A Dynamic Integration Algorithm with Ensemble of Classifiers

Seppo Puuronen¹, Vagan Terziyan², Alexey Tsymbal²

¹ University of Jyväskylä, P.O.Box 35, FIN-40351 Jyväskylä, Finland
sepi@jytco.jyu.fi

² Kharkov State Technical University of Radioelectronics, 14 Lenin Avenue,
310166 Kharkov, Ukraine
vagan@kture.cit-ua.net; vagan@jytco.jyu.fi

Abstract. One of the most important directions in improvement of the data-mining and knowledge discovery methods is the integration of the multiple classification techniques based on ensembles of classifiers. An integration technique should solve the problem of estimation and selection of the most appropriate component classifiers for an ensemble. We discuss an advanced dynamic integration of multiple classifiers as one possible variation of the stacked generalization method using the assumption that each component classifier is best inside certain areas of the application domain. In the learning phase a performance matrix of each component classifier is derived and then used in the application phase to predict performances of each component classifier with new instances.

1 Introduction

Data mining is the process of finding previously unknown and potentially interesting patterns and relations in large databases [6]. 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 abilities to analyze, summarize, and extract knowledge from this data. Numerous data mining methods have recently been developed to extract knowledge from large databases. In many cases it is necessary to solve the problem of evaluation and selection of the most appropriate data-mining method or a group of the most appropriate methods. Often the method selection is done statically without analyzing each particular instance. If the method selection is done dynamically taking into account characteristics of each instance, then data mining usually gives better results.

During the past several years, in a variety of application domains, researchers try to combine efforts to learn how to create and combine an ensemble of classifiers. For example, in [5] integrating multiple classifiers has been shown to be one of the four most important directions in machine learning research. The main discovery is that

ensembles are often much more accurate than the individual classifiers. In [17] the two advantages that can be reached through combining classifiers are shown: (1) the possibility that by combining a set of simple classifiers, we may be able to perform classification better than with any sophisticated classifier alone, and (2) the accuracy of a sophisticated classifier may be increased by combining its predictions with those made by an unsophisticated classifier.

We use an assumption that each data mining method has its competence area inside the application domain. The problem is then to try to estimate these competence areas in a way that helps the dynamic integration of methods. From this point of view data mining with a set of available methods has much in common with the multiple expertise problem. Both problems solve the task of receiving knowledge from several sources and have similar methods of their decision. In [18] we have suggested a voting-type technique and recursive statistical analysis to handle knowledge obtained from multiple medical experts. In [19] we presented a meta-statistical tool to manage different statistical techniques used in knowledge acquisition.

In this paper we apply this kind of thinking to the problem of dynamic integration of classifiers. Classification is a typical data mining task consisting in explaining and predicting the value of some attribute of the data given a collection of tuples with known attribute values [2]. We discuss the problem of integrating multiple classifiers and consider basic approaches suggested by various researchers recently. Then in the third chapter, we present the dynamic classifier integration approach that uses classification error estimates. We conclude with short summary and further research topics.

2 Integrating Multiple Classifiers: Related Work

Integrating multiple classifiers to improve classification has been an area of much research in machine learning and neural networks. Different approaches to integrate the multiple classifiers and their applications have been considered in [2, 5, 7, 9-11, 14, 16-21]. Integrating a set of classifiers has been shown to yield accuracy higher than the most accurate component classifier for different real-world tasks. The challenge of the problem considered is to decide which classifiers to rely on for prediction or how to combine results of component classifiers.

The problem of integrating multiple classifiers can be defined as follows. Let us suppose that a training set $\mathbf{T}$ and a set of classifiers $\mathbf{C}$ are given. The training set $\mathbf{T}$ is:

$$\{(\mathbf{x}_i, y_i), i = 1, \dots, n\},$$

where n is the number of training instances, $\mathbf{x}_i$ is the i -th training instance presented as a vector of attributes $\{x_j\}, j = 1, \dots, l$ (values of attributes are numeric, nominal,

or symbolic), and $y_i \in \{c_1, \dots, c_k\}$ is the actual class of the i -th instance, where k is the total number of classes.

The set of classifiers $\mathbf{C}$ is: $\{C_1, \dots, C_m\}$, where m is the number of available classifiers (we call them *component classifiers*). Each component classifier is either derived using some learning algorithm or hand-crafted (non-learning) classifier constructed using some heuristic knowledge.

