Arbiter Meta-Learning with Dynamic Selection of Classifiers and its Experimental Investigation

Alexey Tsymbal[1], Seppo Puuronen[1], Vagan Terziyan[2]

[1] University of Jyväskylä, P.O.Box 35, FIN-40351 Jyväskylä, Finland
{alexey, sepi}@jytko.jyu.fi
[2] Kharkov State Technical University of Radioelectronics, 14 Lenin av.,
310166 Kharkov, Ukraine
vagan@kture.cit-ua.net

Abstract. In data mining, the selection of an appropriate classifier to estimate the value of an unknown attribute for a new instance has an essential impact to the quality of the classification result. Recently promising approaches using parallel and distributed computing have been presented. In this paper, we consider an approach that uses classifiers trained on a number of data subsets in parallel as in the arbiter meta-learning technique. We suggest that information is collected during the learning phase about the performance of the included base classifiers and arbiters and that this information is used during the application phase to select the best classifier dynamically. We evaluate our technique and compare it with the simple arbiter meta-learning using selected data sets from the UCI machine learning repository. The comparison results show that our dynamic meta-learning technique outperforms the arbiter meta-learning significantly in some cases but further profound analysis is needed to draw general conclusions.

1 Introduction

Currently electronic data repositories are growing quickly and contain huge amount of data from commercial, scientific, and other domain areas. The capabilities for collecting and storing all kinds of data totally exceed the development in abilities to analyze, summarize, and extract knowledge from this data. Data mining is the process of finding previously unknown and potentially interesting patterns and relations in large databases [5].

A typical data mining task is to predict an unknown value of some attribute of a new instance when the values of the other attributes of the new instance are known and a collection of instances with known values of all the attributes is given. The collection of the instances with the known attribute values is treated as a training set for a learning algorithm that derives a logical expression, a concept description, or a

classifier, that is then used to predict the unknown value of the attribute for the new instance [2].

Recently, several parallel and distributed computing approaches have been proposed. The main aim behind these approaches is to search techniques that are suitable for huge amounts of data that cannot efficiently be handled by main-memory-based learning algorithms. It has been shown in [2-4] that parallel and distributed processing provides the best hope of dealing with large amounts of data.

In this paper, we consider an approach that uses classifiers learned on a number of data subsets in parallel and that selects for each new instance the best classifier dynamically. This approach reduces and limits the amount of data inspected by every single learning process and provides dynamic classifier selection, increasing the classification speed without significant losses in the classification accuracy or even with an improvement of the accuracy. The approach is based on the arbiter meta-learning technique [2]. We discuss this technique in chapter 2. In chapter 3, we consider our technique for the dynamic selection of classifiers. In chapter 4, we propose a combination of our dynamic classifier selection technique with the arbiter meta-learning. In chapter 5, we present results of our experiments with the approach, and chapter 6 concludes with a brief summary and further research topics.

2 Arbiter Meta-Learning Technique

In [2-4] the arbiter meta-learning technique was proposed for the parallel integration of multiple classifiers. *Meta-learning* encompasses the use of learning algorithms to learn how to integrate results from multiple learning systems. The approach includes data reduction as a solution to the scaling problem. The whole data set is partitioned into smaller subsets, and learning algorithms are applied on these subsets. This is followed by a part of the learning phase, which combines the learned results. It has been proposed to speed up the process by running the learning programs in parallel using multiple processors. It was shown in experiments that the accuracy would not suffer in such a scheme, as one may presume, in comparison with learning from the entire data set [2].

This meta-learning technique is independent of the underlying learning algorithms employed. Furthermore, it does not require a platform that is especially constructed for parallel processing. Thus, the meta-learning approach is intended to be scalable as well as portable and extensible [2]. In this chapter we discuss both the one-level and multi-level arbiter meta-learning briefly.

