Evaluating Data Mining Models: A Pattern Language

Jerffeson Souza[*] **Stan Matwin** **Nathalie Japkowicz**
School of Information Technology and Engineering
University of Ottawa
K1N 6N5, Canada
{*jsouza,stan,nat*}*@site.uottawa.ca*

Abstract

This paper extracts and documents patterns that identify recurring solutions for the problem of evaluation of data mining models. The five patterns presented in this paper are organized as a pattern language. The patterns differ in their context of application and how they solve the evaluation problem, especially when only limited amounts of data are available. Another contribution of this paper is the introduction of a new pattern section called "Force Resolution Map". We believe that Force Resolution Maps illuminate not only these data mining patterns, but are generally useful in explicating any patterns.

Keywords: Model evaluation, data mining, software patterns, force resolution map.

1 Introduction

Evaluation is one the key points in any data mining process. It serves two purposes: the prediction of how well the final model will work in the future (or even whether it should be used at all), and as an integral part of many learning methods, which help find the model that best represents the training data. In this paper, we deal specifically with the first case, how well the model will work.

[*]Sponsored by CAPES (Brazilian Federal Agency for Graduate Studies).

In most cases, prediction is not as simple as it might appear. The main problem occurs when the data used for training a data mining model are used in estimating the performance of that model. This generates a non-realistic and overoptimistic prediction in a manner considered unacceptable by the data mining community.

When a vast supply of data is available, this problem is minimised, since one could use a large slice of data for training and would still have another large portion to be used as test data. The pattern *Hold-Out* deals with this situation. On the other hand, when only a limited amount of data is available, more complex methods are necessary to achieve an unbiased estimate of the model performance over unseen examples. Patterns *K-fold Cross-validation*, *Leave-one-out* and *Bootstrap* present three alternatives for such evaluation, depending on the number of examples available. The pattern *Model Evaluation* guides a designer in choosing the best technique in a particular context.

These patterns are intended for novices in the data mining world trying to understand the essence of data mining evaluation, for experienced data miners dealing with complex mining projects, and as a reference for experts.

In the next section, we describe how the patterns are organized. Then we propose a new pattern section called "Force Resolution Map", and describe its motivation, notation and applicability. Next, we present an overview of the pattern language, as well as the patterns themselves. Finally, we offer some conclusions.

2 Guidelines for the Readers

The pattern form used here consists of nine sections. The *Context* section shows the situations in which the pattern may be applied. The *Problem* section presents the problem the pattern solves, expressed as a question. The *Forces* section describes the driving forces behind possible solutions. The *Solution* section presents an answer to the problem that resolves the forces as well as possible. The *Rationale* section explains the rationale behind the solution. The *Resulting Context* section describes the context that obtains after the pattern has been applied. In the *Related Patterns* section, we list other patterns that have some relationship with this pattern and describe those relationships. The *Known Uses* section shows where the pattern has been applied. Finally, we introduce the new section *Force Resolution Map* for a graphical presentation of how the forces are solved.

Some sections are not present in some patterns.

The problem and the solution sections are sufficient for an overview of the pattern. The other sections explain the rationale behind the pattern and help readers gain a deeper insight about it.

A detailed description of the new Force Resolution Map section follows. This includes both the rationale for using these maps and instructions on how to do so.

3 Force Resolution Map (FRM): A New Pattern Component

3.1 The Role of FRMs

The pattern form makes a solution more understandable and consequently more reusable. For a reader to be able to reuse a pattern, he/she needs to understand the tradeoffs presented by the solution in order to predict the behaviour of the system after the pattern is applied. The *Forces* section describes these considerations. The *Rationale* section clarifies how the pattern solution balances its forces, i.e., what tradeoffs were considered. From that description, one can predict the behaviour of the applied pattern and check if such behaviour is consistent with system requirements. However, the textual form of the *Rationale* section allows unclear and incomplete descriptions on how the forces were solved.

In this context, we propose a graphical tool called Force Resolution Map (FRM). FRMs provide a clear and precise description of how the solution resolves all of its forces. A reader can use an FRM to more easily predict the behaviour of a system that applies the pattern it augments. FRMs can thereby help a user decide if a pattern fits his/her requirements and could be applied, and if so, whether it is the best candidate. FRMs can also provide the basis for automatic support regarding what pattern to apply.

