2009 Ninth International Conference on Intelligent Systems Design and Applications

Combining Uncertainty Sampling Methods for Active Meta-Learning

Ricardo B. C. Prudêncio and Teresa B. Ludermir

Abstract— **Meta-Learning has been applied to acquire useful knowledge to predict learning performance. Each training example in Meta-Learning (i.e. each meta-example) is related to a learning problem and stores features of the problem plus the performance obtained by a set of candidate algorithms when evaluated on the problem. Based on a set of such meta-examples, a meta-learner will be used to predict algorithm performance for new problems. The generation of a set of meta-examples can be expensive, since for each problem it is necessary to perform an empirical evaluation of the candidate algorithms. In a previous work, we proposed the Active Meta-Learning, in which Active Learning was used to reduce the set of meta-examples by selecting only the most relevant problems for meta-example generation. In the current work, we proposed the combination of different Uncertainty Sampling methods for Active Meta-Learning, considering that each individual method will provide useful information that can be combined in order to have a better assessment of problem relevance for meta-example generation. In our experiments, we observed a gain in Meta-Learning performance when the proposed method was compared to the individual active methods being combined.**

I. Introduction

Meta-Learning is a framework developed in supervised Machine Learning with the aim of relating features of the learning problems to the performance of the learning algorithms [1]. The knowledge in Meta-Learning is acquired from a set of *meta-examples*, in which each meta-example stores the experience obtained from the application of a set of candidate algorithms in a particular problem. The Meta-Learning process can automatically capture the expertise gained on different problems, which will be used to predict learning performance and support algorithm selection.

In the Meta-Learning framework, each meta-example stores: (1) the features used to describe a problem; and (2) information about the performance obtained by the algorithms in the problem. By receiving a set of such meta-examples, another learning algorithm (the *meta-learner*) is applied to acquire knowledge relating the performance of the candidate algorithms and the descriptive features of the problems.

Generating a good set of training examples for Meta-Learning may be a costly process, depending on the context. In fact, in order to produce a single meta-example, it is necessary to perform an empirical evaluation (e.g. cross-validation) of the candidate algorithms on a problem. Hence, the cost of generating a whole set of meta-examples may be high, depending, for instance, on the number and complexity of the candidate algorithms, the methodology of empirical evaluation and the amount of available problems.

Ricardo B. C. Prudêncio, Teresa B. Ludermir - Center of Informatics, Federal University of Pernambuco, 50732-970, Recife (PE), Brazil; email: {rbcp, tbl}@cin.ufpe.br.

In a previous work [2], we proposed the Active Meta-Learning in which Active Learning techniques [3] were used to support the generation of meta-examples. Active Learning is a paradigm of Machine Learning which aims to reduce the number of training examples, at same time maintaining (or even improving) the performance of the learning algorithm. Active Learning is ideal for learning domains in which the acquisition of labeled examples is an expensive process, which is the case of Meta-Learning.

In [2], we presented the first experiments performed to evaluate the viability of Active Meta-Learning. In that work, different active methods based on *Uncertainty Sampling* were used to select meta-examples for an instance-based meta-learner. Uncertainty Sampling is a well established Active Learning approach, and it has been widely used in a variety of domains [4]. The experiments performed in [2] showed a significant gain in Meta-Learning performance when the Uncertainty Sampling methods were used. However, in these experiments, the performance of the evaluated active methods varied a lot during the selection of meta-examples. Each method presented a distinct behavior depending on the number of meta-examples generated at each moment. Based on these results, we envisioned whether the combination of the active methods would eventually produce better results.

Considering the above motivation, in the current work we extend our previous research by combining different Uncertainty Sampling methods for Active Meta-Learning. In our proposal, each method being combined is initially used to generate a *ranking* for the problems avaliable to generate meta-examples. The ranks assigned by the different methods are then averaged in order to provide a final score of relevance for each problem. Finally, the problems with better average ranks are selected to generate new meta-examples.