A new instance $\mathbf{x}^*$ is an assignment of values to the vector of attributes $\{x_j\}$. The task of integrating classifiers is to classify the new instance in the best possible way using the given set $\mathbf{C}$ of classifiers.

Recently two basic approaches are used to integrate multiple classifiers. One considers *combining* the results of component classifiers, while the other considers *selection* of the best classifier for data mining. Several effective methods have been proposed which *combine* results of component classifiers. One of the most popular and simplest methods of combining classifiers is simple voting (also called majority voting and Select All Majority, SAM) [10]. In voting, the prediction of each component classifier is an equally weighted vote for that particular prediction. The prediction with most votes is selected. Ties are broken arbitrarily. There are also more sophisticated methods which combine classifiers. These include the stacking (stacked generalization) architecture [21], SCANN method based on the correspondence analysis and the nearest neighbor procedure [11], combining minimal nearest neighbor classifiers within the stacked generalization framework [17], different versions of resampling (boosting, bagging, and cross-validated resampling) that use one learning algorithm to train different classifiers on subsamples of the training set and then simple voting to combine those classifiers [5,16]. Two effective classifiers' combining strategies based on stacked generalization and called an arbiter and combiner were analyzed in [2]. Hierarchical classifier combination was also considered. Experimental results in [2] showed that the hierarchical combination approach was able to sustain the same level of accuracy as a global classifier trained on the entire data set distributed among a number of sites. Many combining approaches are based on the stacked generalization architecture considered by Wolpert [21]. Stacked generalization is the most widely used and studied model for classifier combination. A basic scheme of this architecture is presented in Fig. 1.

Fig. 1. Basic form of the stacked generalization architecture

The most basic form of the stacked generalization layered architecture consists of a set of component classifiers $\mathbf{C} = \{C_1, \dots, C_m\}$ that form the first layer, and of a single combining classifier M that forms the second layer. When a new instance $\mathbf{x}^*$ is classified, then first, each of the component classifiers C_i is launched to obtain its prediction of the instance's class $C_i(\mathbf{x}^*)$. Then the combining classifier uses the component predictions to yield a prediction for the entire composite classifier $M(C_1(\mathbf{x}^*), \dots, C_m(\mathbf{x}^*))$. The composite classifier is created in two phases: first, the component classifiers C_i are created and second, the combining classifier is trained using predictions of component classifiers $\{C_i(\mathbf{x}_j)\}$ and the training set $\mathbf{T}$ [17,21].

Even such widely used architecture for classifier combination, as stacked generalization, has still many open questions. For example, there are currently no strict rules saying which component classifiers should be used, what features to use to form the meta-level training space for combining classifier, and what combining classifier one should use to obtain an accurate composite classifier. Different combining algorithms have been considered by various researchers. For example, the classic boosting uses simple voting [16], in [17] Skalak discusses ID3 for combining nearest neighbor classifiers, and in [11] the nearest neighbor classification is proposed to make search in the space of correspondence analysis results (not directly on the predictions).

Several effective approaches have recently also been proposed for *selection* of the best classifier. One of the most popular and simplest approaches for classifier selection is CVM (Cross-Validation Majority) [10]. In CVM, the cross-validation accuracy for each classifier is estimated with the training data, and a classifier with the highest accuracy is selected. In the case of ties, the voting is used to select among classifiers with the best accuracy. Some more sophisticated selection approaches include, for example, estimating the local accuracy of component classifiers by considering errors made in instances with similar predictions of component

classifiers (instances of the same response pattern) [10] -level

not for new instances (each “referee” is a C4.5 tree that recognizes two classes) [18–20]. Application of classifier selection to medical diagnostics. It uses predicting the local accuracy of component classifiers by -by instances. This paper considers an improvement of

The set of known classifier selection methods can be partitioned into two subsets *static* and *dynamic* selection. The static approaches propose one “best” method for the whole data space, while selection by dynamic approaches depends on each new is an example of static approach, while the other selection methods above and the one proposed in this paper are dynamic.

3 Dynamic Integration of Classifiers Using Meta-

In this chapter we consider a new variation of stacked generalization which uses a

a meta-classifier -level classifier that will predict errors of component classifiers for each new input instance. Then these errors can be used to make final classification. The goal is to use each

achieve overall results that can be significantly better than those of the best individual classifier alone. Now let us consider how this goal can be justified.

