Iterative Ensemble Classification for Relational Data: A Case Study of Semantic Web Services

Andreas Heß and Nick Kushmerick

Computer Science Department, University College Dublin, Ireland
{andreas.hess, nick}@ucd.ie

Abstract. For the classification of relational data, iterative algorithms that feed back predicted labels of associated objects have been used. In this paper we show two extensions to existing approaches. First, we propose to use two separate classifiers for the intrinsic and the relational (extrinsic) attributes and vote their predictions. Second, we introduce a new way of exploiting the relational structure. When the extrinsic attributes alone are not sufficient to make a prediction, we train specialised classifiers on the intrinsic features and use the extrinsic features as a selector. We apply these techniques to the task of semi-automated Web Service annotation, a task with a rich relational structure.

1 Introduction

In a variety of learning tasks, a number of related views on a problem exist. Several approaches have been suggested for how to make use of these different views for improving accuracy by using more than one classifier. These approaches include ensemble learning, co-training, and link-based classification for relational data. For the latter, the second view for a given object is derived from the objects with which it is related. This view is also known as the *extrinsic* view (as opposed to the *intrinsic* view). The intuition is that an object's class can be predicted from information about related objects.

Iterative classification algorithms for relational data have been proposed to address this setting (e.g. [10]). These algorithms are based on the assumption that, as we classify objects, these predictions have an influence on related objects. Initially, a preliminary set of predictions is made based only on the intrinsic features. Then, predicted labels are fed back to the classifier. Classification accuracy is expected to improve over time, as the labels of related objects become known with more certainty.

In this paper, we present a generic framework for iterative classification of relational data. We combine the intrinsic and the extrinsic view using an ensemble of two classifiers. The first classifier is based on a set of static features: the intrinsic features associated with the object or related objects. We denote the static classifier as A.

The second classifier B is derived from the labels of related objects. In some classification tasks, the dynamic view may be sufficient to classify an object, so

we train B on all the objects and combine the results of the A and B classifier by voting. In more challenging tasks, the dynamic view alone may be insufficient, so we use the extrinsic features to select a specialised intrinsic classifier C trained only on the subset of training instances with the same extrinsic features. This is a novel approach, and we will explain it in detail in the next section.

The task of semi-automatically generating semantic annotation for Web Services serves as a motivating example and case study. The basic idea behind our approach is that Web Service descriptions are structured objects, and that there are dependencies between the components of these objects that we can exploit. In previous work [8], we used a Bayesian setting and evaluated it on Web forms. In this paper, we present an iterative classification algorithm similar to the one introduced by Neville and Jensen in [10].

2 Semantic Web Services

We have built a tool that learns from Web Services with existing semantic annotation and provides the user with annotation recommendations in an interactive fashion. We present this application called ASSAM[1] in greater detail in [7].

The predictions made by an algorithm that classifies parts of Web Services do not have to be perfectly accurate to be helpful. In fact, the classification task is quite hard, because the domain ontologies can be very large. But for that reason it is very helpful for a human annotator if he or she could choose only between a small number of ontological concepts rather than from the full domain ontology. In previous work [8] we have shown that the category of a service can be reliably predicted, if we stipulate merely that the correct concept be one of the top few (e.g., three) suggestions.

Before describing our approach in detail, we begin with some terminology. Our terminology is a bit different from the terminology used in Web Services and Semantic Web description languages, but by introducing this terminology we do not intend to advocate a new standard. Instead we believe that our approach is generic and independent of the actual format used for the semantic Web Service description. However, in the application that we have built the input consists of Web Services in the Web Service Description Language (WSDL [3]) format, and the final annotations can be exported as OWL-S [12].

A WSDL Web Service corresponds in many ways to a set of methods in any programming language. The equivalent of a method is called an *operation*. Operations can have input, output and fault *messages*. This corresponds to a method's parameters, return values and possible exceptions. A single parameter in WSDL is called a *part* in a message. It is also possible to make use of *complex types* defined in XML Schema. This corresponds to a structure in a typical programming language.

We use the word *category* to denote the semantic meaning of the service as a whole. For example, the Amazon Web Service belongs to the "book selling" category. The category ontology corresponds to a *profile hierarchy* in OWL-S.