The combining procedure was evaluated in a case study which consisted of predicting the performance of Multi-Layer Perceptron (MLP) networks [5] for regression problems. In the performed experiments, the proposed solution was used to combine the same active methods evaluated in [2]. The obtained results revealed a gain in the Meta-Learning performance when the combining procedure was used to generate meta-examples.

Section II brings a brief presentation of Meta-Learning, followed by section III which presents the Active Meta-Learning proposal. Section IV describes the developed work, followed by section V which presents the experiments and results. Finally, section VI concludes the paper.

II. Meta-Learning

Meta-Learning predicts the performance of learning algorithms based on features of the learning problems [1]. It acquires knowledge from a set of meta-examples, which store the experience obtained from applying the algorithms to different problems in the past. According to [6], Meta-Learning can be defined by considering four aspects:

- the problem space, P, representing the set of instances of a given problem class (usually classification and regression problems);
- the meta-feature space, F, that contains characteristics used to describe the problems (e.g., number of training examples, correlations between attributes, ...);
- the algorithm space, A, that is a set of one or more candidate algorithms to solve the problems in P;
- a performance information, Y, that characterizes the performance of an algorithm on a problem (e.g., classification accuracy estimated by cross-validation).

In this framework, Meta-Learning receives as input a set of meta-examples, in which each meta-example is derived from the empirical evaluation of the algorithms in A on a given problem in P. More specifically, each meta-example stores: (1) the values of the meta-features F extracted from a problem; and (2) the performance information Y estimated for the problem. Hence, the *meta-learner* is another learning technique that relates a set of predictor attributes to the performance information.

Different Meta-Learning approaches have been proposed in the literature. In a strict approach (e.g., [7]), each meta-example stores as performance information a class attribute associated to the candidate algorithm which obtained the highest accuracy in the learning problem. In this formulation, the meta-learner just becomes a classifier in which the meta-features correspond to predictor attributes for the class label associated to the best candidate algorithm.

Other approaches have been proposed in order to add new functionalities in the Meta-Learning process. The Meta-Regression approach [8], for instance, tries to directly predict the accuracy (or alternatively the error) of the learning algorithms instead of simply predicting the class that corresponds to the best algorithm. In [9], [10], the authors proposed different Meta-Learning approaches to generate rankings of algorithms. More detailed reviews of these techniques can be found in recent textbooks [1], [11].

III. Active Meta-Learning

As seen in section 2, in order to generate a meta-example from a given problem, it is necessary to perform an empirical evaluation of the candidate algorithms on the problem. The empirical evaluation is performed in order to collect the performance information of the algorithms, and hence, to define the target attribute in the Meta-Learning process (e.g. the class corresponding to the best algorithm).

Although the proposal of Meta-Learning is to perform the empirical evaluation of the algorithms only in a limited number of problems, the cost of generating a set of meta-examples may be high depending on a number of aspects, including the methodology of empirical evaluation and the number and complexity of the candidate algorithms. In this context, the use of Active Learning may improve the Meta-Learning process by reducing the number of required meta-examples, and consequently the number of empirical evaluations on the algorithms.

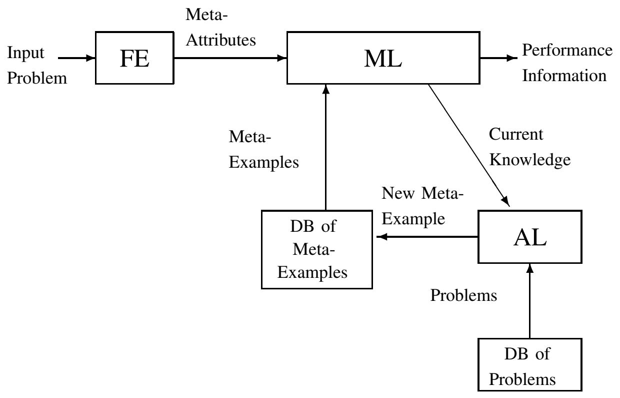

Fig. 1. Active Meta-Learning Architecture.