2.1 One-Level Arbiter Meta-Learning

An *arbiter* is a classifier that is trained to resolve disagreements between the base classifiers. An arbiter is generated using the same learning algorithm that is used to train the base classifiers. In the arbiter technique, the training set for the arbiter is a subset of the union of the training sets for the base classifiers for which the arbiter is

formed. The arbiter training instances are selected to form a particular distribution of the union set [2]. One scheme to select the instances to the arbiter training set, i.e. the *selection rule*, is to pick up the instances for which none of the classes gathers a majority classification [4]. We shall use this selection rule in our experiments in the form presented in [2]. Thus, the predictions of the learned base classifiers determine the subset of the training set that constitutes the arbiter's training set in the learning phase. When a new instance is classified in the application phase, first the base classifiers and the arbiter generate their predictions. Then an *arbitration rule* generates the final prediction using the predictions made by the base classifiers and the arbiter. The generation of an arbiter has much in common with the boosting technique [12] that also filters training instances to train the base classifiers. The approach can be considered as a particular case of the stacked generalization framework [18] that integrates the results of the base classifiers by a trained meta-level classifier.

One arbitration rule that is used to derive the final classification is the following: return the prediction with the majority of occurrences given by the base classifiers and the arbiter. It was used in experiments in [2], and we shall also use it in our experiments in this paper. Preference will be given to the arbiter's choice in the case of a tie. This arbitration rule is based on the most widely used voting principle.

2.2 Arbiter Tree

The one-level arbiter meta-learning is not always able to achieve the same level of accuracy as a global classifier. In [2-4] a hierarchical (multi-level) meta-learning method called *arbiter tree* was considered.

An *arbiter tree* is a hierarchical structure composed of arbiters that are computed in a bottom-up, binary-tree way and it can be generalized to arbiter trees of higher orders. Let there be k base level classifiers. The lowest level arbiters are initially trained using the outputs of a pair of the base classifiers. Thus $k/2$ arbiters are generated at the lowest level. At the second level, arbiters are trained using the outputs of the lowest level arbiters, and recursively at the higher levels of arbiters. For k subsets and k base classifiers there are $log_2(k)$ levels of arbiters generated [2].

When a new instance is classified by the arbiter tree in the application phase, predictions flow from the leaves to the root of the tree. First, each of the leaf classifiers, i.e. base classifiers, produces its classification of the new instance. From a pair of predictions and the classification of the parent arbiter, a classification is produced as a result of the arbitration rule. This process is applied at each higher level until a final classification is produced at the root of the arbiter tree.

It is noted in [2] that in a distributed environment the union sets need not be formed at one processing site. Rather, one can classify each subset by transmitting each learned classifier to each site, which is used to scan the local data set labeled with the predictions of the classifier. Each classifier is a computational object that is far smaller in size than the training set from which it is derived. To reduce the complexity of learning the arbiter trees, the size of the training sets for the arbiters can

be purposely restricted to be no larger than the training sets used to compute the base classifiers. Thus, the parallel processing time at each level of the arbiter tree is approximately equal throughout the tree. In several experiments in [2], it was shown that the classification accuracy does not suffer too much from such a restriction. Without the restriction the size of the training set for an arbiter would be comparable to the size of the entire training set at the root level. Experiments in [2] showed in some cases an accuracy improvement of this multi-level arbiter tree approach over the one-level techniques, which generally could not maintain the accuracy of the global classifier trained on the whole data set.

3 Dynamic Selection of Classifiers

In [9, 10] we proposed a technique for the dynamic integration of classifiers. In [14,15,17] we considered some medical applications of this technique. This technique is based on the assumption that each base classifier gives the best prediction inside certain subdomains of the whole application domain, i.e. inside its competence areas. The main problem in the technique is to estimate the competence areas of the base classifiers in a way that helps the dynamic selection of classifiers for each new instance. Our goal is to use each base classifier just in the subdomain where it is the most reliable one and thus to achieve overall results that can be considerably better than those achieved using the best individual classifier alone. In this chapter, we describe our dynamic selection technique briefly.

The proposed meta-classification framework consists of two levels. The first level contains base classifiers, while the second level contains a combining algorithm that predicts the error for each of the base classifiers. In the training phase we derive the information about the performance of the base classifiers calculating for each training instance the classification errors. These errors can be binary (i.e. a classifier gives a correct/incorrect classification) or they can be numerical values representing corresponding misclassification costs. This information about the base classifier performance is then stacked (as in stacked generalization [18]) and is later used together with the initial training set as meta-level knowledge to estimate the classification error for a new instance.