[1] Automated Semantic Service Annotation with Machine learning

We use the word *domain* to denote the semantic meaning of an operation. For example, a particular operation in the Amazon Web Service called "QueryAuthor" would belong to the "Query books by author name" domain. An operation in WSDL usually maps to an *atomic process* in OWL-S, but there is no direct relation between the domain of an operation and OWL-S, as atomic processes are only characterised through their inputs, outputs, preconditions and effects. One could think of the *domain* as describing the semantics of the preconditions and effects, but we currently do not use the domain annotation for our OWL-S export.

We use the word *datatype* to denote the semantic type of a single variable. This applies to message parts as well as complex types. This usage of the word datatype is consistent with how the word is used when describing a property in an ontology, but should not be confused with low-level syntactic datatypes such as "integer" or "string". For example, the input parameter for the "QueryAuthor" operation mentioned above would be of the datatype "author".

For information retrieval or classification tasks the objects that are classified or searched are also referred to as documents. When we use the word *document*, we mean the Web Service as whole as represented by its WSDL. We use *document part* to denote an object within the Web Service that we want to classify: Operations, input and output messages, XML schema types etc. are document parts or *object components*.

3 Iterative Ensemble Classification

Following [10], we distinguish between *intrinsic*, *static extrinsic* and *dynamic extrinsic* features. In contrast to [10], we assume that the object graph has more than one connected component. In our example, we have one connected component for each Web Service.

Static intrinsic features are attributes that are inherent to the component that is to be classified. We denote the static intrinsic features for a component j as a_j. Dynamic extrinsic features derive from the relationship between different components. We denote dynamic extrinsic features as b_j. In our Web Services dataset, we use the class labels of linked components as dynamic extrinsic features. Static extrinsic features are based on other components, but do not change when more classifications are made. Static extrinsic features are static intrinsic features of linked components. When we speak of extrinsic features in the remaining sections, we usually mean dynamic extrinsic features if not otherwise stated.

Initially, when the adjacent objects are unlabeled, the dynamic extrinsic features are unknown. In the first pass classifications are made based on the intrinsic features only. In subsequent iterations we include extrinsic features that are based on the class labels predicted in the previous round. The classification process is repeated until a certain termination criterion (e.g. either convergence or a fixed number of iterations) is met.

Our iterative algorithm differs in some points from Neville and Jensen's algorithm. In their approach, one single classifier is trained on all (intrinsic and extrinsic) features. In a variety of tasks, ensembles of several classifiers have been shown to be more effective (e.g., [4]). For this reason, we train two separate classifiers, one on the intrinsic features ("A") and one on the extrinsic features ("B"), and vote together their predictions. Another advantage of combining the evidence in that way is that the classifier cannot be mislead by missing features in the beginning when the extrinsic features are yet unknown, because the classifier trained on the extrinsic features is simply not used for the first pass.

Specialised classifiers. This split approach, where the A and B classifiers are combined using voting, works best, when both views would by themselves be sufficient to make a prediction. This intuition is the same as behind Co-Training [1]. However, in more challenging prediction tasks, the extrinsic view alone gives additional evidence, but is not sufficient to make a prediction. In our Web Services dataset, this is the case on the datatype level. The fact that a Web Service belongs to the "book selling" category is by itself not sufficient to classify its input parameters, but the information is still useful. We therefore introduce a second mode for incorporating the extrinsic features: We train a set of classifiers on the intrinsic features of the components, but each of them is only trained on the subset of the instances that belong to one specific class. In our setting, we train specialised classifiers on the datatypes level on all instances that belong to the same category[2].

To formally specify these specialised classifiers, we first define what we mean by a specialisation. We take a very generic view: the k'th specialisation for component j is formally just a set Φ_j^k of instances of component j that satisfy the specialisation.

In our example, if component j represents a parameter, then one Φ_j^k might be "all parameters of services that are in the 'book selling' category."

We require that all specialisations for j be mutually exclusive and exhaustive. Let $\phi_j^k(b_j) = 1$ if there exists an instance in Φ_j^k with extrinsic features b_j, and 0 otherwise. Mutually exclusiveness and exhaustivity means simply that $\forall_j : \sum_k \phi_j^k(b_j) = 1$.