Fig. 1 presents the architecture of system following our proposal, which has three phases. In the meta-example generation, the Active Learning (AL) module selects from a base of problems, those ones considered the most relevant for the Meta-Learning task. The selection of problems is performed based on a pre-defined criteria implemented in the module, which takes into account the features of the problems and the current knowledge of the Meta-Learner (ML). The candidate algorithms are then empirically evaluated on the selected problems, in order to collect the performance information related to the algorithms. Each generated meta-example (composed by meta-features and performance information) is then stored in an appropriate database.

In the training phase, the Meta-Learner acquires knowledge from the database of meta-examples generated by the AL module. This knowledge associates meta-features to the performance of the candidate algorithms. The acquired knowledge may be refined as more meta-examples are provided by the AL module. In the use phase, given a new input problem, the Feature Extractor (FE) module extracts the values of the meta-features. According to these values, the ML module predicts the performance information of the algorithms. For that, it uses the knowledge previously acquired as a result of the training phase.

IV. Developed Work

In [2], we presented the initial experiments performed to evaluate the viability of the proposed solution. In this work, two different Uncertainty Sampling methods were evaluated in an implemented prototype. Experiments performed in case studies revealed that no active method was steadily better than the other. In the current work, we proposed to combine different active methods, aiming to have a better assessment of the relevance of each learning problem available in the generation of meta-examples. Combining different techniques is a very common approach in Machine Learning, and it is motivated by both theoretical and empirical results [12]. In this work, we deployed a combining procedure in a problem which was not investigated yet.

In order to evaluate the viability of our proposal, we implemented a prototype which was applied in a case study. In this prototype, the k-Nearest Neighbors (k-NN) algorithm was used to predict the performance of MLPs for regression problems. Two different Uncertainty Sampling methods were used in isolation and also being combined by the proposed solution. In the next sections, we provide more details of the implemented prototype.

A. Meta-Learner

The Meta-Learner in the prototype corresponds to a conventional classifier, applicable to tasks in which the performance information is formulated as a class attribute (e.g. the class associated to the best algorithm or the class related to patterns of performance). In the implemented prototype, we used the k-NN algorithm which has some advantages when applied to Meta-Learning [10]. For instance, when a new meta-example becomes available, it can be easily integrated without the need to initiate re-learning [10]. In this section, we provide a description of the meta-learner based on k-NN.

Let $E = \{e_1, \ldots, e_n\}$ be the set of n problems used to generate a set of n meta-examples $ME = \{me_1, \ldots, me_n\}$. Each meta-example is related to a problem and stores the values of p features $X_1, \ldots, X_p$ (implemented in the FE module) for the problem and the value of a class attribute C, which is the performance information

Let $\mathcal{C} = \{c_1, \ldots, c_L\}$ be the domain of the class attribute C, which has L possible class labels. In this way, each meta-example $me_i \in ME$ is represented as the pair $(\mathbf{x}_i, C(e_i))$ storing: (1) the description $\mathbf{x}_i$ of the problem e_i, where $\mathbf{x}_i = (x_i^1, \ldots, x_i^p)$ and $x_i^j = X_j(e_i)$; and (2) the class label associated to e_i, i.e. $C(e_i) = c_l$, where $c_l \in \mathcal{C}$.

Given a new input problem described by the vector $\mathbf{x} = (x^1, \ldots, x^p)$, the k-NN meta-learner retrieves the k most similar meta-examples from ME, according to the distance between meta-attributes. The distance function (*dist*) implemented in the prototype was the unweighted L_1-Norm, defined as:

$$dist(\mathbf{x}, \mathbf{x}_i) = \sum_{j=1}^{p} \frac{|x^j - x_i^j|}{max_i(x_i^j) - min_i(x_i^j)} \qquad (1)$$

The prediction of the class label for the new problem is performed according to the number of occurrences (votes) of each $c_l \in \mathcal{C}$ in the class labels associated to the retrieved meta-examples.

B. Active Learning

As seen, the Meta-Learner acquires knowledge from a set of labeled meta-examples associated to a set of learning problems. The Active Learning module, in turn, receives a set of unlabeled meta-examples, associated to the problems in which the candidate algorithms were not yet evaluated and, hence, the class labels are not known. Therefore, the main objective of this module is to incrementally select unlabeled meta-examples to be labeled.

