Improving Predictive Specificity of Description Logic Learners by Fortification

An C. Tran TCANVN@GMAIL.COM
Jens Dietrich J.B.DIETRICH@MASSEY.AC.NZ
Hans W. Guesgen H.W.GUESGEN@MASSEY.AC.NZ
Stephen Marsland S.R.MARSLAND@MASSEY.AC.NZ
School of Engineering and Advanced Technology
Massey University
Palmerston North 4442, New Zealand.

Editor: Cheng Soon Ong and Tu Bao Ho

Abstract

The predictive accuracy of a learning algorithm can be split into specificity and sensitivity, amongst other decompositions. Sensitivity, also known as completeness, is the ratio of true positives to the total number of positive examples, while specificity is the ratio of true negative to the total negative examples. In top-down learning methods of inductive logic programming, there is generally a bias towards sensitivity, since the learning starts from the most general rule (everything is positive) and specialises by excluding some of the negative examples. While this is often useful, it is not always the best choice: for example, in novelty detection, where the negative examples are rare and often varied, they may well be ignored by the learning. In this paper we introduce a method that attempts to remove the bias towards sensitivity by *fortifying* the model by computing and then including in the model some descriptions of the negative data even if they are considered redundant by the normal learning algorithm. We demonstrate the method on a set of standard datasets for description logic learning and show that the predictive accuracy increases.

Keywords: predictive accuracy, fortification, description logic learning, specificity, sensitivity

1. Introduction

Machine learning aims to generate general hypotheses to explain sets of observed examples. While there are various measurements that can be used to assess the predictive power of the learnt results, the most common measurement is still *predictive accuracy*, i.e., the number of correct predictions on the test set divided by the size of the test set. It is often more useful to split accuracy into two parts, of which there are several alternatives. In machine learning it is common to consider sensitivity (sometimes known as completeness) and specificity, which are respectively the ratio of true positives to the total number of positive examples, and the ratio of true negative to the total number of negative examples. An alternative way of evaluating our method would have been to use precision and recall as measures. However, specificity and sensitivity are more suitable for evaluation of our method as their combination provides a symmetric assessment of both positive and negative examples, particularly the true positives and true negatives. Precision and recall are asymmetric in that they focus only on the positive examples and mostly ignore the negative ones.

Methods of inductive logic learning that perform top-down search start from the most general statement (everything is positive) and specialise it. This will tend to overestimate the positive examples, implying a bias towards sensitivity. While this is often a valid assumption, it is not always the case. For example, consider the case of novelty detection (also known as one-class classification) where the class of interest is relatively rare, and possibly widely variable, within the data. In this case, it is common to learn a model of the normal (positive) data and argue that anything that does not fit that model is novel (Marsland, 2003). It can be used for applications such as machine fault detection, inspection applications, medical tests, and security. For novelty detection, a bias towards recall means insufficient specialisation, which will result in novel datapoints being classified as normal. However, care does need to be taken to avoid too much of a bias towards specificity, as this can produce large numbers of false positives (the system 'crying wolf'), which quickly lead to the system being ignored and the true positives being missed (Breznitz, 1984).

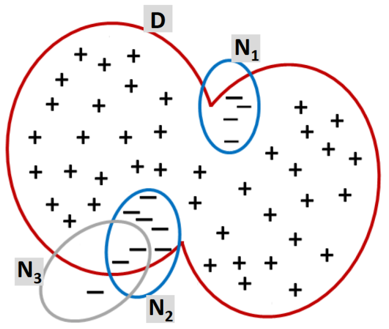
As another example, consider the actual purpose of an automatic social benefit fraud detection system. While the government certainly wishes to detect the fraud and make examples of fraudulent claimants, they will also wish to avoid the bad publicity that false positives generally cause: missing a few fraudsters does not cost votes, but bad publicity probably does. For this reason they may prefer their system to be biased away from false positives, i.e., to have a bias towards specificity.

In this paper we will introduce a method for reducing the bias towards sensitivity and demonstrate our method on description logic learning (a sub-field of inductive logic programming that uses description logics as the representation language (Muggleton, 1991; Lavrac and Dzeroski, 1994; Muggleton and De Raedt, 1994; Baader et al., 2005)).

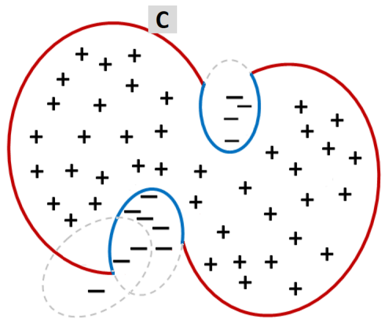
Typical top-down learning in description logic learning computes a description of the positive examples in the training set by successive specialisation of the most general concept. The negative examples are only used to assist in this specialisation. In this paper we describe a method that makes more use of the negative data, thus giving a more even weight to sensitivity and specificity. We use the same learning procedure as for the positive examples to learn about the negative examples separately, and use this knowledge to *fortify* the description of the positive examples. As well as wishing to avoid the sensitivity bias, another motivation is the expectation that knowing some rules about some of the negative examples can help to exclude false positives. We demonstrate our approach on a collection of standard datasets for description logic learning.

Figures 1 and 2 describe the basic idea. They show a learning problem with a training set of positive (pluses) and negative (minuses) examples. In Figure 1 D is a description of the positive data, while N_1, N_2 and N_3 are definitions of some of the negative data generated by some learning algorithm with their coverage are described in Figure 1(a). Notes that these negative definitions are *partial* in the sense that each covers only a limited set of the negative examples. They are called *counter-partial definitions* in this paper. A possible prediction model (definition) for the learning problem is $C \equiv D \sqcap \neg(N_1 \sqcup N_2)$, as depicted in Figure 1(b). N_3 is not included in the learning result as C is sufficient to accurately define the positive examples of the learning problem.

