INDIANA UNIVERSITY
BLOOMINGTON

Manual for EAR4 and CAAR Weka Plugins

Case-Based Regression with Ensembles of Adaptations

Version 1.0

Vahid Jalali and David Leake
vjalalib@umail.indiana.edu, leake@indiana.edu

Technical Report 717
School of Informatics and Computing
Indiana University

April 25, 2015

Abstract

EAR4 and CAAR are lazy learners applying the case-based reasoning (CBR) paradigm to numerical prediction tasks. Both augment standard instance-based learning methods by applying automatically generated case adaptation rules to adjust solutions of prior cases, and both apply ensembles of the generated rules. CAAR augments the EAR approach with a richer treatment of case context, more context-aware rule generation, and context-sensitive ranking of the generated adaptation rules. This manual describes installation and use of plugins enabling use of EAR4 and CAAR within the Weka workbench for machine learning.

1 Introduction

This document provides a brief overview of the lazy learning approaches EAR4 [1] and CAAR [2], and provides step by step instructions for using the plugins based on these two methods for regression (numerical prediction) tasks within the Weka workbench [3]. Both EAR4 and CAAR apply the case-based reasoning (CBR) paradigm (e.g., [4, 5, 6, 7]) to regression, reusing and adapting previous problems' solutions for solving similar new problems. To clarify the concepts underlying these learners, Section 2 provides an introduction to points that are common between both learners. Section 3 describes EAR4, with instructions for running its plugin in Section 4, and Sections 5 and 6 describe CAAR and running its plugin.

2 Background

2.1 Case-Based Regression

Case-based regression computes the numerical solution of an input problem by using the values of its nearest neighbors. In this regard case-based regression is very similar to instance-based learning (*e.g.*, IBk [8]). However, case-based regression differs from instance-based learning in its emphasis on CBR's *case adaptation* step to revise prior solutions to new problems. Instance-based systems normally predict the value of an input query directly from the solutions of the top k nearest neighbors, applying a combination function such as distance-weighted averaging. In contrast, before combining the nearest neighbors' values, EAR4 and CAAR both first apply *case adaptation rules* to those values, to adjust them based on differences between the old and new problems. However, their difference lies in their method of generating and retrieving adaptation rules. While EAR4 relies on cases in the local neighborhood of the input query for generating adaptation rules, CAAR generates adaptations from the individual cases' neighborhoods. Also, CAAR considers context in defining and retrieving the adaptation rules, while the context of cases and rules is not considered in EAR4's method.

Fig. 1 illustrates the generic case-based regression process. Given a new problem, case-based regression generates a solution by retrieving a set of cases for similar problems, adjusting the solution values of the retrieved cases (which we will call source cases), and then combining the adjusted values to generate the final predicted value.

2.2 Case Difference Heuristic

EAR4 and CAAR generate adaptation rules automatically from the cases in its case base, using a domain-independent method. This enables them to use case adaptation without requiring costly manual rule acquisition. In addition, they exploits the availability of multiple automatically-generated adaptation rules by applying an ensemble-based adaptation method: They apply a set of adaptation rules to each adaptation problem. These methods use *Case Difference Heuristic* [10] method to derive adaptation rules. The case difference heuristic method compares pairs of cases in the case base, generating a rule for each pair. Intuitively, it ascribes the observed difference in the solution values to the