The information about the base classifier performance is derived using the cross-validation technique [1,7] for learned base classifiers and directly calculated for heuristic non-learned classifiers.

In [9,10] we considered the weighted nearest neighbor classification (WNN) as the meta-level classifier. The WNN simplifies the training phase of the composite classifier because with the WNN there is no need to train referees, only the base classifier performance matrix is needed. In the application phase, the nearest neighbors of a new instance are found out among the training instances and the performances of the corresponding base classifiers are used to predict the performance of each base classifier. In this calculation, we sum up the corresponding performance values of each classifier using weights that depend on the distances between a new instance and its nearest neighbors in the training set.

The use of WNN as the meta-level classifier is based on the assumption that each classifier has certain subdomains in the space of instance attributes, where it is more reliable than the other classifiers. This assumption is supported by the experiences that base classifiers usually work well not only in certain points of the domain space, but in certain subareas of the domain space. The performance of a classifier usually changes gradually from one instance to another near-by instance. Thus if a classifier does not work well with the instances near the new instance, then it is quite probable that it will not work well with the new instance.

4 Application of Dynamic Integration of Classifiers with Arbiter Meta-Learning

In chapter 2, we discussed the arbiter meta-learning technique, which uses a kind of voting, *arbitration rule*, to integrate multiple classifiers both in the one-level approach and in the arbiter tree approach. The voting technique, however, has several shortcomings (see for example [13]). From our point of view the most important shortcoming is that the voting technique is unable to take into account the local expertise. When a new instance is difficult to classify, then the average classifier will give a wrong prediction, and the majority vote will more probably result in a wrong prediction. In that case, one can advise the use of another arbitration rule as in [4]: if the majority of the base classifiers disagrees with the arbiter, then use the arbiter's prediction. However, the arbiter can itself make mistakes in a situation when the majority of the base classifiers does not make them.

In order to improve the meta-learning, we propose the use of our dynamic classifier selection (chapter 3) as the arbitration rule. This helps the selection of the base classifiers and arbiters so that they will be used in the subarea of their expertise. In this chapter, we consider both one-level and multi-level arbiter meta-learning combined with our dynamic classifier selection, and a general algorithm for the arbiter meta-learning technique with the dynamic classifier selection.

4.1 One-Level Arbiter Meta-Learning with Dynamic Classifier Selection

We suggest that the training phase remains bottom-up and the base classifiers are trained first. Then, an arbiter is formed using the same procedure (selection rule), as in the simple arbiter meta-learning [4]. Next, the meta-level information about the performances of the base classifiers and the arbiter is derived comparing their predictions with the actual classifications included in the training set.

Our application phase is top-down. First, the performances of the base classifiers and the arbiter are estimated for a new instance to be classified. Then, a base classifier or the arbiter with the best estimated performance is launched to make the final classification. Thus, during the application phase only one classifier is launched (not all as in the simple arbiter meta-learning).

4.2 Arbiter Tree with Dynamic Classifier Selection

The first level arbiters are initially trained using the outputs of the pairs of the base classifiers, and, recursively, the arbiters at the higher levels are trained using the outputs of the subtrees. An example of an arbiter tree that uses the dynamic selection of classifiers is shown in Figure 1.

The training phase is similar to the simple arbiter tree (chapter 2) with added collection of the classification error information. Let there be initially four training subsets T_1 - T_4. First, the four classifiers C_1 - C_4 are generated in parallel for each T_1 - T_4. Then the union of T_1 and T_2 is classified by C_1 and C_2, and these predictions are used to form the training set for the arbiter A_{12} using the selection rule. Next, the arbiter is trained using the same learning algorithm, and it is used to classify the instances in the union of T_1 and T_2. Based on the classification results of C_1, C_2, and A_{12} the error information of these classifiers on the union of the training sets T_1 and T_2 is collected for dynamic selection DS_{12}. The subtree rooted with DS_{34} is trained using the same procedure as for DS_{12} using the union of T_3 and T_4. After that the union of T_1, T_2, T_3, and T_4 is classified by the subtrees rooted with DS_{12} and DS_{34}. The root arbiter $A_{12,34}$ is formed and the dynamic selection classifier $DS_{12,34}$ is generated, completing the arbiter tree.

