A Computational Environment for Extracting Rules from Databases

J. A. Baranauskas[1], M. C. Monard[1] & G. E. A. P. A. Batista[1]
[1] *Institute of Mathematics and Computer Sciences,*
Department of Computer Science and Statistics,
Laboratory of Computational Intelligence,
University of São Paulo at São Carlos, Brazil.

Abstract

Classification for very large databases has many practical applications in Data Mining. Thus, Machine Learning algorithms should be able to operate in massive datasets. When a dataset is too large for a particular learning algorithm to be applied, there are other ways to make learning feasible; preprocessing techniques and dataset sampling can be used to scale up classifiers to large datasets.

In this work we propose a computational environment based on two architectures, one for data pre-processing and one for post-processing which allow evaluation of induced knowledge. The two architecture share a set of learning systems, which can be enhanced to support new ones. The environment is designed as a test-bed for Data Mining research, as well as a generic knowledge discovery tool for varied database domains. Flexibility is achieved by an open-ended design for extensibility, enabling integration of existing Machine Learning algorithms, support functions for pre-processing as well as new locally developed algorithm and functions.

1 Introduction

One of the challenge in Data Mining — DM — is to exploit and combine existing Machine Learning — ML — algorithms effectively. ML algorithms are usually interested in the predictive power of the induced knowledge on new instances i.e., accuracy. Although prediction can be considered the strongest goal of Data Mining, human understanding and evaluation of the induced knowledge also plays an important role, which is often neglected.

In this work we propose a computational environment based on two architectures, one for data pre-processing and one for post-processing which allow evaluation of induced knowledge. The two architecture share a set of learning systems, which can be enhanced to support new ones. We consider domain independent data pre-processing, such as missing data treatment and noise reduction, since this sort of task can be solved by automatic techniques. Partially automating data pre-processing is important since it is very often cited as being the most time consuming step in Data Mining. One of the most important problems in data pre-processing is not destroying valuable information in the raw data.

During the pre-processing phase the system takes several samples from the original database. After that, a cyclic process of domain independent experiments with those samples that are going to be used by the set (or subsets) of learning algorithms known by the system can be carried out.

At this point the post-processing phase can be initiated: different learning algorithms could be applied to the pre-processed datasets, inducing several symbolic classifiers for each sample. Next, the induced rules are converted into a common-syntax rule set, keeping track of the ML algorithm that has generated each rule. Then, the complete rule set can be filtered and evaluated using a covering algorithm that tries to increase the predictive power of the final rule set that represents the knowledge of the original database.

The emphasis is not centered in classification performance but in understanding for explanation since often it is necessary that a potential user/especialist has an appreciation of the methods and rationale behind a classification rule. In fact, in some contexts, such as credit scoring, this is a legal requirement.

This work is organized as follows. Section 2 describes the preprocessing phase and Section 3 describes some important concepts considered when designing the post-processing phase. Section 4 outlines the architecture of the computational environment currently under development, sketching its main components. Finally, Section 5 presents some conclusions.

2 Data Preprocessing Phase

The main objective of a data analysis process is to discover knowledge that will be used to solve problems or make decisions. However, problems with the data may prevent this. Data is acquired in the form of symbolic and numeric features and from a variety of sources. These sources may vary from human beings to sensors with different degrees of reliability [13].

Data quality is a central issue in Data Mining as most learning systems are not able to consider additional information during the learning phase. In order to assure the data quality, data preprocessing techniques should be applied to the data before the learning algorithms are fed with the data.

Data preprocessing is a time consuming task, which in many cases is

semi-automatic. Growing amounts of data produced by modern data acquisitions systems has resulted in large amounts of data. Therefore, techniques for automatic data preprocessing are important. In a general way, data preprocessing can be divided into two main groups of tasks:

1. Domain specific tasks that are solved by ad hoc techniques implemented using domain knowledge, such as data consistency verification and data granularity;

2. Domain independent tasks that can be automated, such as missing data treatment, noise reduction, unbalanced dataset treatment, etc.