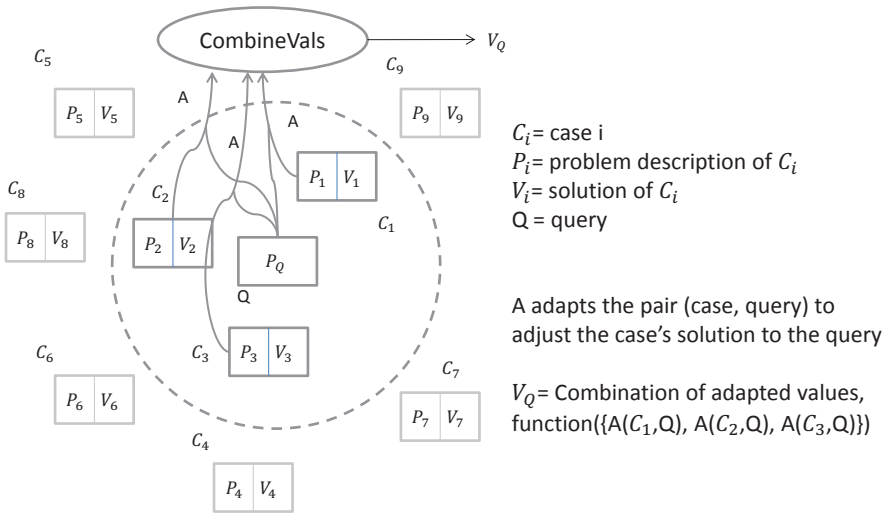

Figure 1: Illustration of the generic case-based regression process [9]

differences in the two problems. More specifically, the differences in the input features of the pair of cases form the antecedent part of the adaptation rule and the difference in their solutions forms the consequent. Fig. 2 depicts a simple example of rule generation and application of the generated rule, for the sample task of predicting automobile gasoline mileage (MPG, Miles Per Gallon). Part a of Fig. 2 shows how an adaptation rule is generated based on a pair of cases and part b shows the application of the generated rule to adjust the value of a nearest neighbor case.

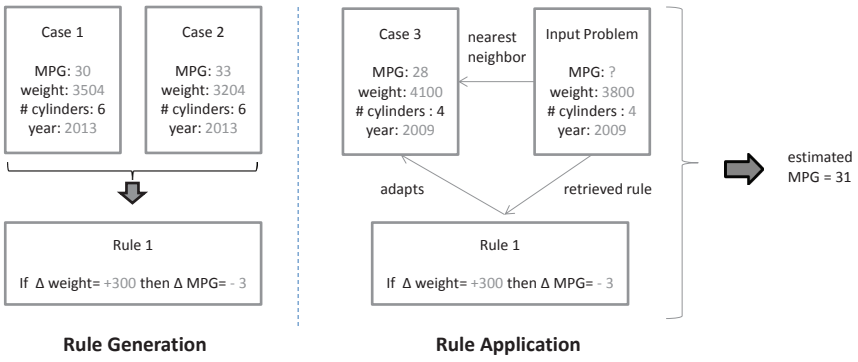

Figure 2: Illustration of using the case difference heuristic to generate an adaptation rule and to generate an MPG estimate.

Different pairs of cases may differ in the similar ways, so multiple adaptation rules could be generated to address any given difference. Adaptation rules can

be generated from pairs of cases from different parts of the domain, potentially resulting in different rules depending on the regions from which the pairs are drawn, or even the specific pairs chosen.

3 A Quick Sketch of EAR4's Approach

EAR4 is a lazy learner introduced by Jalali and Leake [1]. EAR4 generates adaptation rules by applying the case difference heuristic to local information, comparing the top nearest neighbors of the input query when that query is presented to the system, for lazy adaptation rule generation. To take advantage of multiple adaptation rules, it adjusts solution values of source cases using an ensemble of rules.

For full details about EAR4's underlying process, a discussion of related work, and comparative evaluations of EAR4's performance, we refer the reader to research publications on EAR4 (*e.g.*, [1, 9]).

4 How to Use EAR4 in Weka

4.1 Downloading EAR4 for Weka

EAR4's Weka plugin currently works for domains with numeric input features and target values and can be downloaded in any of the following ways:

- Install using Weka plug-in manager: EAR4 is part of Weka's official repository. If your Weka version is greater than 3.7.2, under "Tools" menu you can find and select "Package Manager". In the dialog page that opens, you will see the list of packages in Weka's repository. Look for the entry for EAR4 and install it. You may need to relaunch Weka after installing the package.