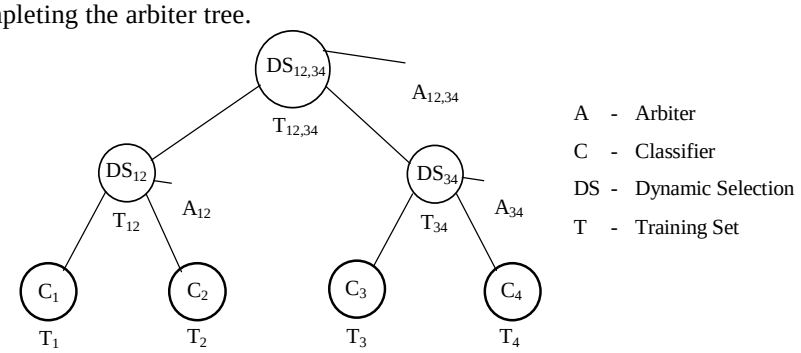

Fig. 1. A simple arbiter tree with the dynamic selection of classifiers

The application phase, however, is completely different from the application phase of the simple arbiter tree technique. In our approach, this phase is a top-down procedure using the attribute values of a new instance as input. First, for a new instance the topmost dynamic selection classifier $DS_{12,34}$ estimates the performances of the subtrees DS_{12}, DS_{34}, and the arbiter $A_{12,34}$. The subtree or the arbiter with the best performance estimate is selected. If the root arbiter $A_{12,34}$ is selected, it is used to make the final classification. Otherwise, the dynamic selection classifier of the selected subtree (either DS_{12} or DS_{34}) is used to estimate the performances of the lower level classifiers and the arbiter. One of the classifiers or the arbiter with the best performance is selected to make the final classification. Thus, during the application phase, at worst as many classifiers are launched as there are levels in the tree. In the simple arbiter tree approach all the classifiers and arbiters must be launched during this phase.

```
Node current node at the tree being learned
fs the first subset for the current node
nos number of subsets for the current node
arity the arity of the tree being learned
Subtree_1...Subtree_arity subtrees or base classifiers
  of the current node
TS_i i-th training subset for meta-learning
C_i base classifier trained on the i-th subset
Arbiter arbiter classifier at the current node
Instance a new instance to be classified
Best_Classifier a subtree, classifier or arbiter
  selected according to the local performance info

Procedure Dynamic_Arbiter_Meta-Learning(Node, fs, nos)
Begin

   If nos=arity
   then
      {train base classifiers}
      Subtree_1=Train(C_fs);
               . . .
      Subtree_arity=Train(C_fs+arity-1);
   else
      {develop the tree further}
      Dynamic_Arbiter_Meta-Learning(Subtree_1,
      fs, nos/arity);
               . . .
      Dynamic_Arbiter_Meta-Learning(Subtree_arity,
      fs+(arity-1)*(nos/arity), nos/arity);
   endif
   {Derive the performance info for
   Subtree_1...Subtree_arity on TS_fs,...,TS_fs+nos-1}
   Evaluate(Subtree_1, fs, nos)
               . . .
   Evaluate(Subtree_arity, fs, nos)
   {Generate a set for the arbiter using
   a selection rule}
   Selection_Rule(fs, nos);
   {train the arbiter}
   Train(Arbiter);
   {Derive the performance info for Arbiter
   on TS_fs,...,TS_fs+nos-1}
   Evaluate(Arbiter, fs, nos);

End
```

```
Function Dynamic_Arbiter_Tree_Application(Node,
   Instance) returns class of Instance
Begin

   If Node is a base classifier or arbiter
   then return Node.Classify(Instance)
   else
      {Select the best classifier using the performance
      info in near-by instances to Instance}
      Best_Classifier=Select_Classifier(Subtree1, ...,
      Subtreearity, Arbiter)
      return Dynamic_Arbiter_Tree_Application
             (Best_Classifier, Instance);
   endif

End
```