The specialisation Φ_j^k is used to define a set of training data for learning a specialised classifier C_j^k. Specifically, C_j^k is trained on the a_j features for every training instance of component j in Φ_j^k.

To avoid biasing the algorithm too strongly, we still combine the results of the C classifier with the A classifier in each iteration. For each level we use either the B or C classifier, but not both. We chose the C method for the datatypes and the B method for the category and the domain.

[2] To avoid over-specialisation, these classifiers are actually not trained on instances from a single category, but rather on instances from a complete top-level branch of the hierarchically organised category ontology. Note that this is the only place where we make use of the fact that the class labels are organised as an ontology, and we do not do any further inference.

Algorithm 1 Iterative Ensemble Classification

for $j \leftarrow 1 \ldots n$ **do**
  $\alpha_j \leftarrow$ predictions made by A_j with features a_j
end for
$(b_1, \ldots, b_n) \leftarrow (\alpha_1, \ldots, \alpha_n)$
for a fixed number of iterations **do**
  for $j \leftarrow 1 \ldots n$ **do**
    $\beta_j \leftarrow$ predictions made by B_j or C_j with features b_j
    $\pi_j \leftarrow \alpha_j * \beta_j$
  end for
  $(b_1, \ldots, b_n) \leftarrow (\pi_1, \ldots, \pi_n)$
end for
return $(\pi_1, \ldots, \pi_n)$

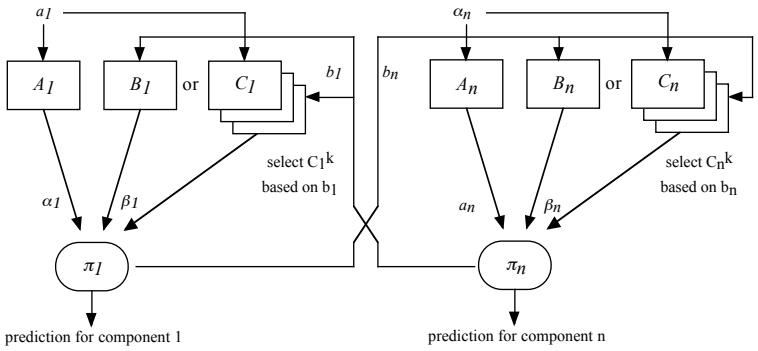

Fig. 1. Iterative Ensemble Classification.

Algorithm 1 explains our iterative classification ensemble, and Fig. 1 illustrates its classification phase. In Fig. 1 and algorithm 1, let n denote the number of object components; a_j the static and b_j the dynamic features; A_j and B_j the intrinsic and extrinsic classifiers, and C_j^k denotes a specialised classifier as discussed above. α_j denotes the predictions made by A_j and β_j the predictions made by the extrinsic classifier B_j resp. C_j^k.

4 Related Work

Several variations of link-based text classification have been proposed. The classical task here is to classify web pages connected by hyperlinks. Lu and Getoor address this task in [9]. They cast the problem as predicting the classes of objects or nodes in a graph considering links between objects or edges in the graph. However, in their model only one flavour of objects exists: the classes for all objects are drawn from only one ontology. Iterative classification algorithms were also

used by Chakrabarti [2] before. We already mentioned the approach by Neville and Jensen [10].

Although not mentioned explicitly, our C approach is used by Finn and Kushmerick [6] for information extraction. They use a two-level learning approach. At the first level (L1), each token in a document is predicted either as the start of a text fragment to extract, or not. L1 is trained on all available data. A second classifier (L2) is then used to identify the end of a fragment, given that its start was predicted by L1. L2 is trained on just the subset of the training data, namely a window of tokens near field starting positions. In our terminology, the L2 classifier is a specialised classifier C, and the L1 predictions are used to decide when to invoke L2, just as we use extrinsic features to select the appropriate C.

Clearly, our approach is related to ensemble learning (e.g. [4]). We have used ensemble learning in a non-iterative fashion for classifying the category of a web service in our earlier paper [8]. Another well-known way to combine the output of multiple classifiers is stacked generalization [14]. We already mentioned Co-Training [1] shortly; however, note that our data is fully labeled.