In our research, we are most interested in domain independent data preprocessing and how these problems can be solved by automatic techniques [3]. One of the most important problems in data preprocessing is how to know if the valuable information in the raw data is not being destroyed. Data preprocessing techniques must be carefully chosen in order to introduce a minimal amount of bias.

Although data preprocessing is not considered a glamorous task, many of the problems found in this phase are common to a number of applications. These problems can be automatically solved by a set of domain independent techniques, decreasing the overall time expended in the data preprocessing phase. Some of these problems are:

- noisy data treatment;
- missing values treatment;
- unbalanced dataset treatment;
- feature selection;
- feature construction and
- instance selection.

The computational environment here described integrates preprocessing an post-processing facilities. This environment supplies to the data analyst a set of data preprocessing techniques to identify and treat the most common domain independent data problems, such that he or she will be able to choose among techniques that introduce a minimal amount of bias in the data.

Each method of data preprocessing implemented in the computational environment can be analyzed in many real-world datasets. This analysis supplies some guidelines that help in the decision of which method is appropriate to a determined dataset, such as time requirement, robustness, insertion of bias, and others.

The handling of missing values will be used to illustrate how data preprocessing is implemented in this environment. Many learning systems are able to treat missing values automatically. However, this handling is frequently made with very simple techniques, for instance the substitution of all missing values of a feature for the its average. This substitution method may introduce some bias in the data, since the relationship between the features is not take into account.

A more efficient technique to treat missing values, used in this work, is to test how the missing values are distributed before any data manipulation. Missing values randomly distributed are considered less dangerous and can be manipulated by simple techniques, such as the removal of all instances with missing values. However, missing values not randomly distributed need to be treated more carefully, through more robust methods. A robust method for missing value manipulation consists of creating a model through a learning system to predict the missing values. Many learning systems can be used to create these models, such as neural networks and decision trees. However, this solution can be very time-consuming and some simpler models such as the k-nearest neighbor can supply faster solutions for large amounts of data.

This sort of knowledge regarding preprocessing techniques is being used to construct a simple expert system capable to guide the data analyst through a series of data preprocessing transformations. This expert system will be able to identify several data problems and suggest a variety of data preprocessing transformation based on the data problems characteristics. For instance, the expert system will identify that a dataset has some missing values, check for the randomness of the missing values and to suggest a method for treating the missing values based on the randomness and quantity of missing values as well as on the total volume of data.

3 Post-processing Phase

In supervised classification an instance is a pair $(\mathbf{X}, Y) = (\mathbf{X}, f(\mathbf{X}))$ where $\mathbf{X}$ is the input and $f(\mathbf{X})$ is the output. The task of an inducer is, given a set of training instances, to induce a function (classifier) h that accurately approximates f. In this case, h is called an *hypothesis* over f. Observe that each $\mathbf{X}$ is an element of the set $X_1 \times X_2 \times \ldots \times X_m$ where X_j is the domain of the j-th feature and Y belongs to one of the k classes *i.e.*, $Y \in \{C_1, C_2, \ldots, C_k\}$. All classifiers use stored data structures that are then interpreted as a mapping for an unclassified instance to a label [1].

In the post-processing phase, two important concepts are applied that concern classifiers: the bias plus variance decomposition of a classifier and ensembles of classifier. They will be shortly described in Sections 3.1 and 3.2.

3.1 The Bias plus Variance Decomposition

The classifier *Fundamental Decomposition* [8, 6] principle states that the classifier error can be viewed as three basic components:

1. the minimum error that can be obtained by the ideal classifier: the lower bound on the expected error of any learning algorithm;
2. the bias which measures how closely the learning algorithm's average guess, over all possible training sets of the given size that matches the target;
3. the variance which measures how much the guesses of the learning algorithm will vary with respect to each other *i.e.*, how often it fluctuates for different training sets of the given size.