Common solutions of the classification problem in data mining are based on the assumption that low-entropy areas, in other words, that points of one class are not

usually called training set, a learning algorithm creates certain model that describes this space. Fig. 2 represents a simple two-rule:

$$\begin{cases} \text{yes, if } x_2 > x_1, \\ \text{no, otherwise} \end{cases}$$

and a classifier built by the C4.5 algorithm [15] that splits the space into some rectangles, where each rectangle corresponds to a leaf of the decision tree.

We can see that the model does not describe the space exactly. In fact, the accuracy of a model depends on many factors: the size and shape of decision boundaries, the number of low-entropy areas (problem-dependent factors), the completeness and cleanliness of the training set (sample-dependent factors) and certainly, the individual peculiarities of the learning algorithm (algorithm-dependent factors). We can see that points misclassified by the model (gray areas in Fig. 2) are not uniformly scattered and are concentrated in some zones also.

Fig. 2. An example of space model built by C4.5

Thus we can consider the entire space as a set of null-entropy areas of points with categories “a classifier is correct” and “a classifier is incorrect”. The dynamic approach to integration of multiple classifiers attempts to create a meta-model that describes the space according to this vision. This meta-model is then used for predicting errors of component classifiers in new instances.

Our approach contains two phases. In the learning phase a performance matrix that describes the space of classifiers’ performances is formed and stacked. It contains information about the performance of each component classifier in each training instance. In the second (application) phase, the combining classifier is used to predict the performance of each component classifier for a new instance. We consider two variations of the application phase that differently use of the predicted classifiers’ performances. We call them Dynamic Selection (DS) and Dynamic Voting (DV). In the first variation (DS), just the classifier with the best predicted performance is used to make the final classification. In the other variation (DV), each component classifier receives a weight that depends on the local classifier performance, then each classifier votes with its weight and the prediction with the highest vote is selected.

Ortega, Koppel, and Argamon-Engelson in [14] propose to build a referee predictor for each component classifier that will be able to predict whether corresponding classifier can be trusted for each instance. In their approach the final classification is that returned by the component classifier whose correctness can be trusted the most according to the predictions of referees. For the referee induction they use the decision tree induction algorithm C4.5. This approach was used for selection among a set of binary classifiers which classify only two cases. To train each referee they divided the training set into two parts: those instances that were 1) correctly and 2) incorrectly classified by the corresponding component classifier. Our approach is closely related to that considered by Ortega, Koppel, and Argamon-Engelson [14]. The same basic idea is used: each classifier has a particular subdomain for which it is most reliable. A meta-classification architecture also consists of two levels. The first level contains component classifiers, while the second level contains a combining classifier, in our case an algorithm that predicts the error for each component classifier.

The main difference between these approaches is in the combining algorithm. Instead of the C4.5 decision tree induction algorithm used in [14] to predict errors of component classifiers, we use the weighted nearest neighbor classification (WNN) [1,3]. WNN simplifies the learning phase of the composite classifier. The composite classifier that uses WNN does not need learning m referees (meta-level classifiers) as in [14]. One needs only to calculate the performance matrix for component classifiers during the learning phase. In the application phase, nearest neighbors of a new instance among the training instances are found out and corresponding component classifier performances are used to calculate the predicted classifier performance for each component classifier (a degree of belief to the classifier). In this calculation we sum up corresponding performance values of a classifier using weights that depend on the distances between a new instance and its nearest neighbors.

The use of WNN as a meta-level classifier is agreed with the assumption that each component classifier has certain subdomains in the space of instance attributes, where it is more reliable than the others. This assumption is supported by the experiences, that classifiers usually work well not only in certain points of the domain space, but in certain subareas of the domain space (as in the example shown in Fig. 2). Thus if a classifier does not work well with the instances near a new instance, then it is quite probable that it will not work well with the new instance. Of course, the training set should provide a good representation of the whole domain space in order to make such statement highly confident.

Our approach can be considered as a particular case of the stacked generalization architecture (Fig. 1). Instead of the component classifier predictions, we use information about the performance of component classifiers for each training instance. For each training instance we calculate a vector of classifier errors that contains m values. These values can be binary (i.e. classifier gives correct or incorrect result) or can represent corresponding misclassification costs. This information about the performance of component classifiers is then stacked (as in stacked generalization) and is further used together with the initial training set as meta-level knowledge for estimating errors in new instances.