- Install from file: You can also download EAR4's plug-in directly from the EAR4 project's home page[1] to your local machine. Launch "Package Manager", choose the "File/URL" button, and select the downloaded archive file (it should be titled EAR4.zip) from your local machine. After installing the package you may need to relaunch Weka.

- Binaries: We have also compiled two versions (3.6.11 and 3.7.11) of Weka binary files with EAR4 learner embedded in them. You can directly download and use EAR4 in binary files for version 3.6.11[2] and for version 3.7.11[3].

The source code of EAR4 can also be found in the plugin's archive[4]. The source is located in the source folder inside the archive. For those who wish to modify the source code or compile it, Weka instructions are provided at How to compile Weka[5].

[1] http://sourceforge.net/projects/ear4/files/EAR4.zip/download
[2] http://sourceforge.net/projects/ear4/files/weka3611.jar/download
[3] http://sourceforge.net/projects/ear4/files/weka3711.jar/download
[4] http://sourceforge.net/projects/ear4/files/EAR4.zip/download
[5] http://weka.wikispaces.com/How+do+I+compile+WEKA%3F

4.2 Running EAR4 in Weka

After installing EAR4 and launching Weka (or launching Weka binary with EAR4 embedded), you will see Weka GUI as shown in Fig. 3.

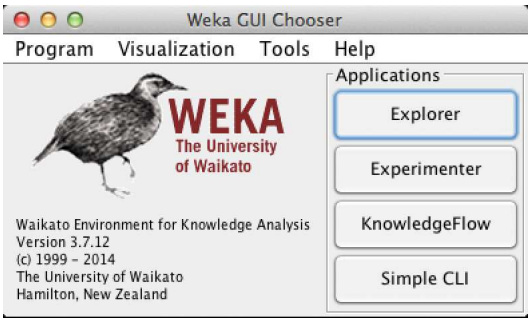

Figure 3: Weka GUI.

Select the Explorer button and Weka Explorer will be launched as shown in Fig. 4. Select the open file button in Weka Explorer and choose the data set to be tested. You can download a sample Weka data set[6] from EAR4's homepage. The data set is a cleaned version of the MPG data set from the UCI Machine Learning Repository [11].

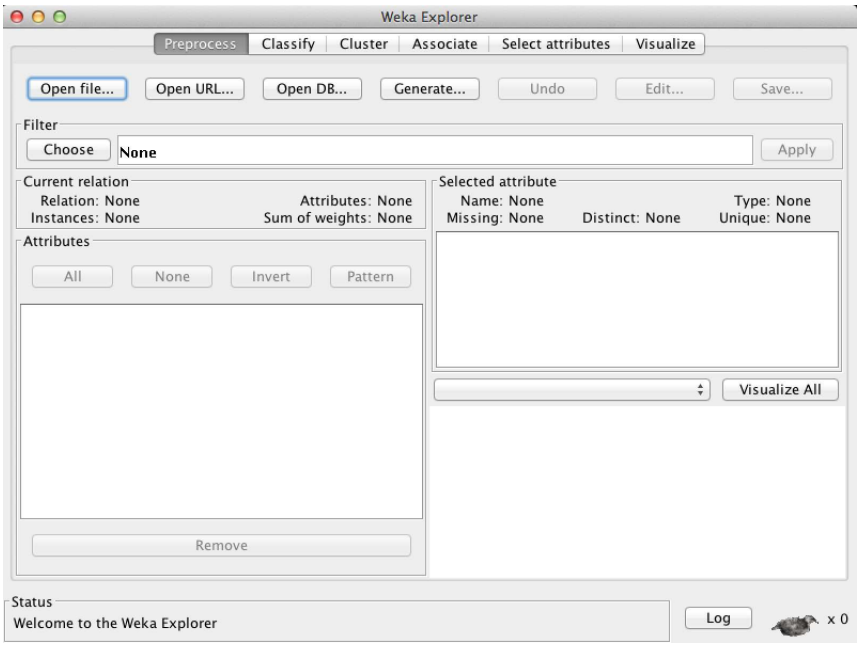

Figure 4: Weka Explorer.