In our prototype, we evaluated three different methods for uncertainty sampling. The first two methods described in this section deployed different criteria for assigning degrees of classification uncertainty to the unlabeled meta-examples. The third method deploys a combining procedure, aiming to explore the eventual advantages of the first two methods.

1) Uncertainty Method A: The uncertainty of k-NN was defined in [13] as the ratio of: (1) the distance between the unlabeled example and its nearest labeled neighbor; and (2) the sum of the distances between the unlabeled example and its nearest labeled neighbors of different classes. A high value of uncertainty indicates that the unlabeled example has nearest neighbors with similar distances but conflicting labeling. Hence, once the unlabeled example is labeled, it is expected that the uncertainty in its neighborhood should be reduced.

In our context, let E be the set of problems associated to the labeled meta-examples, and let $\widetilde{E}$ be the set of problems used to generate unlabeled meta-examples. Let E_l be the subset of labeled problems associated to the class label c_l, i.e. $E_l = \{e_i \in E | C(e_i) = c_l\}$. Given E, the classification uncertainty of k-NN for each $\widetilde{e} \in \widetilde{E}$ is defined as:

$$\mathcal{S}(\widetilde{e}|E) = \frac{\min_{e_i \in E} dist(\widetilde{\mathbf{x}}, \mathbf{x}_i)}{\sum_{l=1}^{L} \min_{e_i \in E_l} dist(\widetilde{\mathbf{x}}, \mathbf{x}_i)} \qquad (2)$$

In the above equation, $\widetilde{\mathbf{x}}$ is the description of problem $\widetilde{e}$. The AL module then selects, for generating a new labeled meta-example, the problem $\widetilde{e}^* \in \widetilde{E}$ with highest uncertainty:

$$\widetilde{e}^* = argmax_{\widetilde{e} \in \widetilde{E}} \mathcal{S}(\widetilde{e}|E) \qquad (3)$$

Finally, the selected problem is labeled (i.e. the class value $C(\widetilde{e}^*)$ is defined), through the empirical evaluation of the candidate algorithms using the avaliable data of the problem.

2) Uncertainty Method B: In the second active method, we adopted the concept of entropy to define classification uncertainty. Assume that the k-NN can predict for each given example a probability distribution over the possible class values. Formally, the probability distribution for an unlabeled problem $\widetilde{e}$ can be represented as:

$$p_C(\widetilde{e}|E) = (p(C(\widetilde{e}) = c_1|E), \ldots, p(C(\widetilde{e}) = c_L|E)) \qquad (4)$$

According to [14], the entropy of the probability distribution reflects the uncertainty of the classifier in the predicted class value. The entropy of the probability distribution is computed as:

$$Ent(\widetilde{e}|E) = -\sum_{l=1}^{L} p(C(\widetilde{e}) = c_l|E) * \log_2 p(C(\widetilde{e}) = c_l|E) \qquad (5)$$

If the probability distribution is highly spread, the value of entropy will be high, which indicates that the classifier is not certain in its prediction. On the other hand, if the distribution is highly focused on a single class label, the entropy is low, indicating a low degree of uncertainty in predicted class value.

As in the previous section, in this method, the AL module selects the problem $\widetilde{e}^* \in \widetilde{E}$ with highest uncertainty defined by the entropy measure:

$$\widetilde{e}^* = argmax_{\widetilde{e} \in \widetilde{E}} Ent(\widetilde{e}|E) \tag{6}$$

In our work, the class probability distribution for a given example is estimated by using the number of votes that each class label received among the retrieved meta-examples.

3) Combining Method: In this section, we described a method to combine the uncertainty criteria described in the previous subsections. The combining method was adapted from a previous work which used the concept of *average ranks* originally applied to active construction of user profiles in recommending systems [15].

In the combination, each uncertainty method being combined is initially used to generate a ranking of unlabeled meta-examples. Following, the rankings provided by the different methods are averaged and the unlabeled meta-example with the best average rank is selected. The combining method can be formally described as follows.