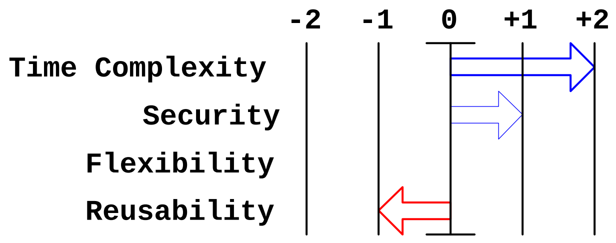
3.2 The Graphical Representation

Force Resolution Maps apply a numeric representation to indicate how a pattern behaves in respect of its forces. With values ranging from -2 to +2, each indicator represents how a force influences this pattern in comparison to other solutions for the same problem. A more detailed description on each numeric meaning follows:

- -2 : the pattern behaves more strongly in the opposite direction of the force than in other solutions.
- -1 : the behaviour of the pattern follows the force indication less than other solutions.
- 0 : the pattern has a behaviour according to this force that is comparable with most of the alternative solutions.
- +1 : the indication of this force has a relatively higher influence on how the pattern behaves.
- +2 : the pattern follows that force strongly in its behaviour, more than most of the other solutions.

3.2.1 Example of FRM

Following we present an example of a FRM and then an explanation on how to read this map:



The pattern with the above FRM presents a solution that is very complex in terms of time compared to any other solution for the same problem, with a level of security stronger than most of the alternative solutions. However, it presents itself as not so reusable, but as flexible as others.

3.2.2 Setting the Values for the Forces in a FRM

The process of choosing a precise value for each force in a FRM is essential to make that FRM representative of the expected behaviour of a system that applies this pattern.

The most important aspect in this process is that the values assigned to the forces are relative to other solutions. That is, when creating a FRM for a pattern, the writer must consider alternative solutions for the problem in order to analyse the relative influence of each force. Consider the example below:

Suppose a developer is creating a FRM for a pattern about the “Bubble Sort” algorithm, created to order a collection of elements. The first step would be to consider other solutions for the same problem and compare them with “Bubble Sort” in respect to each of the considered forces. He/she could think of “Insertion Sort”, “Selection Sort”, “Merge Sort”, “Quick Sort” and so on. In making this comparison, these two forces must receive a value for the FRM:

- **Simplicity:** compared to the other algorithms, the Bubble Sort is among the most simple. Insertion Sort seems to be the simplest one. In this situation a good choice for this force would be “+1”.
- **Time Efficiency:** This algorithm is notoriously inefficient. One can easily consider it to be the slowest among all sorting methods. A good value for this force would be then “-2”.

3.3 Validating Patterns and Pattern Language with FRMs

In order to be applied by a user, a pattern should solve his or her problem encountered in the context in which it occurs, and should meet the non-functional requirements such as speed, accuracy, maintainability, time complexity and so on. These requirements are not part of the problem *per se* but must be considered when choosing one solution over another. The *Context* and *Problem* sections in a pattern state clearly in which situations the pattern should be applied. Thus, the user can check these two sections against his or her needs. However, for a user to understand the tradeoffs considered by the pattern he/she needs to go through the *Rationale* section. The textual form of such section bars a fast validation, and allows an ambiguous and incomplete description of the tradeoffs.

A graphical and precise description, as provided by Force Resolution Maps, would allow faster validation of the pattern against the user requirements. And, for pattern writers, such representation avoids ambiguous descriptions of the pattern tradeoffs.

Another important use of FRMs is in the possibility of predicting the behaviour of a pattern language. The numeric description of how the pattern resolves a particular force allows FRMs to be combined to generate a FRM for a whole pattern language. These combined FRMs are possible for pattern languages intended to build systems. As such, they do not apply to the pattern language presented here.

Intermediate FRMs can also be created to represent the behaviour of parts of the language. For example, suppose someone is interested in creating a client-server application where most of the processing should be done on the server side. A good solution would require a thin client and a powerful server. When looking for patterns with such behaviour, one could compose all patterns for the client side to check the particular behaviour of that part and do the same for the server side.

Automatic support is another possibility with Force Resolution Maps. For this, a user could set values for each one of the forces to represent the expected behaviour for a solution. With these values, a system would be able to search a base of patterns for one that best fits this expected behaviour by trying to match the values for each one of the forces. However, this validation would be only one step in the process of deciding on a pattern to apply, since decisions regarding whether patterns appear in the expected context and solve the faced problem are still human-made.

4 The Pattern Language

4.1 Overview

The pattern language presented in this paper has five patterns. The pattern *Model Evaluation* is the main one. It helps decide which one of the other four patterns should be applied. Figure 1 describes this relationship between the patterns.