Fig. 2. A general algorithm of the arbiter meta-learning with the dynamic classifier selection

Two recursive algorithms for the learning and application phases of the arbiter meta-learning with the dynamic classifier selection are presented in Figure 2. The procedure *Dynamic_Arbiter_Meta-Learning* implements the learning phase of the technique, while the *Dynamic_Arbiter_Tree_Application* implements the application phase. The goal of the learning phase is to build an arbiter tree recursively and to collect the performance information at each node. The goal of the application phase is to use the performance information at each node to select an appropriate classifier for the final classification. These procedures can be applied for the generation and use of trees with different arity and different number of subsets. For example, to train a binary arbiter tree on 32 subsets it is necessary to set *arity=2* and to execute *Dynamic_Arbiter_Meta-Learning*(*Root*, 1, 32), where *Root* denotes the root node of the arbiter tree being learned. When an *Instance* is classified, *Dynamic_Arbiter_Tree_Application*(*Root*, *Instance*) should be executed.

5 Experimental Evaluation

In this chapter, we present an experimental evaluation of the arbiter meta-learning with the dynamic classifier selection. We compare it with the simple arbiter meta-learning. First we present the experimental setting and then describe the data sets and the results of our experiments.

In our experiments, we use the common technique used in evaluating the accuracy of a learning algorithm, the cross-validation technique [1,7]. In the cross-validation technique, all the instances of the training set are divided randomly into *v* approximately equal sized partitions that are usually called *folds*. Each classifier being evaluated is then trained on *v-1* folds *v* times and tested *v* times on the fold

being held-out. We use 10-fold cross-validation, and the averages of 10-fold cross-validation runs are presented in Figure 3. We measure statistical significance of the difference of averages by using the paired differences *t*-test based on train/test cross-validation splits.

In our experiments, we use the C4.5 inductive learning algorithm taken from the machine learning library in C++ (MLC++)[6]. The experiments are implemented within the MLC++ framework. In all the experiments, we use only the Euclidean distance metrics (the standard squared-distance metrics) for finding the nearest neighbors. We vary the number of equi-sized subsets of the training data from 2 to 32 ensuring that the subsets are disjoint with the same distribution of instances of each class. Stratified sampling was made so that each training subset represents a good but smaller model of the entire training set. No pairing strategy for the tree generation and no restriction on the size of the data set for training an arbiter is used, because the main goal of the experiments is to compare our technique with the simple arbiter meta-learning. The experiments with one-level classification trees (*n*-ary trees on *n* training subsets) and with binary multi-level classification trees are conducted on four datasets from the UCI machine learning repository: three MONK's problem datasets donated by Sebastian Thrun and the Tic-Tac-Toe Endgame dataset donated by David W. Aha [8].

The MONK's problems are a collection of three artificial binary classification problems over the same six-attribute discrete domain (a1,…,a6). All MONK's datasets contain 432 instances without missing values, representing the full truth tables in the space of the attributes. The "true" concepts MONK-1, MONK-2, and MONK-3 underlying each MONK's problem are given by: (a1=a2)or(a5=1) for MONK-1, exactly two of {a1=1, a2=1, a3=1, a4=1, a5=1, a6=1} for MONK-2, and (a5=3 and a4=1)or(a5<>4 and a2<>3) for MONK-3. MONK-3 has 5% additional noise (misclassifications) in the training set. The MONK's problems were the basis of the first international comparison of learning algorithms [16].

The Tic-Tac-Toe Endgame dataset encodes the complete set of possible board configurations at the end of tic-tac-toe games, where "x" is assumed to have played first. The target concept is "win for x" (i.e., true when "x" has one of 8 possible ways to create a "three-in-a-row"). The dataset contains 958 instances without missing values, each with 9 attributes, corresponding to tic-tac-toe squares and taking on 1 of 3 possible values: "x", "o", and "empty".