This principle is given by Equation 1 where $f = B^*$ is the Bayes classifier, which gives the minimum classification error rate $\mathsf{ce}(f) = \mathsf{ce}(B^*)$. Bias and variance are always positive terms. At some data points bias predominates, at others the variance. But, in general, at each point $\mathbf{X}$ both contributions are positive.

$$\mathsf{ce}(h) = \mathsf{ce}(f) + \mathsf{bias}(h) + \mathsf{variance}(h) \tag{1}$$

This decomposition is important in comprehending the relationship between bias and variance and the behaviour of a classifier. In general, an inducer builds partitions in the description space in a certain way such that can be considered as a family of functions H. For instance, most of decision trees or rule induction inducers divide the space into rectangular regions whereas neural nets can divide the space into more complex regions. In any case, each inducer tries to select the best classifier h, using the training set, from the set of functions H.

3.2 Ensembles

An ensemble consists of a set of individual classifiers whose predictions are combined when predicting novel instances labels. Previous research has shown that an ensemble is often more accurate than any of the individual classifier in the ensemble *i.e.*, multiple classifiers have been shown to lead to improved predictive accuracy when classifying instances that are not among the training set.

There is considerable diversity in the methods used to assemble the ensembles, including stacking [24], windowing [21], bagging [7], wagging [4], and more recently boosting [22, 14, 15] and arcing [6, 8]. Usually, classifiers are combined using a majority or weighted vote mechanism. It has been suggested that both bagging and boosting reduce error by reducing the variance term in Equation 1 [8]. Also, [15] argue that boosting attempts

to reduce the error in the bias term in the equation since it focuses on misclassified instances.

Considering the two most recent ensemble methods, in general, bagging is almost always more accurate than a single classifier, but it is sometimes much less accurate than boosting. On the other hand, boosting can create ensembles that are less accurate than a single classifier. In some situations, boosting can overfit noisy datasets, thus decreasing its performance.

On the other hand, ensembles usually generate a large classifier, contrary as stated in the Ockham's razor principle [16]. For example, in [18], using the Frey-Slater letter dataset [5] with 16 numeric features and 16000 instances, it is possible to achieve very good accuracy on the test set with 4000 instances by voting 200 classifiers. Including training and test sets, the dataset requires less than 700 Kbytes. Although each individual classifier requires 295 Kbytes of memory, an ensemble of 200 classifiers requires 58 Mbytes, more than 85 times greater than the dataset.

3.3 Symbolic Ensembles

As stated in [6], ensembles are perturb and combine (P&C) algorithms. Although there are benefits in accuracy that can be obtained from the use of ensembles, since they usually reduce the error by reducing bias and variance, one problem is the interpretability by humans. It can be noted that combining symbolic classifiers into one single classifier by a majority (or other) vote mechanism does not result into a symbolic classifier any more *i.e.*, an ensemble of symbolic classifiers is no longer symbolic.

This has motivated our work on trying to use the benefits from ensembles while still keeping the symbolic component in the learning system. The next section outlines the computational environment under development.

4 Architecture

This section provides some basic ideas about our architecture since a full description is beyond the scope of this paper. The idea used in our the computational environment is to sample the original database (or to use bagging if the database is too small). For each sample, a classifier is extracted from that sample. In this step, it is possible to use several different inducers like $\mathcal{ID}3$ [20], $\mathcal{C}4.5$ and $\mathcal{C}5.0$ [21], $\mathcal{CN}2$ [10, 11, 9] and $\mathcal{R}ipper$ [12] thus leading to classifiers with different inductive bias and variance [2]. The system is flexible enough to accept any other symbolic inducer.

After the induction step, each classifier is translated into a common-syntax rule set similar to the $\mathcal{CN}2$ rule induction system. Then, rules can be evaluated using a separated test sample from the same database. It is also possible to combine all rule sets individually extracted into one final rule set, using a greedy cover algorithm.