Figure 1: The creation of a prediction model for a learning problem. Pluses (+) and minuses (−) represent positive and negative examples in training set respectively.



(a) Generation of a prediction model: D is a description, N_1, N_2, and N_3 are *counter-partial definitions* produced by a learning algorithm.



(b) Prediction model after the combination and reduction: $C \equiv D \sqcap \neg(N_1 \sqcup N_2)$ describes all the positive examples and no negative examples.

In Figure 2 the same problem is shown, but with the test set also included (in grey). The predictive accuracy of the model is based on how well the model explains the training data and how representative the training set was of the underlying data. Of course, the learning algorithm does not have access to the test set, and so it can't be used in the learning; however, there may be a validation dataset that can be utilised; this will be discussed more later in the paper. Looking at the region around N_3 in the figure, it can be seen that there were datapoints missing in the training data that meant that the model has not been specialised far enough by choosing not to include N_3: there are a set of four negative examples in the test set that will be classified as positive. The idea of fortification is to identify that partial definitions like N_3 are required: including some redundant specialisations into the model can improve results.

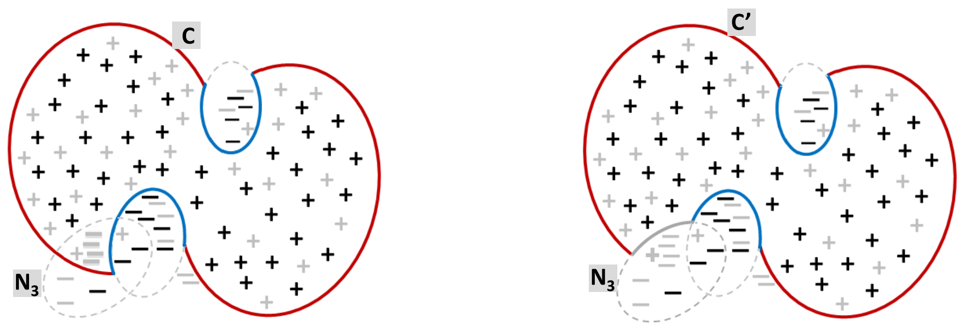
Figure 2 suggests that some additional specialisation is useful, but the question is how much? In the example, adding N_3 gained four negative examples at the cost of one positive example on the test set, but the gain may not be so high: Figure 3 shows two additional partial definitions of negative concepts (N_4 and N_5) which either do not help the model (N_4), or even make it worse (N_5). Thus, inappropriate choice of fortifiers can decrease the accuracy.

This overview of the problem has suggested that the requirements of a fortification strategy are to answer the following questions:

Q1. *How to generate candidates for fortification?* Fortification candidates are partial descriptions of the negative data that can remove negative examples covered by the learnt concept.

Q2. *How to choose the best candidates for fortification, and how many should there be?* Fortification candidates may have different effects on the fortification. Some descriptions may help to increase the specificity (true negatives) while some others do not. Moreover, some of them may effect the predictive accuracy badly, i.e. they decrease

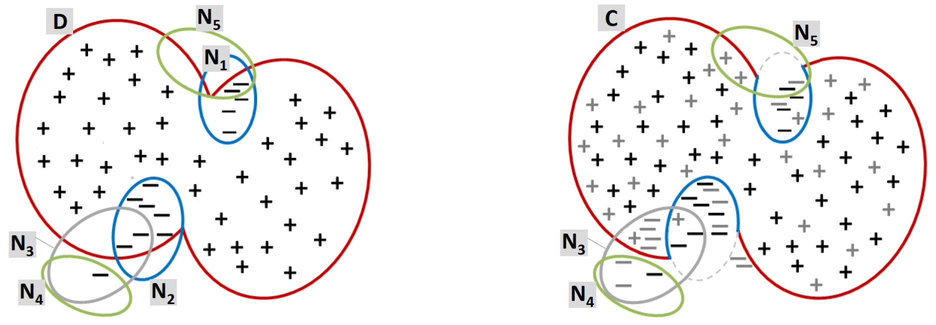
Figure 2: A prediction scenario of the prediction model in Figure 1. Grey pluses and minuses represent positive and negative examples in the test set.



(a) Prediction model without N_3. There are 4 negative examples covered by C (bold minuses) that are also covered by N_3. However, N_3 was removed by reduction as it is redundant in the training model.

(b) Prediction model with N_3. If N_3 is included in C, 4 covered negative examples in (a) can be removed from the C coverage. However, N_3 also excludes one positive example (bold plus) from C coverage.

Figure 3: Decrease of predictive accuracy caused by inappropriate fortification.



(a) A learning problem with several descriptions produced. An accurate definition for this learning problem can be constructed from D, N_1 and N_2 : $D \sqcap \neg(N_1 \sqcup N_2)$. N_3, N_4, N_5 can be considered as redundant in the prediction model.

(b) Using N_3 for fortification increased the specificity and accuracy, but using N_4 does not help. Moreover, using N_5 decreases the predictive accuracy as it decreases the sensitivity.

the predictive sensitivity (true positives). Further, in general, the more descriptions are used for fortification, the more chances there are to increase the specificity, at the potential cost of sensitivity.

In the rest of this paper we consider possible approaches to answer these questions, and demonstrate their use on a set of standard datasets for description logic learning.

2. Improving predictive specificity by fortification

2.1. Fortification candidates generation

In this section, we propose a method for generating candidates for fortification. One of the most important objectives of this method is that it must be general so that it can be used with common description logic learning algorithms.