The experimental results are presented in Figure 3. The four meta-learning techniques described above are analyzed: the one-level simple arbiter meta-learning (*Arbiter*), the one-level arbiter meta-learning with the dynamic classifier selection (*Dynamic*), the simple binary arbiter tree (*Arbiter Tree*), and the binary arbiter tree with the dynamic classifier selection (*Dynamic Tree*). The plotted accuracies in the left charts are the averages of 10-fold cross-validation runs. The accuracy of the global classifier is plotted as "the number of subsets=1" that means that the learning algorithm was applied to the whole training set to produce the baseline accuracy results for comparisons. The charts on the right present the learning curves for the datasets. Each point of a learning curve is an average over 70 runs with random training subsets of appropriate size.

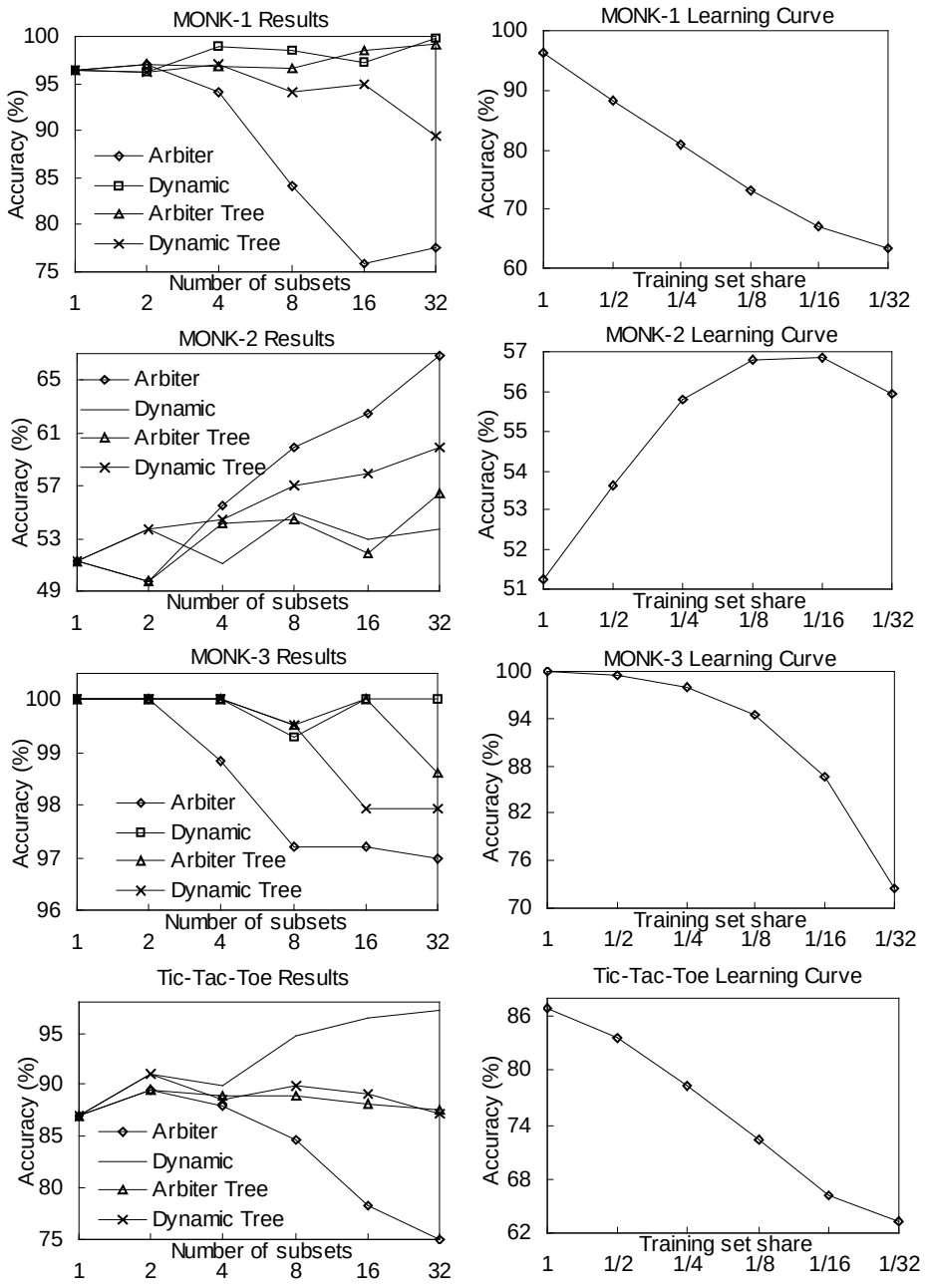

Fig. 3. Experimental results on the 4 data sets from the UCI machine learning repository