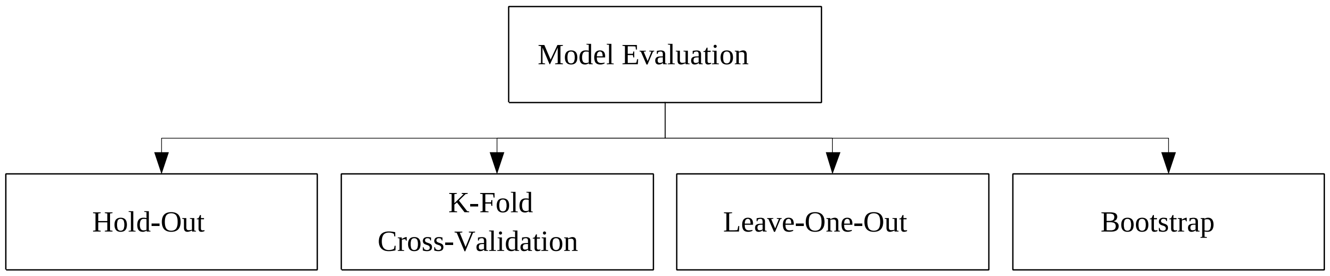


Figure 1: The Pattern Language Structure

In the figure above, each arrow represents the relationship between two patterns where the pattern in the beginning of the arrow is applied before the one in the end of the arrow.

The table below summarises the solutions for all patterns of this pattern language.

Pattern	**Solution**
Model Evaluation	1. If a separate testing set is available apply *Hold-Out*. 2. Otherwise: - If more than 100 examples are available, apply *K-fold Cross-validation* with $K = 10$. - With fewer than 100, apply *Leave-One-Out*. - With fewer than 50 and Leave-One-Out resulting in an estimate for the error rate less than 0.632, apply *Bootstrap*.

Pattern	**Solution**
Hold-Out	1. Separate the available examples into training set (examples used for training) and testing set (the rest of the data). 2. Calculate the accuracy over the testing set.
K-fold Cross-validation	1. Partition randomly the available examples into k disjoint subsets. 2. Use one of the partitions as a testing set to evaluate a model generated by considering the other subsets as the training set. 3. Repeat this process with all subsets and average the obtained errors.
Leave-one-out	1. Use one of the examples to test a model generated by considering the other examples. 2. Calculate judgement for this model: 0 if model and example are consistent, 1, otherwise. 3. Repeat these two steps for all examples and average the obtained judgement values.
Bootstrap	1. Sample a subset of m (number of instances in the dataset) instances from D (the original dataset) with replacement. 2. Train and evaluate a model with this subset, calculating the *e(training)* (error over this subset) and *e(testing)* (over the remaining examples) 3. Estimate the final error $e = 0.632 \cdot e(training) + 0.368 \cdot e(testing)$. 4. Repeat these steps m times and average the obtained error estimates.

4.2 The Patterns

4.2.1 Model Evaluation

Context

- During a data mining process where a certain amount of data D is available, with m examples, you encounter a model M generated from D to be used in an application.

Problem

- How to estimate the error rate of that data mining model?

Forces

- **Accuracy** - You want to be as accurate as possible in the evaluation.
- **Speed** - You don't want to consume a great amount of time in the evaluation.
- **Flexibility** - You want to use a method for evaluation that is flexible.

Solution

- If the model M was trained by using only a partition of D as the training set, apply the pattern *Hold-Out*.
- If the model M was trained with the whole data D, then:

  - If the amount of data D available is not so small, say more than 100 examples, apply the pattern *K-fold Cross-validation* with $K = 10$.

  - For a small D, with fewer than 100 examples, apply *Leave-One-Out*.

  - However, in case where the number of examples is fewer than 50 and *Leave-One-Out* results in an estimate for the error rate of M less than .632, apply *Bootstrap*.

Rationale

In the case where only part of the data was used for training and still another large partition is available, this partition should be used for testing. The fact that none of the examples are used for both training and testing avoids biased estimates.

On the other hand, when a model has used all the data for training, more complex methods are necessary to diminish the effect of bias. For models generated from sufficient examples, *K-Fold Cross-validation* provides a relatively fast and still precise estimate of the performance of such models.

With an even smaller amount of data, we could apply more precise techniques without affecting the time performance of the system. *Leave-One-Out* is a specialisation of *K-Fold Cross-validation* that sets K to 1. By doing this, a precise estimate can be obtained. And since the number of iterations would be small (due to the small number of examples), this method would still provide good time performance.