A description logic learning problem uses a knowledge base and sets of positive and negative examples as input and produces one or more descriptions (called definitions) such that each of them *covers* all positive examples and no negative examples. An example e is *covered* by a concept C w.r.t. a knowledge base $\mathcal{K}$ if $\mathcal{K} \models C(e)$, where notation $\models$ denotes the instance checking task defined as follows:

Definition 1 (Instance check) *Given a description logic knowledge base $\mathcal{K}$, a concept C and an individual name a, a is an* instance *of C w.r.t. $\mathcal{K}$, denoted by $\mathcal{K} \models C(a)$, iff for any models $\mathcal{I}$ of $\mathcal{K}$, $a^{\mathcal{I}} \in C^{\mathcal{I}}$ holds.*

A description logic knowledge base is constructed based on a *vocabulary*, which consists of *concept names*, *role names* and *individuals*, using a set of concept *constructors*. A concept name is a common name for a set of objects, e.g. `Person, Mother, Father`. Roles are used to describe relationships between objects, e.g `hasChild, daughterOf`; and objects are constants that represent objects in the application domain, e.g. `tom`, `mary`. Concept constructors are used to create *complex concepts* (or *concepts* in short, or *descriptions*). A concept in description logic is a formal definition of a concept (or notation) in the application domain. For example, the complex concept $\texttt{Woman} \sqcap \exists\, \texttt{hasChild.Person}$ defines the concept *Mother* in the real world in which the notations $\sqcap$ and $\exists$ are the *intersection* and *existential restriction* concept constructors in description logics.

A typical knowledge base consists of two parts: i) a TBox, also called *terminology*, that contains the *terminological axioms*, and ii) an ABox that contain assertions of concepts and roles. Given two concepts C and D, a *terminological axiom* has the form of an *inclusion axiom* $C \sqsubseteq D$ or an *equality axiom* $C \equiv D$. An equality axiom (equivalence) is called a *concept definition* if its left hand side is an atomic concept. For example, $\texttt{Woman} \sqsubseteq \texttt{Person}$ is an inclusion axiom and $\texttt{Mother} \equiv \texttt{Woman} \sqcap\ \exists\texttt{hasChild.Person}$ is a concept definition. An assertion has the form of $C(a)$ (*concept assertion*) or $r(a, b)$ (*role assertion*), where a, b are individuals and r is a role name. For example, `Woman(mary)` is a concept assertion that states `mary` is a woman, and `hasChild(mary, tom)` is a role assertion that states `tom` is `mary`'s child.

More information about description logics can be found in (Baader et al., 2010). A description logic learning problem can now be defined as follows:

Definition 2 (Description logic learning problem) *Given a description logic knowledge base $\mathcal{K}$, sets of positive $\mathcal{E}^+$ and negative $\mathcal{E}^-$ examples and a noise value $\epsilon \in [0,1]$ (0 means no noise), a* description logic learning problem*, denoted by a structure $\langle \mathcal{K}, (\mathcal{E}^+, \mathcal{E}^-), \epsilon \rangle$, is to compute a set of descriptions $\mathcal{X} = \{C \mid |\text{cover}(\mathcal{K}, C, \mathcal{E}^+)| \geq (|\mathcal{E}^+| \times (1-\epsilon))$ and $|\text{cover}(\mathcal{K}, C, \mathcal{E}^-)| = 0\}$.*

Some description logics learning algorithms return only the best solution. This can be consider a special case of the learning problem described in Definition 2 where $\mathcal{X}$ contains only one definition. However, in practice, they can be modified to be fitted into the above definition by changing the stopping condition of the algorithms.

The basic idea of fortification is to fortify the learnt concepts using a redundant specification. This is essentially a further specialisation of the learnt concepts using a set of descriptions that are expected to be able to cover some negative examples in the test set (prediction). Therefore, the descriptions used for fortification are actually the definitions of negative examples generated by a class expression learning algorithm so that they can predict the negative examples in the test set. A fortification candidates learning problem can now be defined:

Definition 3 (Fortification candidates learning problem) *Given a class expression learning algorithm $\mathcal{L}$, a learning problem $LP = \langle \mathcal{K}, (\mathcal{E}^+, \mathcal{E}^-) \rangle$, a noise value $\epsilon \in [0,1]$ and a fortification coverage threshold $t \in [0,1]$, a* fortification candidates learning problem *is to find a description C such that $|cover(\mathcal{K}, C, \mathcal{E}^+)| \geq (|\mathcal{E}^+| \times (1-\epsilon))$ and $cover(\mathcal{K}, C, \mathcal{E}^-) = \emptyset$ and a set of descriptions $\mathcal{Y} = \{D \mid |cover(\mathcal{K}, D, \mathcal{E}^-)| \geq (|\mathcal{E}^-| \times (1-t))$ and $cover(\mathcal{K}, D, \mathcal{E}^+) = \emptyset\}$ using the learning algorithm $\mathcal{L}$.*

In Definition 3, C is the definition of the positive examples of the learning problem LP produced by $\mathcal{L}$ and the descriptions in $\mathcal{Y}$ are the candidates for the fortification of the learnt definition, named the *fortifying definitions*. Given a general class expression learning algorithm, a method for learning an additional set of *fortifying definitions* is described in Algorithm 1.

The learning problem LP is first passed to the class learning algorithm $\mathcal{L}$ to compute the set of definitions of the positive examples. Then, the set of positive and negative examples are swapped and passed to the algorithm $\mathcal{L}$ again together with the fortification threshold to compute a set of fortifying definitions. In this step, the fortification threshold is used as the noise percentage for the swapped learning problem (i.e. the learning problem with positive and negative examples being swapped). For example, if the fortification threshold is set to 10%, the learning algorithm can accept descriptions that cover 10% of the positive examples of the swapped learning problem, i.e. 10% of the negative examples in the original fortification candidates learning problem. Therefore, this is a constraint on the selection of the class expression learning algorithm used for learning the fortification candidates.