Next choose the Classify menu in Weka Explorer. Weka will display the

[6] http://sourceforge.net/projects/ear4/files/mpg.arff/download

available classifiers, as shown in Fig. 5. EAR4 will be listed in the "lazy" classifiers category.

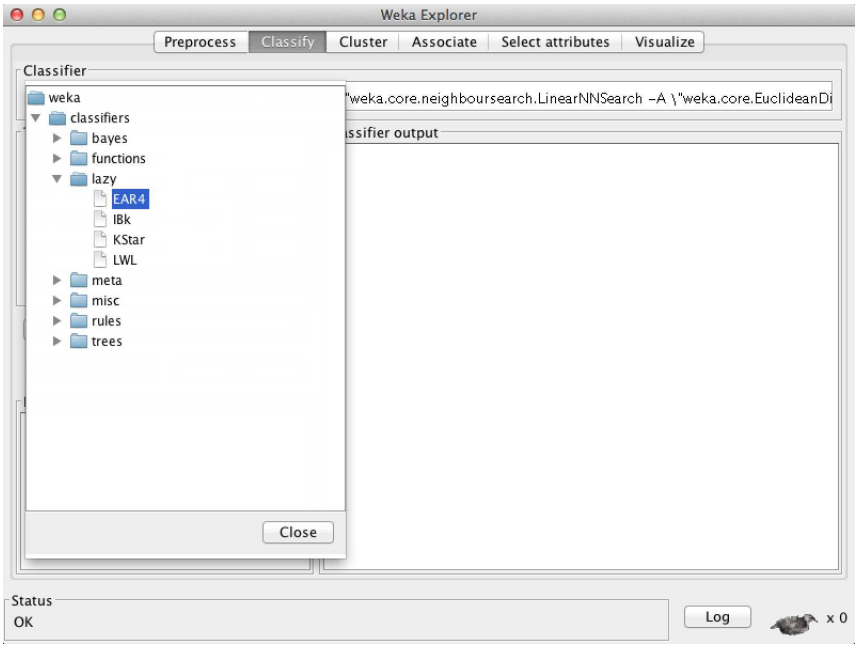

Figure 5: Weka Classifiers.

Choose EAR4 as the classifier. The parameters used by EAR4 can be tuned by clicking on the text box next to the Choose button as shown in Fig 6.

Fig 6 shows the parameters that can be set for EAR4:

- kNN specifies the number k of nearest neighbors for k-NN to use in predicting the solution.
- r specifies the number of adaptation rules to be applied for adjusting the value of each nearest neighbor (source case).
- o is a coefficient determining the number of cases used for generating adaptation rules. EAR4 generates adaptations based on the local neighborhood of the input query. The number of nearest neighbors of the input query from which adaptations are generated is $kNN \times o$. The default value of o is one, so that by default, adaptation rules are generated based on the top kNN nearest neighbors of the input query.
- $nearestNeighbourSearchAlgorithm$ enables selection of the similarity measures used for retrieving the source cases and $ruleNearestNeighbourSearchAlgorithm$ is used to retrieve the adaptations rules to apply.

Figure 6 shows sample values for EAR4 parameters. In this case, EAR4 estimates the input query's value by combining the adjusted values of the five