Finally, for really small number of examples, it is necessary to increase the number in a controlled way so that a more precise estimate can be obtained. *Bootstrap* works exactly with this situation. However, since empirical studies have shown that *Bootstrap* performs better for datasets with relatively low true error rates, its use is suggested only when the error rate estimated by *Leave-One-Out* is less than .632 (The actual number .632 was extracted from [6], as described in 4.2.5).

Resulting Context

The mining model has been evaluated and its error rate estimated.

Related Patterns

One of the patterns *Hold-Out*, *K-fold Cross-validation*, *Leave-one-out* and *Bootstrap* is now applied.

4.2.2 Hold-Out

Context

- During a data mining process where a certain amount of data D is available, with m examples, you encounter a model M generated from a subset (the training set) of D to be used in an application.

Problem

- How to estimate the error rate of that data mining model?

Forces

- **Accuracy** - You want to be as accurate as possible in the evaluation.
- **Speed** - You don't want to consume a great amount of time in the evaluation.
- **Flexibility** - You want to use a method for evaluation that is flexible.

Solution

- Follow the procedure below to estimate the error for the model M :

  1. Separate the m available examples into two subsets: the *training_set* (examples used for training M) and the *test_set* (the rest of the data).
  2. Calculate the accuracy of M over the *test_set* to estimate the error for the model M.

Rationale

The use of a separate dataset for testing avoids biased estimations.

Even though it is very fast in evaluating a model, this technique can present serious problems when the set of examples used for testing is not representative. For example, if the distributions of attribute values in the full dataset and in the test set are different, the estimation can be very negatively affected.

Resulting Context

The mining model has been evaluated and its error rate estimated.

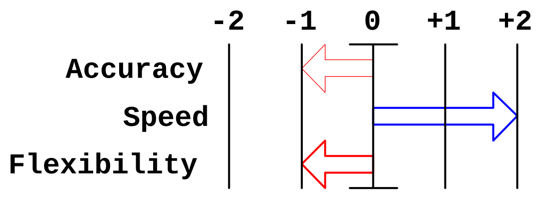
Related Patterns

The patterns *K-fold Cross-validation*, *Leave-one-out* and *Bootstrap* also deal with the problem of evaluating data mining models. They treat cases where the model was trained with all available data.

Known Uses

The implementation of this pattern can be found in data mining tools such as CART [11], Clementine [12], Weka [13], CBA [3] and $\mathcal{MLC}$++Library [7].

Force Resolution Map



4.2.3 K-fold Cross-validation

Context

- During a data mining process where a certain amount of data D is available, with m examples (m more than 100), you encounter a model M generated from D to be used in an application.

Problem

- How to estimate the error rate of that data mining model ?

Forces

- **Accuracy** - You want to be as accurate as possible in the evaluation.
- **Speed** - You don't want to consume a great amount of time in the evaluation.
- **Flexibility** - You want to use a method for evaluation that is flexible.

Solution

- Follow the procedure below to estimate the error for the model M :

  1. Partition randomly the m available examples into k disjoint subsets $s_1, s_2, s_3, ..., s_k$, each with m/k examples.
  2. Use one of the partitions, say s_i, as a testing set to evaluate a model M_i generated by considering the other $k-1$ subsets as training set. Suppose you obtained e_i as the error estimate for M_i.
  3. Repeat step 2 k times until you have used all subsets as testing sets, in order to obtain the error estimates for all k models, say $e_1, e_2, e_3, ...e_k$.
  4. Average the obtained errors calculated in all k iterations to estimate $e = \sum_{1<i<k} e_i/m$, the error for the model M.

Rationale

The main problem of evaluation of data mining models generated from a limited amount of data come from the fact that data used for creating (training) the model should not be used to evaluate the model, since such evaluation would present biased results.

The split of the dataset in k partitions, as described above, allows for the training of k different models and the testing of such models without any example being used for training and testing within a same model. This way the accuracy of a model generated from the whole dataset can be estimated by the average of the accuracy of all k models. This method has been proven to provide good estimates of the accuracy [8].

The flexibility in choosing the parameter k allows a miner to optimise the evaluation process. For example, one could pick a big k to increase the accuracy of the evaluation, however, with an increase in time consumed.

Resulting Context

The mining model has been evaluated and its error rate estimated.

Related Patterns

The two patterns *Leave-one-out* and *Bootstrap* solve the same problem in similar contexts.