2.2. Fortification candidate scoring

The aim of fortification candidates scoring is to score the fortifying definitions according to their potential impact. The scores are then used to rank the fortifying definitions to support the selection of the most promising fortifying definitions. Therefore, the scoring implies the ranking of fortification candidates.

Algorithm 1: Fortification candidates learning – FCandLearning($\mathcal{L}, \mathcal{K}, \mathcal{E}^+, \mathcal{E}^-, \varepsilon, t$).

```
Input: a class expression leaning algorithm $\mathcal{L}$, a class expression learning problem
       $\langle \mathcal{K}, (\mathcal{E}^+, \mathcal{E}^-) \rangle$, a learning noise value $\varepsilon \in [0,1]$ (0 means no noise) and a
       fortification coverage threshold value $t \in (0,1)$.
Output: a definition $C$ and a set of fortifying definitions $\mathcal{X}$ produced by $\mathcal{L}$ such that
        $|cover(\mathcal{K}, C, \mathcal{E}^+)| \geq (|\mathcal{E}^+| \times (1-\varepsilon))$ and $|cover(\mathcal{K}, C, \mathcal{E}^-)| = 0$ and
        $\forall D \in \mathcal{X} : |\ cover(\mathcal{K}, D, \mathcal{E}^-)\ | \geq (|\mathcal{E}^-| \times (1-t))$ and $cover(\mathcal{K}, D, \mathcal{E}^+) = \emptyset$.
1 begin
      /* perform normal learning to learn the definition for pos. examples */
2     $\mathcal{Y} = \mathcal{L}(K, \mathcal{E}^+, \mathcal{E}^-, \varepsilon)$          /* find a set of definitions using $\mathcal{L}$ */
3     select a definition $C \in \mathcal{Y} \mid \forall E \in \mathcal{Y}$ : Accuracy($C$) $\geq$ Accuracy($E$)
      /* learn the fortifying definitions by reversing the sets of examples and
      using the threshold $t$ */
4     $\mathcal{X} = \mathcal{L}(K, \mathcal{E}^-, \mathcal{E}^+, t)$
5     return $(C, \mathcal{X})$
6 end
```

A fortifying definition is *promising* for the fortification if it can help to exclude the negative examples covered by the learnt concept in the test set (see Figure 4(b)). However, as the negative examples in the test set are unknown when the learning takes place, a method for predicting the influence of the fortifying definitions is needed. Fortifying definitions can be considered as prediction models for the negative examples. Therefore, scoring the fortifying definitions is basically the estimation of their predictive power. This motivates two methods for scoring the fortifying definitions. The first is to consider their training coverage as the factor that represents their predictive power. The second is to use another dataset, which is disjoint from the training and test sets, to evaluate their predictive power. A fortifying definition that gives a good prediction power on the validation dataset is expected to have a similar performance on the test set.

2.2.1. Training coverage scoring

In the first method, the training coverage of the fortifying definitions is used as their score. This information is the result of the training (learning) process and it is associated with every fortifying definition. The essential idea behind this method is that the training coverage potentially implies the generality and the predictive power of the fortifying definitions. The more general a fortifying definition is, the more chance for it to cover more negative examples in the test set in comparison with a less general definition.

The biggest advantage of this method is that the coverage is available from the learning process. Therefore, there is no extra computation required for scoring the fortifying definitions. In addition, high predictive power of the fortifying definitions (high coverage) may imply a higher chance for it to cover negative examples in the test set. The score of a fortifying definition C in this method is defined as follows:

$$\mathit{score}(C) = \frac{|\mathit{cover}(\mathcal{K}, C, \mathcal{E}^-)|}{|\mathcal{E}^-|}$$

This value is actually the training sensitivity of the fortifying definition. Therefore, this information is associated with each fortifying definition and thus it does not need to be re-computed. In addition, higher priority is given to short fortifying definitions in the case where the fortifying definitions have the same coverage on $\mathcal{E}^+$. In the top-down approach, shorter definitions are usually more general than longer ones. Therefore, this factor is used as a tiebreaker in scoring the fortifying definitions. The training coverage score can now be defined as:

Definition 4 (Training coverage score) *Let $LP = \langle \mathcal{K}, (\mathcal{E}^+, \mathcal{E}^-)\rangle$ be a learning problem, C a fortifying definition learnt from LP, max_fdlength the maximal length of all fortifying definitions learnt and length(C) a function that returns the length of the definition C. The score of the fortifying definition C based on the training coverage scoring method is defined as follows:*

$$\mathit{TCScore}(C) = \frac{|\mathit{cover}(\mathcal{K}, C, \mathcal{E}^-)| \times \mathit{max\_fdlength} + \mathit{length}(C)}{\mathit{max\_fdlength} \times (1 + |\mathcal{E}^-|)} \qquad (1)$$

The multiplication $\mathit{cover}(\mathcal{K}, C, \mathcal{E}^-)$ with $\mathit{max\_fdlength}$ in Equation (1) is to ensure the training coverage is used as the primary criteria and definition length as a tiebreaker when ranking (sorting) the fortifying definition. The division is used to scale the score into [0, 1] (the min-max normalisation).

2.2.2. Fortification validation scoring

This scoring method employs the idea of the cross-validation method in machine learning. In this method, a dataset, called the fortification validation dataset, is used to evaluate the fortifying definitions to select the most promising ones. This is a labelled dataset, i.e. the examples are labelled as positive or negative examples, and it is different from the training dataset.

The fortification validation dataset is used similar to a test set in cross-validation to measure the metrics that constitute the score of the fortifying definitions. The rationale behind this method is that we use the fortification validation dataset to simulate a real scenario for fortification to score the fortifying definitions accordingly to their performance on this dataset. Therefore, the best fortifying definitions in this step are assumed to have similar performance in real scenarios.