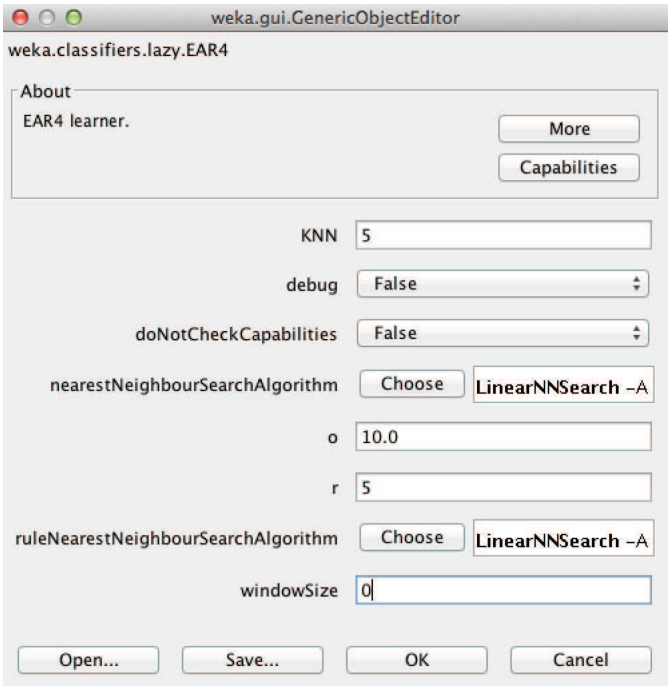

Figure 6: Setting EAR4's parameters.

nearest neighbors of the input query. The value of each of those nearest neighbors is adjusted by applying five adaptation rules. Adaptation rules are built from the top 50 (*i.e.*, 5×10) cases of the input query.

Note that values of EAR4's parameters can be tuned by hill climbing on the training data using cross validation, as has been done in experimental evaluations of EAR4. However, this process is not currently implemented in the Weka version of EAR4, so it is up to the user to determine parameter tunings for a particular data set.

5 A Quick Sketch of CAAR's Approach

CAAR is a lazy learner, introduced by Jalali and Leake [2], that uses a context-aware method for retrieving adaptations. CAAR's underlying approach is very similar to that of Ensemble of Adaptations for Regression (EAR) [1] in that EAR also adjusts the instance-based solutions by applying an ensemble of adaptations and combines the adjusted values for building the final solution. Like EAR, CAAR generates adaptation rules by using the *case difference heuristic*.

CAAR builds adaptations by comparing each case in the case base with its top l neighbors. It defines the context of a case as the rate of change of the target value given changes in the input features. CAAR determines the context of a case by training a linear regression model based on its top c nearest neighbors. In order to assess the appropriateness of an adaptation rule to adapt a similar retrieved case, CAAR computes and uses the context of the "composing cases"

of that adaptation (the cases from which the adaptation was generated), and the context of the input query and the retrieved case. Therefore, the values of the retrieved cases are adjusted by applying adaptations expected to address the contextual differences between the input query and the retrieved case and their effect on their solution differences. More details about calculating context and the adaptation retrieval process can be found in [2].

CAAR retrieves the top k nearest neighbors of the input query and applies r adaptation rules for adjusting the values of each of those nearest neighbors. The final solution is built by averaging the adjusted values of the nearest neighbors.

6 How to Use CAAR in Weka

6.1 Downloading CAAR for Weka

CAAR's Weka plugin currently works for domains with numeric input features and target values. The plugin can be downloaded in any of the following ways:

- Install using Weka plug-in manager: CAAR is part of Weka's official repository. If your Weka version is greater than 3.7.2, under "Tools" menu, find and select "Package Manager". In the dialog page that opens, you will see the list of packages in Weka's repository. Look for the entry for CAAR and install it. You may need to relaunch Weka after installing the package.
- Install from file: You can also download CAAR's plug-in directly from the CAAR project's home page[7] to your local machine. Launch "Package Manager", choose the "File/URL" button, and select the downloaded archive file (it should be titled CAAR.zip) from your local machine. After installing the package you may need to relaunch Weka.

The source code of CAAR can also be found in the plugin's archive[8]. The source is located in the source folder inside the archive. For those who wish to modify the source code or compile it, Weka instructions are provided at How to compile Weka[9].

6.2 Running CAAR in Weka

After installing CAAR and launching Weka, you will see the Weka GUI as shown in Fig. 3. Select the Explorer button and Weka Explorer will be launched as shown in Fig. 4. Select the open file button in Weka Explorer and choose the data set to be tested. You can download a sample Weka data set[10] from CAAR's homepage.

Next choose the Classify menu in Weka Explorer. Weka will display the available classifiers, as shown in Fig. 7. CAAR will be listed in the "lazy" classifiers category.