In [18, 19], the jackknife method (also called leave-one-out cross-validation and the sliding exam) [7] was used for calculation of component classifiers' errors on the training set. In the jackknife method, for each instance in the training set each of the m component classifiers is trained using all the other instances of the training set. After training, the held-out instance is classified by each of the trained component classifiers. Then, comparing the values of predictions for each of the component classifiers with the actual class of that instance, a vector of classification errors (or misclassification costs) is formed. Those n vectors (matrix of classifier performance $\mathbf{P}_{n \times m}$) together with the set of attributes of training instances $\mathbf{T}_{n \times l}$, where l is the number of attributes, are used as a training set for the meta-level classifier $\mathbf{T}_{n \times (l+m)}^*$.

It is necessary to note that the jackknife and cross-validation usually give pessimistically biased estimates. It is a result of the fact that under the estimation, classifiers are trained only on a part of the training set, and classifiers trained on the whole training set will be used in real problems. Although we obtain biased

estimations, these estimations are similarly biased for all component classifiers and have small variance. That is why the jackknife and cross-validation can be used for generating the meta-level error information about component classifiers in our case.

Pseudo-code for the dynamic classifier integration algorithm is given in Fig. 3 that uses one cross-validation run for the case when all component classifiers are learning.

```

T component classifier training set
Ti i-th fold of the training set
T* meta-level training set for the combining algorithm
x attributes of an instance
c(x) class of x
C set of component classifiers
Cj j-th component classifier
Cj(x) prediction of Cj on instance x
Ej(x) estimation of error of Cj on instance x
E*j(x) prediction of error of Cj on instance x
m number of component classifiers
W vector of weights for component classifiers
nn number of near-by instances for error prediction
WNNi weight of i-th near-by instance
procedure learning_phase(T,C)
    begin {fills the meta-level training set T*}
        partition T into v folds
        loop for Ti ⊂ T, i = 1, ..., v
            loop for j from 1 to m
                train(Cj, T-Ti)
            loop for x ∈ Ti
                loop for j from 1 to m
                    compare Cj(x) with c(x) and derive Ej(x)
                collect (x, E1(x), ..., Em(x)) into T*
            loop for j from 1 to m
                train(Cj, T)
    end
function DS_application_phase(T*,C,x) returns class of x
    begin
        loop for j from 1 to m
            
$$E_j^* \leftarrow \frac{1}{nn} \sum_{i=1}^{nn} W_{NN_i} \cdot E_j(\mathbf{x}_{NN_i}) \quad \{\text{WNN estimation}\}$$


```



```

     $l \leftarrow \arg \min_j E_j^* \{ \text{number of cl-er with min. } E_j^* \}$ 
    {the least j in the case of ties}
    return  $C_j(\mathbf{x})$ 
end
function DV_application_phase( $\mathbf{T}^*, \mathbf{C}, \mathbf{x}$ ) returns class of  $\mathbf{x}$ 
begin
    loop for j from 1 to m
         $W_j \leftarrow 1 - \frac{1}{nn} \sum_{i=1}^{nn} W_{nn_i} \cdot E_j(\mathbf{x}_{nn_i})$  {WNN estimation}
    return Weighted_Voting( $\mathbf{W}, C_1(\mathbf{x}), \dots, C_m(\mathbf{x})$ )
end

```

Fig. 3. Pseudo-code for the proposed classifier integration technique

The Fig.3 includes both: the learning phase (procedure *learning_phase*) and the two variations of the application phase (functions *DS_application_phase* and *DV_application_phase*). In the learning phase first, the training set $\mathbf{T}$ is partitioned into folds. Then the cross-validation technique is used to estimate errors of component classifiers E_j on the training set and to form the meta-level training set $\mathbf{T}^*$ that contains features of training instances $\mathbf{x}_i$ and estimations of errors of component classifiers on those instances E_j . The learning phase finishes with training the component classifiers C_j on the whole training set. In the DS application phase the classification error E_j^* is predicted for each component classifier C_j using the WNN procedure and a classifier with the least error (with the least number in the case of ties) is selected to make the final classification. In the DV application phase each component classifier C_j receives a weight W_j and the final classification is conducted by voting classifier predictions $C_j(\mathbf{x})$ with their weights W_j . Thus one can see that DS is a particular case of DV, where the selected classifier receives the weight 1 and the others - 0.

In [10] it was proposed to conduct several cross-validation runs in order to obtain more accurate estimates of the component classifiers' errors. Then each estimated error will be equal to the number of times that an instance was incorrectly predicted by the classifier when it appeared as a test example in a cross-validation run. Several cross-validation runs can be useful when one has enough computational resources.