Let $r^A(\widetilde{e})$ and $r^B(\widetilde{e})$ be the ranks of the unlabeled meta-example $\widetilde{e}$, by respectively considering the uncertainty methods A and B (i.e., by using the measures $\mathcal{S}(\widetilde{e}|E)$ and $Ent(\widetilde{e}|E)$ to sort the meta-examples in a decreasing order and to assign the respective ranks). Hence, the lower is the rank of a meta-example, the higher is its uncertainty. The two ranks can be combined using the equation:

$$\overline{r}(\widetilde{e}) = w^A * r^A(\widetilde{e}) + w^B * r^B(\widetilde{e}) \tag{7}$$

where $w^A, w^B \in [0; 1]$ and $w^A + w^B = 1$. In this equation, $\overline{r}(\widetilde{e})$ is the average rank of $\widetilde{e}$, weighted by the numerical values w^A and w^B. Finally, the unlabeled meta-example $\widetilde{e}^* \in \widetilde{E}$ with lowest average rank is selected:

$$\widetilde{e}^* = argmin_{\widetilde{e} \in \widetilde{E}} \overline{r}(\widetilde{e}|E) \tag{8}$$

We highlight that the combining method described above can be easily generalized to combine more than two methods. Also, different values can be assigned to w^A and w^B in order to control the importance of each method being combined. In our work, for simplicity, we define $w^A = w^B = 0.5$, although different configurations of weights can be evaluated in the future.

V. Case Study

In the case study, the prototype was evaluated in a meta-learning task which consisted in predicting the performance of Multi-Layer Perceptron (MLP) networks for regression problems. The set of meta-examples was generated from the application of MLP to 50 different regression problems, available in the WEKA project[1]. Each meta-example was related to a regression problem and stored: (1) the values of $p = 10$ meta-attributes describing the problem; and (2) a class attribute C which categorized the performance obtained by the MLP network on the problem.

The first step to generate a meta-example from a problem is to extract its meta-features. In the case study, a total number of $p = 10$ meta-features adopted in [2] was used to describe the datasets of regression problems:

1) X_1 - Log of the number of training examples;
2) X_2 - Log of the ratio between number of training examples and attributes;
3) X_3, X_4, X_5 and X_6 - Minimum, maximum, mean and standard deviation of the absolute values of correlation between predictor attributes and the target attribute;
4) X_7, X_8, X_9 and X_{10} - Minimum, maximum, mean and standard deviation of the absolute values of correlation between pairs of predictor attributes.

The meta-feature X_1 is an indicator of the amount of data available for training, and X_2, in turn, indicates the dimensionality of the dataset. The meta-features X_3, X_4, X_5 and X_6 indicate the amount of relevant information available to predict the target attribute. The meta-features X_7, X_8, X_9 and X_{10}, in turn, indicate the amount of redundant information in the dataset.

The second step to generate a meta-example is to estimate the performance information on the problem being tackled. In our case study, this step consists of evaluating the performance of one-hidden layer MLPs trained by the standard BackPropagation (BP) algorithm [2]. In order to define the performance information of each problem, the following methodology of evaluation was applied.

The problem's dataset was divided in the training, validation and test sets, in the proportion of 50%, 25% and 25%. As usual, the training set was used to adjust the MLP's weights, the validation set was used to estimate the MLP performance during training, and the test set was used to evaluate the performance of the trained MLP. The optimal number of hidden nodes was defined by testing the values 1, 2, 4, 8, 16 and 32. For each number of nodes, the MLP was trained 10 times with random initial weights. In the training process, we adopted benchmarking rules [16]: early stopping was used to avoid overfitting with the GL_5 stopping criterion and a maximum number of 1000 training epochs (see [16] for details of these rules). The optimal number of nodes was chosen as the value in which the MLP obtained the lowest average NMSE (Normalized Mean Squared Error) on the validation set over the 10 runs. The NMSE is defined as:

$$NMSE = \frac{\sum_{i=1}^{n_v}(t_i - o_i)^2}{\sum_{i=1}^{n_v}(t_i - \overline{t})^2} \tag{9}$$

[1]These datasets are specifically the sets provided in the files *numeric* and *regression* available to download in http://www.cs.waikato.ac.nz/ml/weka/

[2]The BP algorithm was implemented by using the NNET Matlab toolbox. Learning rates were defined by default.