Our experimental results support the findings and conclusions made in [4]. All the meta-learning strategies do show a consistent improvement in the classification accuracy over the base classifiers trained on a subset of the training data. This can be seen comparing the resulting charts with corresponding learning curves. Our experimental results show also that both the one-level meta-learning schemes (*Dynamic* and *Arbiter*) and the hierarchical meta-learning schemes (*Dynamic Tree* and *Arbiter Tree*) are often able to sustain the same level of accuracy as a global classifier trained on the entire data set. Thus meta-learning over data partitions can maintain or even boost the accuracy of a single global classifier under certain circumstances. For example, it was done by *Dynamic* on the Tic-Tac-Toe dataset, where the best base classifier on 32 subsets has 73% accuracy, and the global classifier 87% only, but the one-level dynamic arbiter meta-learning classifier has 97% accuracy! It can be seen from the experimental results that this is a very common result. Our experimental results confirm that maximal parallelism can be effectively exploited by the meta-learning over disjoint data partitions without a substantial loss of accuracy. Hierarchically learned classifiers can work better than a single layered meta-learning under certain circumstances. For example, on the MONK-1, MONK-3, and Tic-Tac-Toe datasets the *Arbiter Tree* works significantly better than the *Arbiter*, and on the MONK-2 dataset the *Dynamic Tree* works significantly better than the *Dynamic*.

One can see from the experimental results that under some circumstances our dynamic meta-learning techniques are better than the corresponding simple meta-learning techniques. For example, on the MONK-1, MONK-3, and Tic-Tac-Toe datasets the *Dynamic* is significantly better than the *Arbiter*, and on the MONK-2 dataset the *Dynamic Tree* is significantly better than the *Arbiter Tree*. However, it is not clear under what circumstances our dynamic meta-learning techniques will be better and further studies are needed to gain understanding of these circumstances.

6 Conclusion

We considered the use of the dynamic classifier selection in the arbiter meta-learning. We evaluated our technique and compared it with the simple arbiter meta-learning using selected data sets from the UCI machine learning repository. We showed that our technique often results in the better classification accuracy than the simple arbiter meta-learning. However, multiple experiments on large real-world databases are needed to compare these meta-learning techniques and to find out the conditions under which our technique is better.

There are also many open questions related to our dynamic meta-learning technique which require further experiments. In the above arbiter tree algorithm, the training set grows at each level of the tree, and at the root level a dynamic selection classifier is trained using the whole training set. Unfortunately, such an algorithm is computationally expensive. In [4] it was shown in experiments that the overall accuracy would not suffer significantly if the size of training sets is restricted to the size

of the initial subsets at all nodes. However, this is still open for our approach and it should be checked with multiple experiments.

Another interesting question is "How to pair classifiers in order to construct the best tree?". In [4] Chan proposed three strategies of pairing classifiers: a random pairing, and pairings with the maximal and with the minimal possible arbiter training sets. The experiments in [4] showed that the pairing with the maximal arbiter training sets usually gives better results. We expect that this pairing will also be better in our approach resulting in more diverse arbitrated classifiers. In [10] we have shown that the bigger base classifier diversity results usually in the better coverage by the composite classifier and, consequently, the higher accuracy. The above pairing will generate more compact trees but requires significantly more computer time than the random pairing.

Another open question is "What is the optimal order of the arbiter tree with the dynamic classifier selection?". Experiments in [4] showed that the higher order trees are generally less accurate than the lower order ones. It can be explained by the fact that in the higher order trees it is more difficult for arbiters to arbitrate among bigger number of classifiers. However, this question is also still open for our approach and needs further research.