Related Work in the Web Services Area. As far as we are aware, we were the first to propose the use of machine learning for the task of semi-automatically annotating Semantic Web Services. However, this task could also be viewed as a schema matching problem. Patil et al [11] are working on matching XML schemas to ontologies. They use a combination of lexical and structural similarity measures. They assume that the user's intention is not to annotate similar services with one common ontology, rather they also address the problem of choosing the right domain ontology among a set of ontologies. The Web Service search engine Woogle [5] allows not only for keyword queries, but also for searching structurally similar operations.

5 Parameters

In an iterative feed-back setting such as ours, a sensible choice of all parameters is both crucial and rewarding: An improvement in one of the classifiers can lead to an improvement in other classifiers as well. In some respects the parameters we discuss here are relevant not only to our specific setting, but are common to all iterative classification algorithms. The important parameters are:

1. Structure: Which layers of the model are used for feedback and how?
2. Voting: How are the results of the different views combined?
3. Termination criterion: What number of iterations is optimal?
4. Inference, Constraints: Use information from the class ontologies?
5. Features: Use binary or continuous features?
6. Classification algorithms: Which one to use?

Structure. One might expect that this structure is inherent to the problem, but this is only partly true. In our Web Services dataset, we have identified

three layers of semantic metadata, as described in section 1: the category of the service as a whole, the domain of the service's operations and the datatypes of its input and output parameters. This leaves a variety of possible choices how the feed-back is actually performed. One might say that the category of a service is determined by the domain of its operations, and the domain of the operations is determined by the datatypes of its input and output parameters. But it is also possible to see it the other way around: The datatypes of an operation's parameters are dependent on the operation's domain, which is dependent on the service's category. Instead of choosing either a top-down or bottom-up approach, our framework also allows for modeling both dependencies at the same time.

Preliminary tests showed that it is not guaranteed that using as many extrinsic features as possible automatically yields the best results. It is best to choose between either the B or C classifier based on the nature of the extrinsic view: If the extrinsic view alone provides enough information to make a reliable prediction, it is best to combine the extrinsic and intrinsic view in an ensemble way. If the extrinsic view alone is not sufficient for a reliable classification, it is better to use the C classifier. There are, however, two points that are important when considering the C classifier: Using a set of specialised classifiers means that there are as many classifiers as there are distinct class labels in the extrinsic view and each classifier is only trained on a subset of the available training data. To have a set of many classifiers can not only be computationally expensive, it also means that the number of training instances for each classifier is only a fraction of the total number of training instances.

We used *static* extrinsic features on the domain and datatype level by incorporating text from children nodes: Text associated with messages was added to the text used by the operations classifier, and text associated with elements of complex types were added to the text used by the datatype classifier classifying the complex type itself. Note that this appears to contradict our earlier results [8], where we claimed that simply adding text from child nodes does not help. In [8], we where classifying the category level only, and the bag of words for the domain and datatype classifiers consisted of text for all operations/datatypes in that service. In our present experiment, the classifier for the operations and datatypes classify one single operation or parameter only, and in this case it apparently helps to add more text.

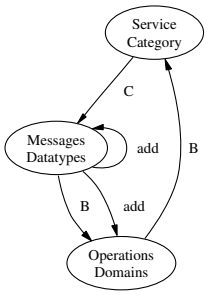

Fig. 2. Feedback structure.

Given these arguments for and against the different modes of incorporating the extrinsic evidence and some empirical preliminary tests, we eventually decided to use the setup shown in Fig. 2, where "add" denotes the use of static extrinsic features by incorporating text.

Combining Static and Dynamic Features. One of the fundamental differences between our approach and the original algorithm by Neville and Jensen is the

way the different views—i.e., the intrinsic and extrinsic features—are combined. Neville and Jensen's approach is to add the extrinsic features as additional attributes in a stacking-like fashion and train a single classifier on all features. Neville and Jensen tested their algorithm on a low-dimensional dataset. On our dataset, however, this way of combining the evidence of the static and dynamic features did not work. In section 6 we also report results for a setup that combines static and dynamic features in this "non-ensemble" way. We conclude that this is due to fact that both the static and the dynamic features are quite high-dimensional, and that the initial noise in the dynamic features strongly affects the overall performance of the classifier.

Number of Iterations. The iterative classification is not guaranteed to converge, and it is also not guaranteed that more iterations always produce better results, as the predictions might get stuck in a local maximum. We tried several fixed numbers of iterations in preliminary experiments, and eventually decided to terminate the classification process after five iterations.