In the equation, n_v is the number of examples in the validation set, t_i and o_i are respectively the true and the predicted value of the target attribute for example i, and $\bar{t}$ is the average of the target attribute. The NMSE values have no scale and are comparable across different datasets, which is adequate to Meta-Learning [17]. Values of NMSE lower than 1 indicate that the MLP provided better predictions than the mean value at least.

Finally, the performance information C related to a problem was defined in our prototype as a binary attribute assigned to 1 if the observed NMSE is lower than 0.5 and assigned to 0 otherwise. Hence, the meta-examples with class label 1 were those problems in which the MLP obtained the best performance patterns.

A. Experiments and Results

The experiments performed in our work were motivated by different aims. First, we intended to evaluate whether the active methods considered in our work were useful to overcome a passive (random) procedure for selecting meta-examples. Also, we evaluated the performance of the proposed combining method compared to the individual methods being combined.

1) Experiments: The prototype was evaluated for different configurations of the k-NN meta-learner (with k = 3, 5, 7, 9 and 11 nearest neighbors). For each configuration, a leave-one-out experiment was performed to evaluate the performance of the meta-learner, also varying the number of meta-examples provided by the Active Learning module. This experiment is described just below.

At each step of leave-one-out, one problem is left out for testing the ML module, and the remaining 49 problems are considered as candidates to generate meta-examples. The AL module progressively includes one meta-example in the training set of the ML module, up to the total number of 49 training meta-examples. At each included meta-example, the ML module is judged on the test problem left out, receiving either 1 or 0 for failure or success. Hence, a curve with 49 binary judgments is produced for each test problem. Finally, the curve of error rates obtained by ML can be computed by averaging the curves of judgments over the 50 steps of the leave-one-out experiment.

The above procedure was applied for each uncertainty sampling method considered in the AL module (see section IV-B). As a basis of comparison, the same above experiment was applied to each configuration of k-NN, but using in the AL module a Random Sampling method for selecting unlabeled problems. According to [13], despite its simplicity, the random method has the advantage of performing a uniform exploration of the example space. Finally, we highlight that the experiments were performed in 30 different runs for each configuration of the k-NN meta-learner.

Fig. 2, 3 and 4 show the curve of error rate of each uncertainty method compared to the curve obtained by using Random Sampling. In the most part of the curves, the error rates achieved by using the three uncertainty methods were lower than the rates observed by using the random procedure. The Uncertainty Method A, B and the Combining Method achieved lower error rates compared to the Random Sampling in 39, 32 and 37 points in the curves respectively (about 79.6%, 65.3% and 75.5% of the 49 points). The good results of the uncertainty methods were also observed to be statistically significant. A t-test (95% of confidence) applied to the difference of error rates indicated that the Uncertainty Method A obtaining a gain in performance compared to the Random Method in 31 points in the curve of error rates (about 63.3% of the 49 points). The Uncertainty Method B in turn obtained a statistical gain in performance in 22 points in the curve of error rates (about 44.9% of the points). Finally, the Combining method was statistically better than the Random Method in 35 points (about 71.4% of the points), which was the best result among the uncertainty methods.

The Uncertainty Method A and B yielded different performance patterns considering different segments of the error curves (see Fig. 5). In fact, in the first half of the curves the performance of the entropy-based method was steadily better than the Uncertainty Method A. However, there is a turning point in the second half of the curves of error rates, in such a way that the Uncertainty Method A became better than method B. The combination of the two methods, in turn, yielded a more consistent performance along the curve. By ranking the three methods in each point of the curves (i.e., rank = 1 for the best method, rank = 2 for the second method and rank = 3 for the worst method), we observed an average rank over the 49 points equal to 2.0 and 2.1 for the Uncertainty Methods A and B, respectively. The average rank of the Combining Method was 1.8 in turn, which indicates that in average the Combining method obtained better positions compared to both the Uncertainty Methods A and B.