The main factor in the score of the fortifying definitions is the number of negative examples that the fortifying definition can remove from the set of negative examples covered by the learnt concept. The removal of negative examples from the coverage of the learnt concept helps to increase the predictive specificity, which is the main objective of our method. However, to avoid any decrease in the predictive accuracy, the fortifying definition score is reduced for each common positive example covered by both the fortifying definitions and the learnt concept. As in the scoring method based on the training coverage, the definition length is also used to favour shorter definitions. This factor has a lower weight so it is used as a secondary criteria in selecting the most promising fortifying definitions. The score of a fortifying definition can now be defined:

Definition 5 (Fortification validation score) *Let $LP = \langle \mathcal{K}, (\mathcal{E}^+, \mathcal{E}^-) \rangle$ be a learning problem, E_v a fortification validation dataset that consists of a set of positive examples $\mathcal{E}_v^+$ and a set of negative examples $\mathcal{E}_v^-$, C a fortifying definition and D a learnt definition of LP, length(C) a function that returns the length of the definition C and max_fdlength a maximal length of all fortifying definitions of LP. The score of the fortifying definition C based on the fortification validation method is defined as follows:*

$$\begin{aligned} FVScore(C) = \lambda \times & \frac{|cover(\mathcal{K}, C, \mathcal{E}_v^-) \cap cover(\mathcal{K}, D, \mathcal{E}_v^-)|}{|\mathcal{E}_v^-|} \\ -\alpha \times & \frac{|cover(\mathcal{K}, C, \mathcal{E}_v^+) \cap cover(\mathcal{K}, D, \mathcal{E}_v^+)|}{|\mathcal{E}_v^+|} \\ +\beta \times & \left(1 - \frac{length(C)}{max\_fdlength}\right) \end{aligned} \tag{2}$$

λ, α and β in Equation (2) are used to adjust the effect of the three factors on the fortification candidate's score. As the number of negative examples covered by a candidate is the main factor, λ often has highest value in comparison with α and β. In our implementation, we chose $\lambda = 1, \alpha = 0.5$ and $\beta = 0.1$ as the default values.

Besides the above scoring metrics, training coverage may be a useful supplement to assess the potential of the fortifying definitions, particularly when the fortification validation dataset is small.

2.2.3. Random score

In this scoring method, the fortifying definition scores are assigned randomly in $[0, 1]$. This method is used in the evaluation to illustrate the effect of the above proposed scoring methods.

2.3. Fortification cut-off point

In this section, a method is proposed to compute a cut-off point in the selection of fortification candidates. The cut-off point computation method is used to define a suitable number of fortifying definitions to fortify the learnt definition.

There may be several ways to estimate the number of fortifying candidates used for fortification. In this paper, we use the fortification validation datasets to empirically define the cut-off point. It is defined as the maximal number of fortifying definitions that are best used to fortify the learnt concepts on the fortification validation datasets. A formal procedure for identifying the cut-off point based on a fortification validation dataset is described in Algorithm 2.

2.4. Description logic leaning with fortification

Given the above algorithms, the algorithm for description logic learning with fortification can now be defined. Basically, the algorithm includes 5 steps and it is described in detail in Algorithm 3.

Algorithm 2: Fortification cut-off point – $\textsc{CutOff}(C, \mathcal{X}, \mathcal{K}, \mathcal{E}_v^+, \mathcal{E}_v^-)$.

Input: a learnt concept C, a set of fortifying definitions $\mathcal{X}$, a knowledge base $\mathcal{K}$ and a set of fortification validation dataset $(\mathcal{E}_v^+, \mathcal{E}_v^-)$.

Output: a cut-off point (the number of fortifying definitions will be used for fortification)

```
1  begin
2      $cp = cover(\mathcal{K}, C, \mathcal{E}_v^+)$                    /* pos. examples covered by C */
3      $cn = cover(\mathcal{K}, C, \mathcal{E}_v^-)$                    /* neg. examples covered by C */
4      $cutoff = 0$                                                     /* cut-off point */
5      foreach $fdef \in \mathcal{X}$ do
6          $cpf = cover(\mathcal{K}, fdef, \mathcal{E}_v^+)$            /* pos. examples covered by fdef */
7          $cnf = cover(\mathcal{K}, fdef, \mathcal{E}_v^-)$            /* neg. examples covered by fdef */
8          if $(|cpf \cap cp| \ / \ |\mathcal{E}_v^+|) \geq (|cnf \cap cn| \ / \ |\mathcal{E}_v^-|)$ then
9              $cutoff = cutoff + 1$
10     return $cutoff$
```

Algorithm 3: Fortification DL learning – $\text{FDL}\textsc{Learning}(\mathcal{L}, \mathcal{K}, \mathcal{E}^+, \mathcal{E}^-, \epsilon, t, \mathcal{E}_v^+, \mathcal{E}_v^-)$.

Input: A tuple of $\mathcal{L}, \mathcal{K}, \mathcal{E}^+, \mathcal{E}^-, \epsilon, t, \mathcal{E}_v^+$ as described in Algorithm 1 and a set of fortifying validation dataset $(\mathcal{E}_v^+, \mathcal{E}_v^-)$.