Choose CAAR as the classifier. The parameters used by CAAR can be tuned by clicking on the text box next to the Choose button as shown in Fig 8.

[7] http://sourceforge.net/projects/caar/files/CAAR.zip/download
[8] http://sourceforge.net/projects/caar/files/CAAR.zip/download
[9] http://weka.wikispaces.com/How+do+I+compile+WEKA%3F
[10] http://sourceforge.net/projects/caar/files/mpg.arff/download

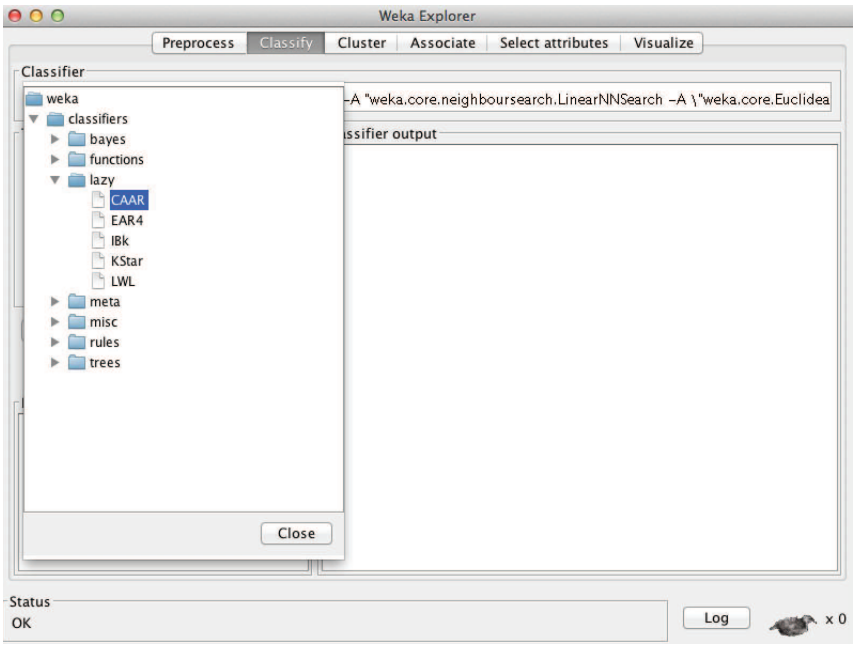

Figure 7: Weka Classifiers.

Fig 8 shows the parameters that can be set for CAAR and sample values for them:

- kNN specifies the number k of nearest neighbors for k-NN to use in predicting the solution.
- c, l and r respectively denote the number of nearest cases to define context for a case, the number of nearest neighbors to generate adaptation rules, and the number of adaptations to apply per base case.
- $nearestNeighbourSearchAlgorithm$ enables selection of the similarity measures used for retrieving the source cases and $ruleNearestNeighbourSearchAlgorithm$ is used to retrieve the adaptations rules to apply.

Figure 8 shows sample values for CAAR parameters. In this case, CAAR estimates the input query's value by combining the adjusted values of the seven nearest neighbors of the input query. The value of each of those nearest neighbors is adjusted by applying seven adaptation rules. The adaptation rules are built from the nearest 30 cases of each case in the case base, and the context of a case is determined by training a linear regression model based on its 70 nearest cases.

Note that values of CAAR's parameters can be tuned by hill climbing on the training data using cross validation, as has been done in experimental evaluations of EAR4. However, this process is not currently implemented in the Weka version of CAAR, so it is up to the user to determine parameter tunings for a particular data set.

