue < OldValue;  /* Hill-climbing on heuristic
                                 evaluation of C */
  return C;
}


/* heuristic evaluation function for concepts:
   proportional classification accuracy on the holdout set*/
GetValue(C,H1,H2) {
  p = number of examples e in H1 with Subsumes(C,e) == True;
  n = number of examples e in H2 with Subsumes(C,e) == True;
  P = number of examples in H1;
  N = number of examples in H2;
  return ((p/P)+(N-n)/N)/2;  /* classification accuracy of C on
                                H1 and H2 */
}
```

Figure 1: FGENERAL Algorithm for DL

the legality of chess endgame positions involving king-rook-king, king-bishop-king, and king-queen-king pieces, respectively. The loan dataset determines whether payment on a student loan can be deferred. Finally, the wine dataset contains food-wine pairings that are labeled acceptable or unacceptable.

The error rates using LCSLearnDisj as a stand-alone learner form the baseline for our evaluation. In order to test whether our feature-based approach can improve over these results, we formed a feature-vector from the concepts computed by LCSLearnDisj and FGENERAL, using the method described before. The feature vector is used to represent the training examples in order to apply three feature-based learners: Ripper, a rule-learning algorithm based on inductive logic programming approaches [2], C5.0 [8], a decision-tree learner, and Naive Bayes [4], a simple and efficient probabilistic learner that is based on Bayesian inference.

The resulting 10-fold cross-validation error rates for the three learners are shown in Table 1. The first column (labeled LCSLD) shows the error rates of LCSLearnDisj alone, and the second column (labeled Comb-features) shows the error rates of the respective learner using the feature representation based on the results of LCSLearnDisj and FGENERAL. In order to show that the combination of both approaches is necessary to improve results, we included the error rates of the feature-based learners using LCSLearnDisj and FGENERAL features alone in the third and fourth column, respectively. The winning error rates are printed in bold face.

5 Discussion

The results of the empirical evaluation show the success of our feature-based learning approach on 4 datasets for Ripper and C5.0, and on only 2 datasets for Naive Bayes. The reasons for the weak results when using Naive Bayes might be found in the principal difference between Naive Bayes and the other learning algorithms used in the empirical tests. Ripper and C4.5 are so-called greedy learning algorithms that start from an empty hypothesis (i.e., all examples belong to the foreground class) and greedily add conditions of the form $\mathrm{F}_i = \mathrm{V}_i$ to it, based on an information-gain heuristic (where F_i is a feature and V_i is a value from the respective feature value domain). Therefore, Ripper and C5.0 could be interpreted as top-down learning algorithms starting with the most general hypothesis and specializing it, while LCSLearnDisj and FGENERAL can be interpreted as bottom-up algorithms that start with a very specific concept and generalize it step by step. One reason for the improved accuracy might be found in this combination of top-down and two different kinds of bottom-up learning.

Naive Bayes, on the other hand, computes a probabilistic weighting of features based on statistical measurements and application of Bayes' rule, and can not be classified as either top-down or bottom-up.

The Loan dataset is a special case, in which the feature-based learning approaches increased the error rates in all cases. This might be based on the fact that there is only a small number of examples of the foreground class in the dataset, which can lead to problems when applying FGENERAL. Experiments on variations of the KxK datasets with a small number of foreground examples have strengthened the suspicion that FGENERAL performs poorly in such cases. One of the reasons is that the FGENERAL heuristic is based on classification accuracy on a hold-out set of training data. If this set is very small then the heuristic can become misleading.

Overall, the empirical results show a modest improvement in accuracy. A disadvantage of the LCSLearnDisj algorithm is its inability to deal effectively with noise in the training data (i.e. where there are errors in the definition of training examples). Feature-based approaches on the other hand are known to handle noisy data quite well. Therefore we suspect that larger improvements of accuracy could be shown with datasets that contain noise.

6 Outlook

There are many interesting areas of future work involving the feature-based approach to learning in DL. In this section we present two research directions we believe might be promising.

6.1 DL Feature Generation Algorithms

Even though the combination of LCSLearnDisj and FGENERAL has shown to be useful, FGENERAL has been specifically developed and optimized for sequence categorization tasks. We believe that there is room for further improvement of results using feature generation techniques better suited for DL. Also note, that since LCSLearnDisj and FGENERAL are based on the computation of the least common subsumer of two concepts, their application is restricted to the set of DLs for which the LCS operation is available (such as $\mathcal{ALN}$ and $\mathcal{ALE}$), and where disjunction is not allowed.