Output: a fortified learnt definition.

```
1  begin
2      $(C, \mathcal{X}) = \text{FCand}\textsc{Learning}(\mathcal{L}, \mathcal{K}, \mathcal{E}^+, \mathcal{E}^-), \epsilon, t)$       /* see Algorithm 1 */
3      score($\mathcal{X}$)                                                                  /* see Section 2.2 */
4      sort($\mathcal{X}$)        /* sort the candidates ascendingly based on their score */
5      $cut\text{-}off = \textsc{CutOff}(C, \mathcal{X}, \mathcal{K}, \mathcal{E}_v^+, \mathcal{E}_v^-)$      /* see Algorithm 2 */
6      $C = C \sqcap \neg \bigsqcup D_i, \forall D_i \in \mathcal{X}$ and $i \leq cut\text{-}off$      /* fortify C */
7      return $C$
8  end
```

3. Evaluation

3.1. Experimental design

Currently, there are many description logic learning algorithms developed such as the $\mathcal{AL}-$QuIn (Lisi and Malerba, 2003), YinYang (Esposito et al., 2004), DL-FOIL (Fanizzi et al., 2008) and a set of OWL class expression learning algorithms in the DL-Learner framework (Lehmann, 2010) such as CELOE, OCEL and ELTL. Amongst the existing description learning algorithms, the learning algorithms in the DL-Learning framework are the most recently developed and they are well evaluated and compared with the other algorithms. In general, these algorithms achieved good result in the evaluation. In addition, they are publicly available and they are implemented in an open framework. Their availability may benefit the implementation of our approach.

Therefore, in our evaluation, we chose CELOE to implement the fortification strategy. To evaluate the approach, we used 10-fold cross-validation on 15 learning problems. We chose the learning problems that vary in: i) size (background knowledge and examples), ii) length of definition (complexity), and iii) noise property. In addition, these learning problems are also commonly used in evaluation of sub-symbolic learning algorithms. A summary of the evaluation learning problems are given in Table 1. The first eleven learning problems were used to evaluate the DL-Learner framework and other learning algorithms. Their details can be found in (Lehmann, 2010). The remaining learning problems are described in (Tran et al., 2012).

The datasets of the learning problems are divided into 10 parts. In each fold, we used 8 parts for training and 1 part for testing. The remaining part is used by the FVScore strategy to score the fortification candidates. Therefore, for a fair evaluation of the FVScore strategy, we also performed an additional experiment that used 9 parts of the datasets for training. It is reported as the *Full training* result in the experimental results. This result is used to make an extra comparison for the FVScore strategy. A t-test of the null hypothesis (Tabachnick et al., 2001) is also used for testing the statistically significant difference between the the experimental results.

The algorithm package and all learning problems are available at `https://code.google.com/p/parcel-2013/`.

3.2. Experimental results

Table 2 shows the experimental result of fortification on the CELOE algorithm. To benefit the analysis of the experimental results, we divide them into 3 groups. The first group includes 7 learning problems that have low to medium complexity and the learning algorithm can find accurate solutions on the training set (i.e. without timeout). The second group contains high to very high complexity learning problems and the algorithm can also find accurate solutions on training sets without timeout. The third contains learning problems for which CELOE could not find accurate solution on the training set, i.e. a timeout occurred. The complexity of learning problems is approximately estimated based on the learnt definition length produced by CELOE: i) low: (0, 8]; ii) medium: (8, 15]; iii) high: (15, 20]; and iv) very high: longer than 20.

Table 1: Properties of the learning problems (learning prob.) used in evaluation. OPs and DPs are short for Object Properties and Data Properties respectively. Number of examples is given in the form positive/negative examples.

No	Learning prob.	Classes	OPs	DPs	Assertions			Examples
					Class	OP	DP	
1.	Moral	43	0	0	4646	0	0	102/100
2.	Forte uncle	3	3	0	86	251	0	23/163
3.	Poker	4	6	0	374	1080	0	4/151
4.	CarcinoGenesis	142	4	15	22,372	40,666	11,185	182/155
5.	Aunt	4	4	0	606	728	0	41/41
6.	Brother	4	4	0	606	728	0	43/30
7.	Cousin	4	4	0	606	728	0	71/71
8.	Daughter	4	4	0	606	728	0	52/52
9.	Father	4	4	0	606	728	0	60/60
10.	Grandson	4	4	0	606	728	0	30/30
11.	Uncle	4	4	0	606	728	0	38/38
12.	UCA1	30	4	11	300	200	200	73/77
13.	MUBus-1	25	0	12	2675	0	32110	383/2292
14.	MUBus-2	25	0	12	6314	0	75766	670/5643
15.	MUBus-3	25	0	12	17128	0	205534	1250/15877

For the low and medium complexity learning problems (Group 1 in the result table), CELOE achieved 100% of accuracy for most of them (5/7 learning problems). For the 5 learning algorithms on which CELOE achieved 100% accuracy, fortification did not reduce the predictive accuracy of any of them. For the two remaining learning problems, the improvement on predictive specificity and accuracy was not significant. There was also no decrease of the predictive sensitivity for the datasets in this group.

In the high and very high complexity learning problems without timeout (Group 2 in the result table), the effect of the fortification was clearer. On the Aunt dataset, TCScore improved both predictive specificity and accuracy significantly while the improvement made by FVScore was not significant. However, the predictive accuracy and specificity of the Cousin learning problem stayed unchanged. Note that CELOE got 100% predictive specificity on this learning problem. Therefore, no more improvement can be made. On the remaining dataset, Uncle, the improvement of the predictive specificity and accuracy was also not statistically significant.

Fortification worked best on learning problems where at least one learning algorithm timed out (Group 3 in Table 2). This group consists of 5 learning problems. All fortification strategies increased the predictive specificity and accuracy for all learning problems. The t-test at the 95% confidence level suggested that the increasing of both accuracy and specificity in 4 learning problems is statistically significant. There is no significant decrease of sensitivity on any dataset.

It is worth noting that the TCScore strategy decreased the predictive sensitivity of the CarcinoGenesis significantly. However, the predictive accuracy and specificity also increased significantly. This was caused by the increase in specificity was more significant than the decrease in sensitivity. The maximal confidence level such that the decrease of sensitivity was still significant was about 97.1% while those for the specificity and accuracy were about 99.8% and 98% respectively.