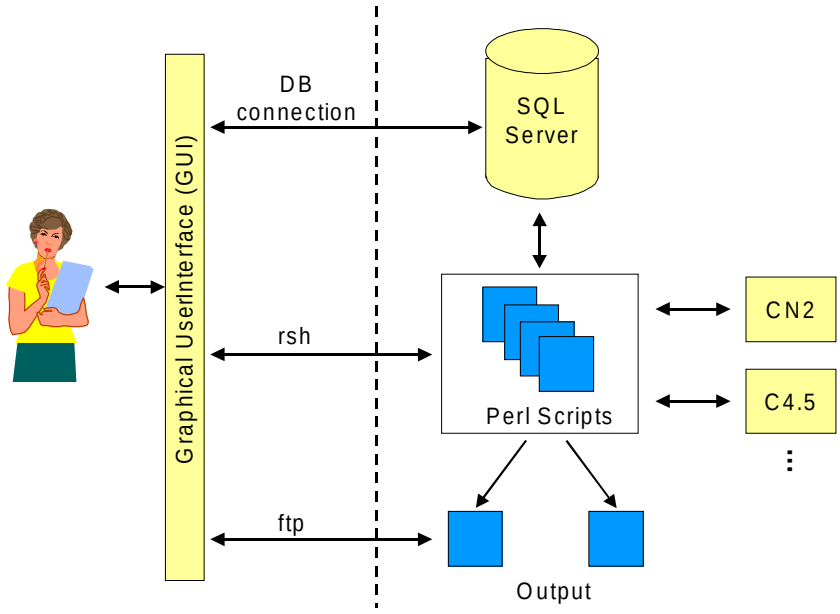

Figure 1: Computational Environment Architecture

Figure 1 shows the proposed client/server architecture. In the server side, the SQL server component maintain databases. Actually, we are using the $\mathcal{MySQL}$ server which is a multi-user, multi-threaded, multi-platform SQL database server [19]. $\mathcal{MySQL}$ is free software for non commercial applications. It is also fast and robust, easily working with millions of records and hundred of gigabytes.

The SQL server maintains datasets, classifiers as well as extracted rules in the common-syntax format. It is also possible to import/export $\mathcal{MySQL}$ tables from/to the $\mathcal{MLC}$++ format [17]. Perl scripts are responsible to perform necessary transformations among datasets, inducers, classifiers and SQL server formats [23].

In the client side, a Graphical User Interface — GUI — manages the server components. Initially, the user selects the original dataset to work with. Then, data preprocessing can take place, thus arriving at a reduced dataset. Besides a standard database connection from the client to the SQL server, the system uses `rsh` and `ftp` TCP/IP protocols to activate the perl scripts as well as to transfer resulting data between the client and server components.

One advantage of this architecture is that the server components can be spread out across several computers. This allows to achieve some degree of parallelism in some situations. For example, the induction of classifiers can be performed simultaneously in different computers. Then, results are

grouped into the SQL component for further analysis or processing. This is a desirable feature for DM tasks since gigabytes of data are frequently available.

5 Concluding Remarks

In this work we described a computational environment under development which addresses the problem of classification in DM whenever the result of learning is expressed in the form of classification rules. Our proposed framework assumes that the dataset is a normal relational table, which consists of n records described by m distinct features (categorical or continuous) that have been classified into K known classes.

Although many works have been published proposing new learning systems as well as modifications to existing ones, little attention has been given to the pre and post-processing phases. One of the possible reasons for this phenomena is the extensive use of ready-to-learn datasets for ML (as those found in data repositories [5]) and the extensive use of accuracy to evaluate the knowledge induced.

The emphasis of this environment and its software architecture is on integration of DM operations, such as database access and selection, a preprocessing phase typically interactive, ML algorithms that induce classification rules, focusing on human understanding and evaluation of the induced knowledge, as well as on extensibility.

There are many different methods for constructing classification rules although no one method is universally the best since different application domains lead to different problems, requiring different solutions. This being the case, it is expected that a combination of these methods, as proposed in this work, may yield better classification rules. Such combinations can be made in various ways. We consider simplicity of rules as a goal in itself, although simplicity does not necessarily lead to greater accuracy

Other important issue considered is that the environment should preserve a friendly and easy to use interface to enable users and domain experts to access the environment from both local and remote locations, and to comment on and evaluate result quality. This scalable and extensible environment is based on the view of Data Mining as an interactive and iterative process.

Acknowledgments: This research is partially supported by National Research Councils Finep and CAPES, Silicon Graphics Brazil as well as FMRP-USP and FAEPA-HCFMRP-USP.