6.2 A Basic Feature Representation

The hypotheses computed by Ripper and C5.0 are conjunctions of feature-value restrictions. Furthermore, one could interpret a normalized DL concept as a conjunction of atomic concept-forming operators (e.g, number restrictions, value restrictions, and primitive superconcepts). Therefore, an interesting feature representation could be based on all possible simple (atomic) concepts such as primitive concepts, concepts that denote only a single number restriction an a single role (e.g.,

Table 1: Error Rates on Test Domains

Dataset (Learner)	LCSLD	Comb-features	LCSLD-features	FGENERAL
KRK (Ripper)	4.0	**3.0**	4.0	16.0
KBK (Ripper)	2.0	**0.0**	2.0	29.0
KQK (Ripper)	6.0	**5.0**	7.0	20.0
Loan (Ripper)	**6.0**	7.7	7.1	22.7
Wines (Ripper)	12.8	**10.2**	12.4	30.2
KRK (C5.0)	4.0	**3.0**	4.0	15.0
KBK (C5.0)	2.0	**0.0**	2.0	28.0
KQK (C5.0)	6.0	**5.0**	7.0	18.0
Loan (C5.0)	**6.0**	7.7	7.1	20.4
Wines (C5.0)	12.8	**12.0**	12.8	29.4
KRK (Naive Bayes)	4.0	**3.0**	4.0	18.0
KBK (Naive Bayes)	**2.0**	7.0	**2.0**	29.0
KQK (Naive Bayes)	6.0	**2.0**	6.0	14.0
Loan (Naive Bayes)	**6.0**	13.8	**6.0**	23.8
Wines (Naive Bayes)	**12.8**	14.2	**12.8**	35.9

`(at-least 2 R)`), and concepts that denote only a single value restriction (e.g., `(all R C)`).

Such a representation could potentially lead to a very large feature set, mainly because of the recursion introduced by the value restrictions. Therefore, simplifying assumptions need to be made.

For example, a very simple feature representation could be based on the following set of atomic concepts:

$$\{C_i \mid \text{for all primitive concepts } C_i\}$$
$$\cup\ \{(\texttt{at-least}\ n\ R) \mid \text{for all roles } R,\ 0 < n < n_{max}\}$$
$$\cup\ \{(\texttt{at-most}\ n\ R) \mid \text{for all roles } R,\ 0 < n < n_{max}\}$$
$$\cup\ \{(\texttt{all}\ R\ C_i) \mid \text{for all roles } R, \text{ primitive concepts } C_i\}$$

Such sets of atomic concepts would allow feature-based learners to "build" (admittedly simple) concept descriptions from scratch and might form the basis of a useful feature-vector representation.

7 Summary

This paper proposed a method to apply feature-based learners to DL learning problems that uses a set of concepts to represent DL individuals as a Boolean feature vector. We presented two approaches to compute such sets of concepts from the training data, and showed empirically that the combination of the two can lead to improvements in the learning accuracy. Furthermore, we proposed areas of interesting future research.

References

[1] A. Borgida, R.J. Brachman, D.L. McGuinness, and L.Resnick. Classic: A structural data model for objects. In *Proceedings of SIGMOD-89*, 1989.

[2] W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, 1995.

[3] W. Cohen and H. Hirsh. Learning the classic description logic: Theoretical and experimental results. In *Principles of knowledge representation and reasoning : proceedings of the third international conference (KR '92)*, 1994.

[4] P. Domingos and M. Pazzani. On the optimality of the simple bayesian classifier under zero-one loss. *Machine Learning*, 29:103–130, 1997.

[5] J.U. Kietz and K. Morik. A polynomial approach to the constructive induction of structural knowledge. *Machine Learning*, 14, 1994.

[6] D. Kudenko. *Feature Generation for Sequence Categorization.* PhD thesis, Rutgers University, 1998.

[7] D. Kudenko and H. Hirsh. Feature generation for sequence categorization. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*, 1998.

[8] J. R. Quinlan. *C4.5: programs for machine learning.* Morgan Kaufmann, San Mateo, CA, 1994.