There were several impressive improvements by fortification in the experiments. For example, in the CarcinoGenesis learning problem, the specificity increased from 0.67 ± 2.11% to 13.46 ± 11.35% by TCScore and to 11.46 ± 8.24% and 8.46 ± 12.13% by FVScore respectively. Even for the learning problems in which CELOE achieved very high accuracy such as the Aunt learning problem, fortification also improved the predictive specificity and accuracy significantly. The predictive accuracy for this learning problem was increased from 95.25 ± 8.45% to 100% which was caused by the increase of the predictive specificity from 90.05 ± 17.07% to 100%.

Using more training data (full training result) produced different results in 7 learning problems. More training data did not always improve the predictive accuracy. It increased predictive accuracy in 5 learning problems and decreased predictive accuracy in 2 learning problems. However, changes of the predictive accuracy in the evaluation were not significant and it did not influence the t-test result.

Table 2: Fortification experimental results with CELOE (*means ± standard deviations of 10 folds*). The number following the learning problem name is the cut-off point. The convention of the t-test result with 95% confidence: i) bold values are statistically significantly higher than the corresponding *No fortification (No fort.)* values, ii) italic values are statistically significantly lower than the corresponding *No Fortification* values, iii) unformatted values are not statistically significantly different from the corresponding *No Fortification* values. The underlined values are the highest results within the fortification strategies results.

Metric	**Full training**		**No fort.**		**TCScore**		**FVScore**		**Random**	
Low and medium complexity learning problems without timeout (Group 1)										
Moral (0)										
Accu.[1]	100.00	±0.00	100.00	±0.00	100.00	±0.00	100.00	±0.00	100.00	±0.00
Spec.[2]	100.00	±0.00	100.00	±0.00	100.00	±0.00	100.00	±0.00	100.00	±0.00
Sens.[3]	100.00	±0.00	100.00	±0.00	100.00	±0.00	100.00	±0.00	100.00	±0.00
Forte (2)										
Accu.	98.86	±2.27	93.10	±11.95	93.10	±11.95	95.40	±7.96	93.10	±11.95
Spec.	100.00	±0.00	92.06	±13.75	92.06	±13.75	95.24	±8.25	92.06	±13.75
Sens.	95.83	±7.22	95.83	±7.22	95.83	±7.22	95.83	±7.22	95.83	±7.22
Poker-Straight (1)										
Accu.	100.00	±0.00	98.08	±3.85	100.00	±0.00	100.00	±0.00	98.08	±3.85
Spec.	100.00	±0.00	97.92	±4.17	100.00	±0.00	100.00	±0.00	97.92	±4.17
Sens.	100.00	±0.00	100.00	±0.00	100.00	±0.00	100.00	±0.00	100.00	±0.00
Brother (0)										
Accu.	100.00	±0.00	100.00	±0.00	100.00	±0.00	100.00	±0.00	100.00	±0.00
Daughter (0)										
Accu.	100.00	±0.00	100.00	±0.00	100.00	±0.00	100.00	±0.00	100.00	±0.00
Father (0)										

Continued on next page

1. Predictive accuracy = (|true positives| + |true negatives|) / (|positive examples| + |negative examples|)
2. Specificity = |true negatives|/|negative examples|, where true negatives are *uncovered negative examples.*
3. Sensitivity = |true positives|/|positive examples|, where true positives are covered positive examples.

Table 2 – continued

Metric	Full training	No fort.	TCScore	FVScore	Random
Accu.	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00
Grandson (0)					
Accu.	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00
High and very high complexity learning problems without timeout (Group 2)					
Aunt (1)					
Accu.	96.50 ±0.00	95.25 ±8.54	**100.00 ±0.00**	97.75 ±0.00	**100.00 ±0.00**
Spec.	95.50 ±9.56	90.50 ±17.07	**100.00 ±0.00**	97.50 ±7.91	**100.00 ±0.00**
Sens.	97.50 ±7.91	100.00 ±0.00	100.00 ±0.00	98.00 ±6.33	100.00 ±0.00
Cousin (0)					
Accu.	99.29 ±2.26	99.29 ±2.26	99.29 ±2.26	99.29 ±2.26	99.29 ±2.26
Spec.	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00
Sens.	98.57 ±4.52	98.57 ±4.52	98.57 ±4.52	98.57 ±4.52	98.57 ±4.52
Uncle (2)					
Accu.	95.83 ±6.80	93.33 ±10.97	93.10 ±11.95	95.40 ±7.96	93.10 ±11.95
Spec.	91.67 ±13.61	89.17 ±18.45	92.06 ±13.75	95.24 ±8.25	92.06 ±13.75
Sens.	100.00 ±0.00	97.50 ±7.91	95.83 ±7.22	95.83 ±7.22	95.83 ±7.22
Learning problems with timeout (Group 3)					
CarcinoGenesis (8)					
Accu.	53.61 ±2.45	54.01 ±1.02	**57.55 ±4.91**	**58.73 ±3.50**	55.49 ±3.00
Spec.	47.91 ±2.87	0.67 ±2.11	**13.46 ±11.35**	**11.46 ±8.24**	**4.54 ±5.33**
Sens.	87.71 ±3.97	99.44 ±1.76	*95.12 ±6.48*	98.89 ±2.34	98.89 ±2.34
UCA1 (1)					
Accu.	91.42 ±7.01	90.74 ±8.12	91.41 ±8.62	91.41 ±8.62	90.74 ±8.12
Spec.	83.39 ±13.55	89.64 ±11.60	91.07 ±11.93	91.07 ±11.93	89.64 ±11.60
Sens.	100.00 ±0.00	91.96 ±14.07	91.96 ±14.07	91.96 ±14.07	91.96 ±14.07
MUBus-1 (6)					
Accu.	53.61 ±2.45	54.31 ±3.28	**68.15 ±2.01**	**66.84 ±2.96**	**68.00 ±3.93**
Spec.	47.91 ±2.87	48.52 ±3.45	**64.66 ±2.25**	**63.13 ±3.33**	**64.48 ±4.70**
Sens.	87.71 ±3.97	89.02 ±3.89	89.02 ±3.89	89.02 ±3.89	89.02 ±3.89
MUBus-2 (4)					
Accu.	14.35 ±0.54	14.35 ±0.54	**35.45 ±6.99**	**35.55 ±9.82**	**33.31 ±6.74**
Spec.	4.18 ±0.60	4.18 ±0.60	**27.79 ±7.82**	**27.89 ±10.98**	**25.40 ±7.54**
Sens.	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00
MUBus-3 (4)					
Accu.	11.34 ±0.14	11.34 ±0.14	**40.18 ±0.95**	**40.18 ±0.95**	**26.16 ±5.83**
Spec.	4.36 ±0.15	4.36 ±0.15	**35.47 ±1.02**	**35.47 ±1.02**	**20.34 ±6.28**
Sens.	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00	100.00 ±0.00