Inference and Constraints. In our Web Services dataset as well as in many other relational learning tasks, the class labels are not arbitrary but rather they are organised as an ontology. This ontology could be a simple hierarchical structure, but it could also convey more complex constraints. In our datatypes ontology, it would be possible to model restrictions on the relations between complex types and their elements. For example, a complex type that represents a book in the datatypes ontology can only have elements that are assigned a class label that represents a property of book in the ontology.

In the present setting we do not make use of constraints that can be inferred from the ontology, with one exception. As mentioned in section 3, we use the hierarchy in the services ontology to prevent over-specialisation of the C classifiers. However, we believe that the use of inference that can be derived from the ontologies where the class labels are drawn from could greatly increase classification accuracy. We will investigate this in future work.

Features. The well-known range of choices for representing words in a bag-of-words model as features in a classifier includes binary features, term frequency and TFIDF. In our case, we also have to choose a suitable representation for the extrinsic features. We can choose between a single feature denoting the most common linked class or use a feature vector with one element per possible class. When using the latter, the feature vector can be binary or frequency based. It is also possible to use the complete ranked distribution or the top n of classes as output by the classifiers instead of just the absolute predictions.

In our experiments, using a single feature for the extrinsic view does not make much sense, as we are using a separate classifier for the dynamic view. In preliminary experiments, using the complete ranked distributions did not work. We believe that this is due to the fact that in conjunction with the base classification algorithm we used and the one-vs-all setting, the absolute confidence

values in the ranked distribution are too close together, and that then combining the distributions for all linked objects introduces too much noise.

We decided to use a binary feature vector with one attribute per possible linked class for the dynamic features. For the static features, we decided to use binary features as well. In section 3, we already described another method of incorporating dynamic extrinsic features, the specialised classifiers, and the possibility to add terms from linked document parts as a way to use static extrinsic features.

Classification algorithms. Due to the large number of classifiers and evaluations in our iterative framework, it is desirable to use a fast classification algorithm, especially if the intended application requires user interaction and low response-times are necessary.

In our implementation we are using the Weka library of classifiers [13] off the shelf. For our experiments we chose the HyperPipes algorithm in a one-against-all configuration, as it offered a good tradeoff between speed and accuracy. We did not use the SMO algorithm because it turned out to be computationally much more expensive.

6 Evaluation

We evaluated our algorithm using a leave-one-service-out methodology. We compared it against a baseline classifier with the same setup for the static features, but without using the dynamic extrinsic features.

To determine the upper bound of improvement that can be achieved using the extrinsic features, we tested our algorithm with the correct class labels given as the extrinsic features. This tests the performance of predicting a class label for a document part when not only the intrinsic features but also the dynamic features (the labels for all other document parts) are known. Following [10], we refer to this upper bound as the "ceiling".

We also compared it against a non-ensemble setup, where the extrinsic features are not added using a separate classifier but rather are just appended to the static features. Classification is then done with a single classifier. This setup closely resembles the original algorithm proposed by Neville and Jensen. Again, the same set of static features was used.

In the evaluation we ignored all classes with one or two instances, such as occurred quite frequently on the datatype level. The distributions are still quite skewed and there is a large number of classes. There are 22 classes on the category level, 136 classes on the domain level and 312 classes on the datatype level. Our corpus consists of 164 services with a total of 1138 annotated operations and 5452 annotated parameters.

Fig. 3 shows the accuracy for categories, domains and datatypes. As mentioned earlier, in mixed-initiative scenario such as our semi-automated ASSAM tool, it is not necessary to be be perfectly accurate. Rather, we strive only to ensure that that the correct ontology class is in the top few suggestions. We