The described algorithms can be easily extended to take into account other than the classification error performance criteria, e.g. time of classification and learning. Then some multi-criteria's metrics should be used, for example, as it was proposed in [13].

4 Conclusion

In data mining one of the key problems is selection of the most appropriate data-mining method. Often the method selection is done statically without paying any attention to the attributes of an instance. Recently the problems of integrating multiple classifiers have also been researched from the dynamic integration point of view.

Our goal in this paper was to further enhance dynamic integration of multiple classifiers. We discussed two basic approaches currently used and represented a meta-classification framework. This meta-classification framework consists of two levels: the component classifier level and the combining classifier level (meta-level). Considering the assumption that each component classifier is best inside certain subdomains of the application domain we proposed a variation of the stacked generalization method where classifiers of different nature can be integrated.

The performance matrix for component classifiers is derived during the learning phase. It is used in the application phase as a base when each component classifier is evaluated. During the evaluation we use features of the current instance to find the most similar instances of the training set. The recorded performance information of component classifiers for these neighbors is used for selection in our algorithms.

Further research is needed to compare the algorithm with others on large databases to find out cases where it gives gains of accuracy. We expect the algorithm can be easily extended taking into account other performance criteria. In further research, the combination of the algorithm with other integration methods (boosting, bagging, minimal nearest neighbor classifiers, etc.), can be investigated.

Acknowledgment: This research is partly supported by the Academy of Finland and the Center for International Mobility (CIMO). We are grateful to Mr. Artyom Katasonov from Kharkov Technical University of Radioelectronics for considerable help in experimental checking of the algorithm.

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. Cost, S., Salzberg, S.: A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features. Machine Learning, Vol. 10, No. 1 (1993) 57-78.
4. Dietterich, T.G.: Approximate Statistical Tests for Comparing Supervised Classification Learning Algorithms. Neural Computation (1998) to appear.
5. Dietterich, T.G.: Machine Learning Research: Four Current Directions. AI Magazine, Vol. 18, No. 4 (1997) 97-136.
6. Fayyad, U., Piatetsky-Shapiro, G., Smyth, P., Uthurusamy, R.: Advances in Knowledge Discovery and Data Mining. AAAI/ MIT Press (1997).
7. Kohavi, R.: A Study of Cross-Validation and Bootstrap for Accuracy Estimation and Model Selection. In: Proceedings of IJCAI'95 (1995).

8. 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.
9. Koppel, M., Engelson, S.P.: Integrating Multiple Classifiers by Finding their Areas of Expertise. In: AAAI-96 Workshop On Integrating Multiple Learning Models (1996) 53-58.
10. Merz, C.: Dynamical Selection of Learning Algorithms. In: D. Fisher, H.-J. Lenz (Eds.), Learning from Data, Artificial Intelligence and Statistics, Springer Verlag, NY (1996).
11. Merz, C.J.: Combining Classifiers Using Correspondence Analysis. In: Advances in Neural Information Processing Systems (NIPS 10) (1998) to appear.
12. 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).
13. Nakhaeizadeh, G., Schnabl, A.: Development of Multi-Criteria Metrics for Evaluation of Data Mining Algorithms. In: Proceedings of the 3rd International Conference On Knowledge Discovery and Data Mining KDD'97 (1997) 37-42.
14. Ortega, J., Koppel, M., Argamon-Engelson, S.: Arbitrating Among Competing Classifiers Using Learned Referees, Machine Learning (1998) to appear.
15. Quinlan, J.R.: C4.5 Programs for Machine Learning. Morgan Kaufmann, San Mateo, CA (1993).
16. Schapire, R.E.: Using Output Codes to Boost Multiclass Learning Problems. In: Machine Learning: Proceedings of the Fourteenth International Conference (1997) 313-321.
17. Skalak, D.B.: Combining Nearest Neighbor Classifiers. Ph.D. Thesis, Dept. of Computer Science, University of Massachusetts, Amherst, MA (1997).
18. Terziyan, V., Tsymbal, A., Puuronen, S.: The Decision Support System for Telemedicine Based on Multiple Expertise. International Journal of Medical Informatics, Vol. 49, No. 2 (1998) 217-229.
19. 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, No. 1,2,3 (1996) 109-114.
20. 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.
21. Wolpert, D.: Stacked Generalization. Neural Networks, Vol. 5 (1992) 241-259.