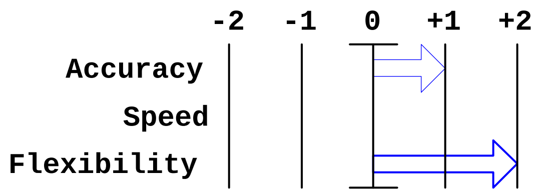
In the case where k is set to be m, the number of training examples, this pattern is essentially identical to *Leave-one-out*.

Yet the pattern *Hold-Out* deals specifically with data mining models generated from only a partition of the available data.

Known Uses

The implementation of this pattern can be found in data mining tools such as CART [11], Clementine [12], Weka [13], CBA [3] and $\mathcal{MLC}$++Library [7].

Force Resolution Map



4.2.4 Leave-one-out

Context

- During a data mining process where a certain amount of data D is available, with m examples (m less than 100), you encounter a model M generated from D to be used in an application.

Problem

- How to estimate the error rate of that data mining model?

Forces

- **Accuracy** - You want to be as accurate as possible in the evaluation.
- **Speed** - You don't want to consume a great amount of time in the evaluation.
- **Flexibility** - You want to use a method for evaluation that is flexible.

Solution

- Follow the procedure below to estimate the accuracy of the model M :

  1. Use one of the examples i (belonging to the class c_i) to test a model M_i generated by considering the other $m - 1$ examples in D for training.
  2. Calculate judgment j_i for M_i:

     - $j_i = 0$ if M_i correctly classifies i, i.e., M_i classifies i as belonging to the class c_i (the actual class of i).

     - $j_i = 1$ otherwise.
  3. Repeat 1) and 2) m times, in order to obtain the error estimate for all m models $M_1, M_2, M_3, ..., M_k$, say $j_1, j_2, j_3, ...j_k$.
  4. Average the obtained judgment values calculated in all m iterations to calculate $e = \sum_{1<i<m} j_i/m$, the error estimate for the model M.

Rationale

This pattern can be seen as a special case of the pattern *K-fold Cross-validation* when setting the number of partitions k for the total number of available examples m. However, due to this choice, a different behaviour can be expected, as seen in the Force Resolution Map of this pattern.

The lack of flexibility can cause problems when a larger number of examples are available. As the number of iterations grow, so would the time necessary to evaluate a model.

On the other hand, this approach obtains high accuracy.

Resulting Context

The mining model has been evaluated and its error rate estimated.

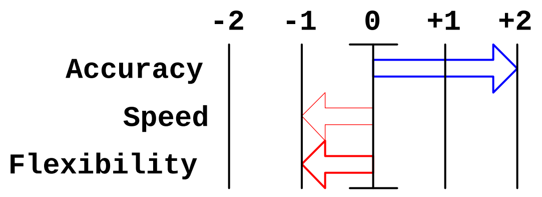
Related Patterns

The two patterns *K-fold Cross-validation* and *Bootstrap* solve the same problem in similar contexts. *K-fold Cross-validation* can be considered a generalisation of this pattern. *Hold-Out*, though, deals specifically with data mining models generated from only a partition of the available data.

Known Uses

The implementation of this pattern can be found in data mining tools such as CART [11], Clementine [12], Weka [13], CBA [3] and $\mathcal{MLC}$++Library [7].

Force Resolution Map



4.2.5 Bootstrap

Context

- During a data mining process where a certain amount of data D is available, with m examples (m less than 50), you encounter a model M generated from D to be used in an application. In addition, the error rate of M has been estimated by *Leave-one-out* to be less than 0.632.

Problem

- How to estimate the error rate of that data mining model?

Forces

- **Accuracy** - You want to be as accurate as possible in the evaluation.
- **Speed** - You don't want to consume a great amount of time in the evaluation.
- **Flexibility** - You want to use a method for evaluation that is flexible.

Solution

- Follow the procedure below to estimate the accuracy of the model M :

  1. Sample a subset of m (the number of examples in D) instances from D with replacement.
  2. Use this subset to train a model M_i.
  3. Calculate the error rate over the training set, say $e(training)_i$.
  4. Use the remaining examples in D to evaluate M_i, calculating the error estimate $e(testing)_i$ for M_i over this testing set.
  5. Estimate the final error e_i for M_i by calculating $e_i = 0.632 \cdot e(training)_i + 0.368 \cdot e(testing)_i$.
  6. Repeat 1),2),3),4) and 5) m times, in order to obtain error estimates values $e_1, e_2, e_3, ..., e_m$.
  7. Average the obtained error estimates calculated in all m iterations to estimate $e = \sum_{1<i<k} e_i/m$, the error for the model M.