References

[1] Baranauskas, J. A. and Monard, M. C. (2000a). Reviewing some machine learning concepts and methods. Technical Report 102, ICMC-USP.

ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/Rt_102.ps.zip.

[2] Baranauskas, J. A. and Monard, M. C. (2000b). An unified overview of six supervised symbolic machine learning inducers. Technical Report 103, ICMC-USP. ftp://ftp.icmc.sc.usp.br/pub/BIBLIOTECA/rel_tec/Rt_103.ps.zip.

[3] Batista, G. E. A. P. A., Carvalho, A. C. P. L., and Monard, M. C. (2000). Applying one-sided selection to unbalanced datasets. In *Proceedings of the Mexican Congress on Artificial Intelligence (MICAI), Lecture Notes in Artificial Intelligence*. Springer-Verlag. (in print).

[4] Bauer, E. and Kohavi, R. (1999). An empirical comparison of voting classification algorithms: Bagging, boosting, and variants. *Machine Learning*, 36:105–142.

[5] Blake, C., Keogh, E., and Merz, C. J. (1998). UCI repository of machine learning databases. http://www.ics.uci.edu/~mlearn/MLRepository.html.

[6] Breiman, L. (1996a). Arcing classifiers. Technical report, Statistics Department, University of California, ftp://ftp.stat.berkeley.edu/pub/users/breiman/.

[7] Breiman, L. (1996b). Bagging predictors. *Machine Learning*, 24(2):123–140.

[8] Breiman, L. (1996c). Bias, variance and arcing classifiers. Technical Report 460, Statistics Department, University of California.

[9] Clark, P. and Boswell, R. (1991). Rule induction with $\mathcal{CN}2$: Some recent improvements. In Kodratoff, Y., editor, *Proceedings of the 5th European Conference (EWSL 91)*, pages 151–163. Springer-Verlag.

[10] Clark, P. and Niblett, T. (1987). Induction in noise domains. In Bratko, I. and Lavrač, N., editors, *Proceedings of the Second European Working Session on Learning*, pages 11–30, Wilmslow, UK. Sigma.

[11] Clark, P. and Niblett, T. (1989). The $\mathcal{CN}2$ induction algorithm. *Machine Learning*, 3(4):261–283.

[12] Cohen, W. W. (1995). Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115–123, San Francisco, CA. Morgan Kaufmann.

[13] Famili, A., Shen, W.-M., Weber, R., and Simoudis, E. (1997). Data preprocessing and intelligent data analysis. *Intelligent Data Analysis*, 1(1). http://www.elsevier.com/locate/ida.

[14] Freund, Y. and Schapire, R. E. (1995). A decision-theoretic generalization of on-line learning and an application to boosting. In *Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37. Springer-Verlag.

[15] Freund, Y. and Schapire, R. E. (1996). Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning*, pages 123–140, Lake Tahoe, California. Morgan Kaufmann.

[16] Kearns, M. J. and Vazirani, U. V. (1994). *An Introduction to Computational Learning Theory*. Ellis Horwood.

[17] Kohavi, R., Sommerfield, D., and Dougherty, J. (1994). *$\mathcal{MLC}$++: A Machine Learning Library in C++*. IEEE Computer Society Press.

[18] Margineantu, D. D. and Dietterich, T. G. (1997). Pruning adaptive boosting. In *Proceedings of the Fourteenth International Conference on Machine Learning*, pages 211–218, San Francisco. Morgan Kaufmann.

[19] MySQL (2000). The $\mathcal{MySQL}$ server. http://www.mysql.com.

[20] Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, 1:81–106. Reprinted in Shavlik and Dieterich (eds.) Readings in Machine Learning.

[21] Quinlan, J. R. (1988). *C4.5 Programs for Machine Learning*. Morgan Kaufmann, CA.

[22] Schapire, R. E. (1990). The strength of weak learnability. *Machine Learning*, 5(2):197–227.

[23] Wall, L., Christiansen, T., and Schwartz, R. L. (1996). *Programming Perl*. O'Reilly & Associates, Inc.

[24] Wolpert, D. H. (1992). Stacked generalization. *Neural Networks*, 5:241–259.