VI. Conclusion

In this paper, we presented the proposal of combining different Uncertainty Sampling techniques for Active Meta-Learning using the concept of average ranks. We can point out contributions of our work to both fields of Meta-Learning and Active Learning. We believe that the current research will result in new developments in the future.

Experiments performed on a case study revealed that the three active methods were significantly better than a random passive procedure for generating meta-examples. The combining method obtained the best results if we consider the statistical significance of the results in comparison to the random method. Also, the combining method obtained the best average position along the curve of error rates, compared to the individual methods being combined.

In future work, we intend to investigate the combination of a higher number of active methods. In the current work, we combined the active methods using an equal weight to each method, thus assuming that each method is equally important in the combination. In future work, we intend to propose procedures to adapt the combining weights in order to assign different contributions for each method. Finally, experiments will be performed in new case studies.

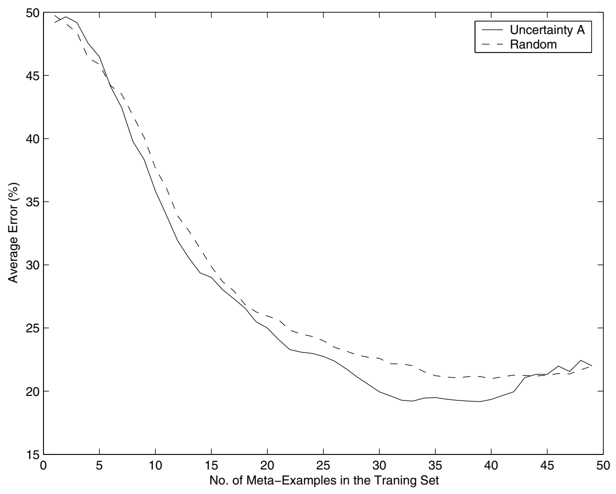

Fig. 2. Curve of error rates: Uncertainty method A vs. Random method

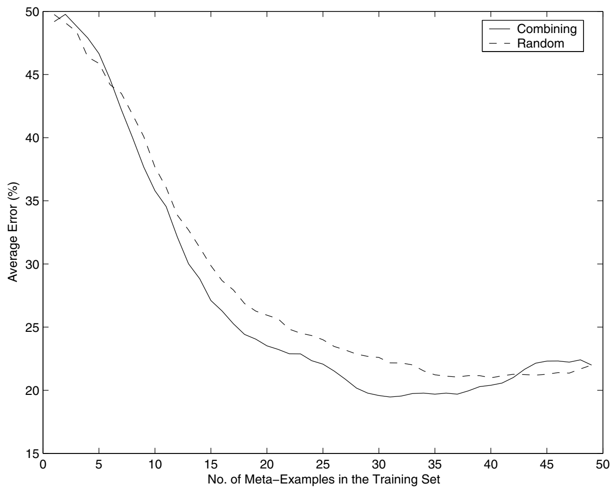

Fig. 4. Curve of error rates: Combining method vs. Random method.