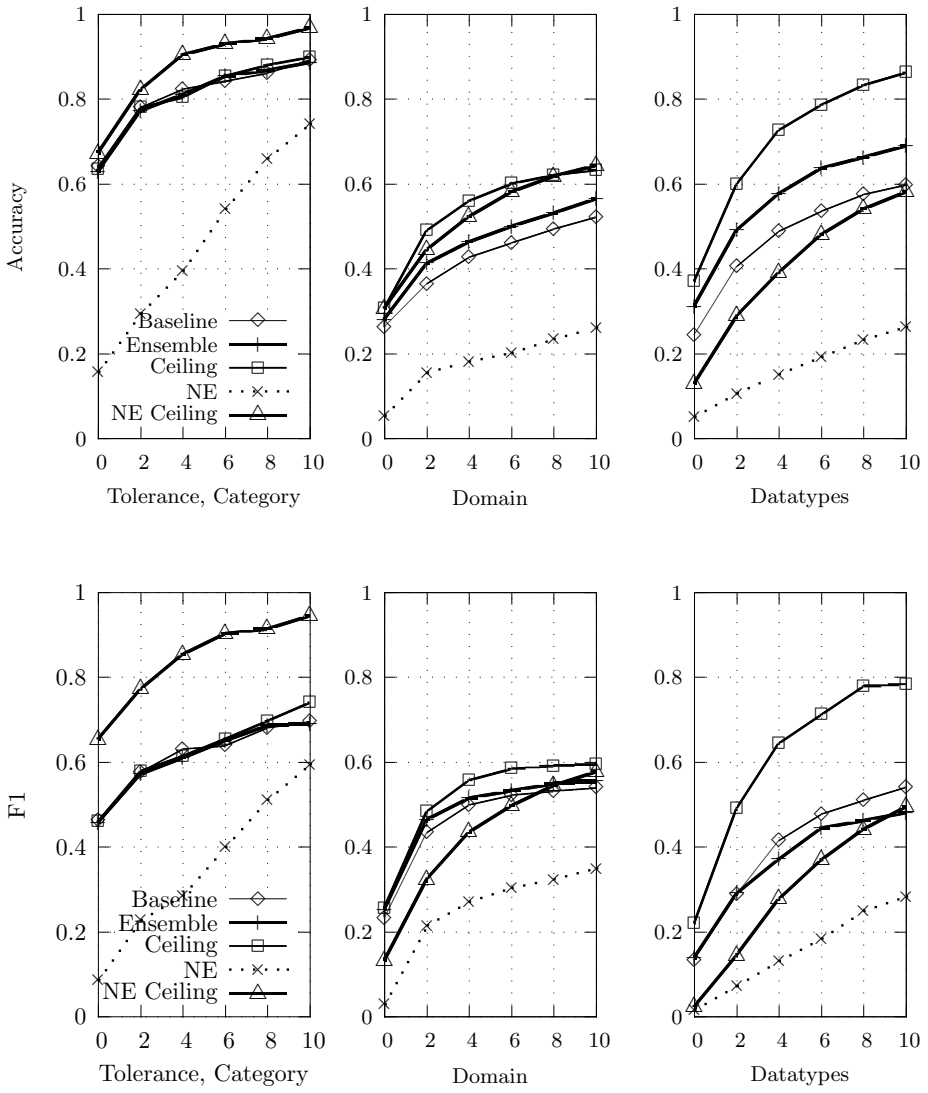

Fig. 3. Accuracy and macroaveraged F1 on the three ontologies.

therefore show how the accuracy increases when we allow a certain tolerance. For example, if the accuracy for tolerance 9 is 0.9, then 90% of the time, the correct prediction is within the top 10 of the ranked predictions. Note that on the datatypes level, macroaveraged F1 (as shown in Fig. 3) for the iterative ensemble is worse than for the baseline while accuracy is still above the baseline. This is

due to the fact that at this level of tolerance the baseline has a higher recall for many small classes. Macroaveraged precision is still higher for the iterative ensemble.

Note that on the category level incorporating the additional evidence from the extrinsic features does not help. In fact, for some tolerance values the ceiling accuracy is even worse than the baseline.

Also, we could not achieve good results with the non-ensemble setup (denoted as "NE"). This setup scored worse than the baseline. For the datatypes, even the ceiling accuracy (denoted as "NE Ceiling") was below the baseline.

On the datatype level, our algorithm achieves 31.2% accuracy, where as the baseline scores only at 24.5%. Thus, our algorithm improves performance by almost one third. The overall performance might be considered quite low, but due to the high number of classes it is a very hard classification problem. Given that in two of three cases the user has to choose only between 10 class labels rather than between all 312 labels in the datatype ontology we are still convinced that this could save a considerable amount of workload. We evaluated the statistical significance of the accuracy improvement of our algorithm with a Wilcoxon Matched-Pairs Signed-Ranks Test. Our algorithm performs significantly better ($p < 0.001$).