4. Conclusion

Normal top-down inductive logic programming techniques have an inherent bias towards sensitivity in their learning approach. While this can be a positive thing, there are also cases where a more equal treatment of specificity and sensitivity is needed, or even where specificity should be favoured. In this paper we have introduced a method for doing precisely this, which we call *fortification*. We generate partial descriptions of negative examples using the standard learning algorithms, and then score these partial definitions in order to choose which should be added to the model.

We have also provided a method for generating fortifying definitions, two methods for scoring fortifying definitions and a strategy to compute the cut-off point. Overall, the experimental results in Table 2 show that our approach improved the predictive accuracy of most learning problems in the experiment. This means that the predictive accuracy was increased and the increase of the predictive specificity was higher than the decrease of the sensitivity (if any).

The fortification experimental results are promising: 9/30 results were above the baseline, i.e. they are statistically significantly better than the prediction result without fortification at 95% confidence. It is worth noting there were 12/30 results in which the predictive specificity is 100%, i.e. they are impossible to improve the results.

Therefore, this approach is beneficial for class expression learning. The experimental results suggest that our approach not only ensured a balanced trade-off between the predictive specificity and the predictive sensitivity, but also achieved promising results. The gained predictive specificity was higher than the lost predictive sensitivity in most learning problems. Therefore, the predictive accuracy was increased.

In the two fortification scoring strategies, the TCScore strategy does not require additional computation and dataset to score the fortification candidates. Therefore, it is suitable for learning problems that do not have enough additional data for scoring. This strategy is also suitable for learning problems where the training data contains enough information to describe the scenario.

Beside the two proposed fortification candidate scoring strategies, Figure 3 also motivates another method for scoring the fortification candidates. The method is based on the *overlap* between the learnt concept and the fortifying definitions. Here, the concept *overlap* is used with the following meaning: Two descriptions are overlapped if they cover some common instances. On the right of Figure 3 is can be seen that if a fortifying definition does not overlap with the learnt concept, it will not influence the predictive specificity even if it covers many negative examples. For example, in that figure, N_3 can help to exclude some negative examples covered by the learnt definition C as it covers some common negative examples (overlapped) with C, but N_4, even though it covers some negative examples, does not influence the predictive specificity as it covers negative examples that are not covered by the learnt definition (i.e. not overlapped). This method can use both the ABox and TBox of the knowledge base for scoring the candidates and we plan to investigate it further in the future.

References

Franz Baader, Ian Horrocks, and Ulrike Sattler. Description logics as ontology languages for the semantic web. *Mechanizing Mathematical Reasoning*, pages 228–248, 2005.

Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider. *The description logic handbook: Theory, implementation and applications.* Cambridge University Press, 2010.

Shlomo Breznitz. *Cry wolf: The psychology of false alarms.* Lawrence Erlbaum Associates Hillsdale, NJ, 1984.

Floriana Esposito, Nicola Fanizzi, Luigi Iannone, Ignazio Palmisano, and Giovanni Semeraro. Knowledge-intensive induction of terminologies from metadata. *The Semantic Web–ISWC 2004*, pages 441–455, 2004.

Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. DL-FOIL concept learning in Description logics. *Inductive Logic Programming*, pages 107–121, 2008.

Nada Lavrac and Saso Dzeroski. *Inductive logic programming: Techniques and applications.* New York, Ellis Horwood, 1994.

Jens Lehmann. *Learning OWL Class Expressions.* AKA Akademische Verlagesellschaft, 2010.

Francesca A. Lisi and Donato Malerba. Ideal refinement of descriptions in $\mathcal{AL}$-log. *Inductive Logic Programming*, pages 215–232, 2003.

Stephen Marsland. Novelty detection in learning systems. *Neural Computing Surveys*, 3: 157–195, 2003.

Stephen Muggleton. Inductive logic programming. *New generation computing*, 8(4):295–318, 1991.

Stephen Muggleton and Luc De Raedt. Inductive logic programming: Theory and methods. *The Journal of Logic Programming*, 19:629–679, 1994.

Barbara G. Tabachnick, Linda S. Fidell, and Steven J. Osterlind. Using multivariate statistics. *Allyn and Bacon Boston*, 2001.

An C. Tran, Jens Dietrich, Hans W. Guesgen, and Stephen Marsland. Two-way parallel class expression learning. *Journal of Machine Learning Research-Proceedings Track*, 25: 443–458, 2012.