Rationale

The creation of a number a datasets from the original one decreases the influence of any bias caused by the small number of examples available. However, it is expected that this solution would require a great amount of time with larger original datasets.

The use of the linear combination $0.632 \cdot e(training)_i + 0.368 \cdot e(testing)_i$ proposed by Efron in [4] is intended to balance the influence of the optimistic $e(training)_i$ and the pessimistic $e(testing)_i$ estimates. The number 0.632 represents the percentage of different instances selected for training by the method describe above. Consequently, 0.328 is the rest of the data used for testing. Such combination has been proved to generate very precise estimates when dealing with small datasets.

Resulting Context

The mining model has been evaluated and its error rate estimated.

Related Patterns

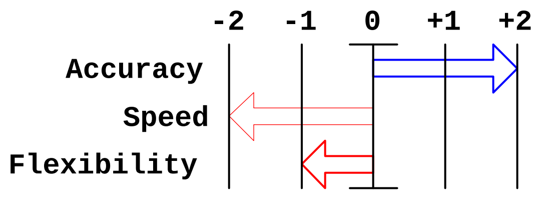
The two patterns *K-fold Cross-validation* and *Leave-one-out* solve the same problem similar contexts.

Hold-Out deals specifically with data mining models generated from only a partition of the available data.

Known Uses

The implementation of this pattern can be found in data mining tools such as Weka [13] and $\mathcal{MLC}$++Library [7].

Force Resolution Map



5 Conclusions

This paper presented a pattern language that deals with the problem of evaluation of data mining models. It concentrates on the issues that arise when only a small amount of data is available.

Force Resolution Maps were introduced as a new pattern section in order to graphically show how the forces in that patterns were resolved. FRMs allow fast and automatic validation of both patterns and pattern languages, and avoid unclear and incomplete descriptions of force resolution.

6 Acknowledgments

We would like to thank our shepherd Paul Asman for his relevant comments on this paper, we truly believe his participation was essential to make the ideas "come together" in this final version. Finally, we acknowledge CAPES (Brazilian Federal Agency for Graduate Studies) for its financial support.

References

[1] Bailey, T. L. and Elkan, C. Estimating the accuracy of learned concepts, in R. Bajcsy, ed., In Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann, pp. 895– 900, 1993.

[2] Blum, A. Kalai, A. and Langford, J. Beating the hold-out: Bounds for k-fold and progressive cross-validation. In Proceedings of the International Conference on Computational Learning Theory. 1998.

[3] DM-II Group. CBA, School of Computing, National University of Singapore, Singapure, 1998. http://www.comp.nus.edu.sg/∼dm2/

[4] Efron, B. Estimating the error rate of a prediction rule: some improvements on cross-validation. Journal of the American Statistical Association 78, 316-331. 1983.

[5] Efron, B. and Tibshirani, R. Cross-validation and the bootstrap: Estimating the error rate of a prediction rule. Technical Report (TR-477), Dept. of Statistics, Stanford University. 1995.

[6] Gamberger, D. and muc, T. DMS Tutorial. Data Mining Server [http://dms.irb.hr/]. Zagreb, Croatia: Rudjer Boskovic Institute, Laboratory for Information Systems. 2002.

[7] Kohavi, R. Sommerfield, D. and Dougherty, J. Data Mining using MLC++: A Machine Learning Library in C++. Tools with AI '96, 234-245, November 1996. http://www.sgi.com/tech/mlc/

[8] Kohavi, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pages 1137–1143. San Mateo, CA: Morgan Kaufmann, 1995.

[9] Martin, J. K. and Hirschberg, D. S. Small sample statistics for classification error rates, I: error rate measurements. Technical Report 96-21, Department of Information and Computer Science, University of California, Irvine, 1996.

[10] Mitchell, T. M. Machine Learning, McGraw-Hill, 1997.

[11] Salford Systems. CART, Salford Systems, San Diego, CA, 1997. http://www.salford-systems.com/

[12] SPSS Inc. Clementine, SPSS Inc, Chicago, IL, 1998. http://www.spss.com/spssbi/clementine/

[13] Waikato ML Group. Weka, Departamemnt of Computer Science, University of Waikato, New Zealand, 1997. http://www.cs.waikato.ac.nz/∼ml/weka/

[14] Witten, I. H. and Frank, E. Data Mining: Practical Machine Learning Tools and Techniques with Java Implementation, Morgan Kaufman, 1999.