On the domain level, our approach increases the accuracy for exact matches from 26.3% to 28%. This is statistically significant ($p < 0.05$) according to a Wilcoxon Test. For both significance tests we performed a 20-fold random split.

7 Conclusion

We have demonstrated two extensions to prior research on iterative classification algorithms for relational tasks. First, we have shown that in a high-dimensional environment such as text classification it is better to use separate classifiers for the intrinsic and extrinsic features and vote their predictions rather than to use one classifier trained on both types of features. Second, we have introduced a new mode for relational classification where the extrinsic features serve as a selector for a specialised intrinsic classifier trained on a subset of the training instances. We have applied these techniques to the domain of semi-automatically annotating Semantic Web Services, and shown that it outperforms conventional approaches.

Future Work. As mentioned, we believe that incorporating domain knowledge in the classification process can increase the overall performance. In future work we will investigate the use of inference over the ontology and how it can help the machine learning algorithm. In particular, currently we ignore the hierarchical structure of the class labels, but we will explore whether this structure can be exploited to improve prediction accuracy. On the application side, we will continue to improve our WSDL annotator tool. We believe that tools such as ours are needed to bootstrap the Semantic Web and Semantic Web Services, and that machine learning is a very helpful technique that should be applied here.

Acknowledgments. This research was supported by grants SFI/01/F.1/C015 from Science Foundation Ireland, and N00014-03-1-0274 from the US Office of Naval Research. We thank Martina Naughton, Jessica Kenny, Wendy McNulty and Andrea Rizzini for helping us manually annotating the corpus of Semantic Web Services used in our evaluation. Thanks to Rinat Khoussainov and Thomas Gottron for helpful discussions.

References

1. Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *COLT: Proceedings of the Workshop on Computational Learning Theory*, San Francisco, 1998. Morgan Kaufmann.
2. Soumen Chakrabarti, Byron E. Dom, and Piotr Indyk. Enhanced hypertext categorization using hyperlinks. In Laura M. Haas and Ashutosh Tiwary, editors, *Proceedings of SIGMOD-98, ACM International Conference on Management of Data*, pages 307–318, Seattle, US, 1998. ACM Press, New York, US.
3. Erik Christensen, Francisco Curbera, Greg Meredtih, and Sanjiva Weerawarana. *Web Services Description Language (WSDL) 1.1.* World Wide Web Consortium, 2001.
4. T. G. Dietterich. Ensemble methods in machine learning. In *Lecture Notes in Computer Science*, volume 1857.
5. Xin Dong, Alon Havey, Jayant Madhavan, Ema Nemes, and Jun Zhang. Similarity search for web services. In *Proceedings of the 30th VLDB Conference*, Toronto, Canada, 2004.
6. Aidan Finn and Nicholas Kushmerick. Multi-level boundary classification for information extraction. In *Proceedings of the European Conference on Machine Learning*, Pisa, 2004.
7. Andreas Heß, Eddie Johnston, and Nicholas Kushmerick. ASSAM: A tool for semi-automatically annotating semantic web services. Submitted to the 3rd International Semantic Web Conference, 2004.
8. Andreas Heß and Nicholas Kushmerick. Learning to attach semantic metadata to web services. In *Proceedings of the 2nd International Semantic Web Conference*, 2003.
9. Qing Lu and Lise Getoor. Link-based classification. In *Int. Conf. on Machine Learning*, Washington, D.C., August 2003.
10. Jennifer Neville and David Jensen. Iterative classification in relational data. In *Proceedings of the AAAI 2000 Workshop Learning Statistical Models from Relational Data*, pages 42–49. AAAI Press, 2000.
11. Abhijit Patil, Swapna Oundhakar, Amit Sheth, and Kunal Verma. Meteor-s web service annotation framework. In *Proceedings of the 13th International World Wide Web Conference*, New York, USA, May 2004.
12. The DAML services coalition. Owl-s 1.0. White Paper, 2003.
13. I. H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools with Java Implementations.* Morgan Kaufmann, San Francisco, 1999.
14. David H. Wolpert. Stacked generalization, 1990.