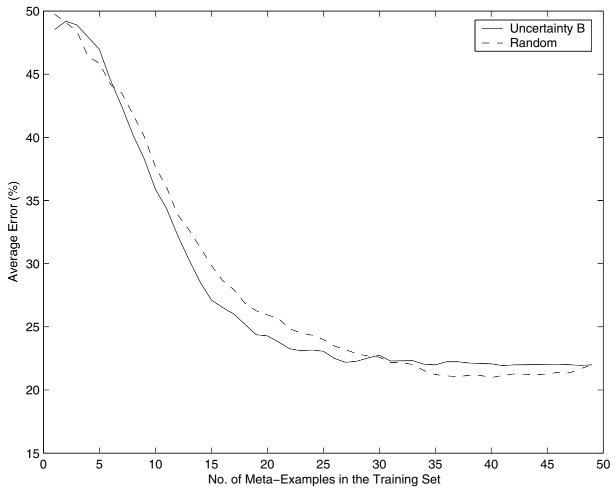

Fig. 3. Curve of error rates: Uncertainty method B vs. Random method

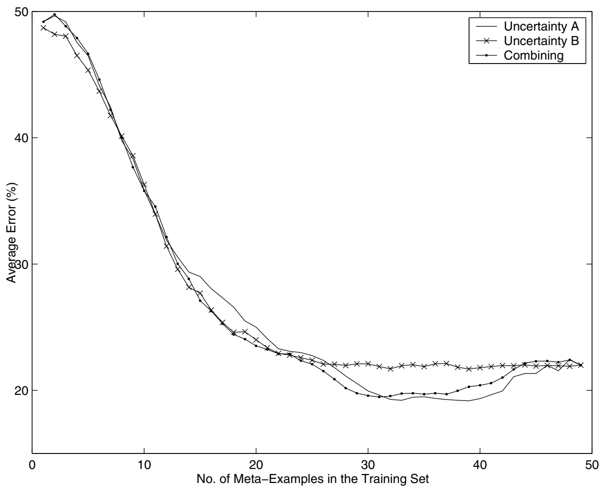

Fig. 5. Curve of error rates for the three uncertainty methods.

Acknowledgments: The authors would like to thank CNPq (Brazilian Agency) for its financial support.

REFERENCES

[1] P. Brazdil, C. Giraud-Carrier, C. Soares, and R. Vilalta, *Meta-Learning: Applications to Data Mining*. Springer, 2008.

[2] R. B. C. Prudêncio and T. B. Ludermir, "Selective generation of training examples in active meta-learning," *International Journal of Hybrid Intelligent Systems*, vol. 5, pp. 59–70, 2008.

[3] D. Cohn, L. Atlas, and R. Ladner, "Improving generalization with active learning," *Machine Learning*, vol. 15, pp. 201–221, 1994.

[4] I. Muslea, S. Minton, and C. Knobrock, "Active learning with multiple views," *Journal of Artif. Intel. Research*, vol. 27, pp. 203–233, 2006.

[5] D. Rumelhart, G. Hinton, and R. Williams, "Learning representations by backpropagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[6] K. Smith-Miles, "Cross disciplinary perspectives on meta-learning for algorithm selection," *ACM Computing Surveys*, 2008.

[7] R. Leite and P. Brazdil, "Predicting relative performance of classifiers from samples," in *22nd Inter. Conf. on Machine Learning*, 2005.

[8] H. Bensusan and K. Alexandros, "Estimating the predictive accuracy of a classifier," in *12th European Conf. on Machine Learning*, 2001, pp. 25–36.

[9] A. Kalousis, J. Gama, and M. Hilario, "On data and algorithms - understanding inductive performance," *Machine Learning*, vol. 54, no. 3, pp. 275–312, 2004.

[10] P. Brazdil, C. Soares, and J. da Costa, "Ranking learning algorithms: Using IBL and meta-learning on accuracy and time results," *Machine Learning*, vol. 50, no. 3, pp. 251–277, 2003.

[11] C. Koepf, in *Meta-Learning: Strategies, Implementations, and Evaluations for Algorithm Selection*, 2006.

[12] L. Kuncheva, *Combining Pattern Classifiers - Methods and Algorithms*. John Wiley and Sons, New Jersey, 2004.

[13] M. Lindenbaum, S. Markovitch, and D. Rusakov, "Selective sampling for nearest neighbor classifiers," *Machine Learning*, vol. 54, pp. 125–152, 2004.

[14] L. Todorovski and S. Dzeroski, "Combining classifiers with meta decision trees," *Machine Learning*, vol. 50, no. 3, pp. 223–249, 2003.

[15] I. Teixeira, "Active cp: A method for speeding up user preferences acquisition in collaborative filtering systems," in *16th Brazilian Symposium on Artificial Intelligence*, 2002, pp. 237–247.

[16] L. Prechelt, "A set of neural network benckmark problems and benchmarking rules," Universitä Karlsruhe, Tech. Report 21/94, 1994.

[17] C. Soares, P. Brazdil, and P. Kuba, "A meta-learning approach to select the kernel width in support vector regression," *Machine Learning*, vol. 54, no. 3, pp. 195–209, 2004.