Acknowledgments: This research is partly supported by the Academy of Finland and the COMAS Graduate School of the University of Jyväskylä. Alexey Tsymbal is thankful to the Kharkov State Technical University of Radioelectronics that allowed him to work in Jyväskylä during the preparation of this article. We would like to thank the UCI machine learning repository of databases, domain theories and data generators for the data sets and the machine learning library in C++ for the source code used in this study.

References

1. Aivazyan, S.A.: Applied Statistics: Classification and Dimension Reduction. Finance and Statistics, Moscow (1989)
2. Chan, P., Stolfo, S.: On the Accuracy of Meta-Learning for Scalable Data Mining. Intelligent Information Systems, Vol. 8 (1997) 5-28
3. Chan, P., Stolfo, S.: Toward Parallel and Distributed Learning by Meta-Learning. In: Working Notes AAAI Work. Knowledge Discovery in Databases (1993) 227-240
4. Chan, P.: An Extensible Meta-Learning Approach for Scalable and Accurate Inductive Learning. PhD Thesis, Columbia University (1996)
5. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R.: Advances in Knowledge Discovery and Data Mining. AAAI/ MIT Press (1997)
6. Kohavi, R., Sommerfield, D., Dougherty, J.: Data Mining Using MLC++: A Machine Learning Library in C++. Tools with Artificial Intelligence, IEEE CS Press (1996) 234-245
7. Kohavi, R.: A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In: Proceedings of IJCAI'95 (1995)
8. Merz, C.J., Murphy, P.M.: UCI Repository of Machine Learning Databases [http://www.ics.uci.edu/ ~mlearn/ MLRepository.html]. Department of Information and Computer Science, University of California, Irvine, CA (1998)

9. Puuronen, S., Terziyan, V., Katasonov, A., Tsymbal, A.: Dynamic Integration of Multiple Data Mining Techniques in a Knowledge Discovery Management System. In: Dasarathy, B.V. (Ed.): Data Mining and Knowledge Discovery: Theory, Tools, and Techniques. Proceedings of SPIE, Vol. 3695. SPIE-The International Society for Optical Engineering, USA (1999) 128-139
10. Puuronen, S., Terziyan, V., Tsymbal, A.: A Dynamic Integration Algorithm with an Ensemble of Classifiers. In: Proceedings ISMIS'99 - The Eleventh International Symposium on Methodologies for Intelligent Systems, Warsaw, Poland, June (1999) (to appear)
11. Quinlan, J.R.: C4.5 Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA (1993)
12. Schapire, R.E.: Using Output Codes to Boost Multiclass Learning Problems. In: Machine Learning: Proceedings of the Fourteenth International Conference (1997) 313-321
13. Skalak, D.B.: Combining Nearest Neighbor Classifiers. Ph.D. Thesis, Dept. of Computer Science, University of Massachusetts, Amherst, MA (1997)
14. Terziyan, V., Tsymbal, A., Puuronen, S.: The Decision Support System for Telemedicine Based on Multiple Expertise. Int. J. of Medical Informatics, Vol. 49, No. 2 (1998) 217-229
15. Terziyan, V., Tsymbal, A., Tkachuk, A., Puuronen, S.: Intelligent Medical Diagnostics System Based on Integration of Statistical Methods. In: Informatica Medica Slovenica, Journal of Slovenian Society of Medical Informatics,Vol.3, Ns. 1,2,3 (1996) 109-114
16. Thrun, S.B., Bala, J, Bloedorn, E., et al.: The MONK's Problems – A Performance Comparison of Different Learning Algorithms. Technical Report CS-CMU-91-197, Carnegie Mellon University, Pittsburg, PA (1991)
17. Tsymbal, A., Puuronen, S., Terziyan, V.: Advanced Dynamic Selection of Diagnostic Methods. In: Proceedings 11th IEEE Symp. on Computer-Based Medical Systems CBMS'98, IEEE CS Press, Lubbock, Texas, June (1998) 50-54
18. Wolpert, D.: Stacked Generalization. Neural Networks, Vol. 5 (1992) 241-259