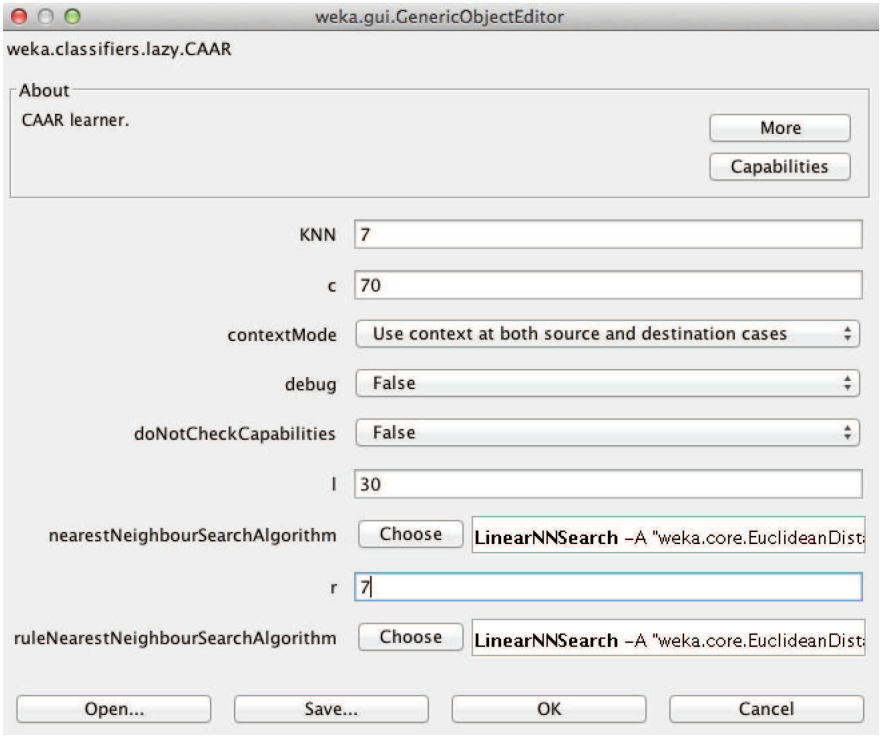

Figure 8: Setting EAR4's parameters.

References

[1] Jalali, V., Leake, D.: Extending case adaptation with automatically-generated ensembles of adaptation rules. In: Case-Based Reasoning Research and Development, ICCBR 2013, Berlin, Springer (2013) 188–202

[2] Jalali, V., Leake, D.: A context-aware approach to selecting adaptations for case-based reasoning. In: Modeling and Using Context. Springer, Berlin (2013) 101–114

[3] Witten, I., Frank, E., Hall, M.: Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. Third edn. Morgan Kaufmann, San Francisco (2011)

[4] Aamodt, A., Plaza, E.: Case-based reasoning: Foundational issues, methodological variations, and system approaches. AI Communications **7**(1) (1994) 39–52 http://www.iiia.csic.es/People/enric/AICom.pdf.

[5] Kolodner, J., Leake, D.: A tutorial introduction to case-based reasoning. In Leake, D., ed.: Case-Based Reasoning: Experiences, Lessons, and Future Directions. AAAI Press, Menlo Park, CA (1996) 31–65

[6] Leake, D.: CBR in context: The present and future. In Leake, D., ed.: Case-Based Reasoning: Experiences, Lessons, and Future Directions. AAAI

Press, Menlo Park, CA (1996) 3–30
http://www.cs.indiana.edu/~leake/papers/a-96-01.html.

[7] López de Mántaras, R., McSherry, D., Bridge, D., Leake, D., Smyth, B., Craw, S., Faltings, B., Maher, M., Cox, M., Forbus, K., Keane, M., Aamodt, A., Watson, I.: Retrieval, reuse, revision, and retention in CBR. Knowledge Engineering Review **20**(3) (2005)

[8] Aha, D.: Case-based learning algorithms. In: Proceedings of the 1991 DARPA Case-Based Reasoning Workshop. Volume 1. (1991) 147–158

[9] Jalali, V., Leake, D.: Enhancing case-based regression with automatically-generated ensembles of adaptations. Journal of Intelligent Information Systems (2015) In press.

[10] Hanney, K., Keane, M.: The adaptation knowledge bottleneck: How to ease it by learning from cases. In: Proceedings of the Second International Conference on Case-Based Reasoning, Berlin, Springer Verlag (1997) 359–370

[11] Blake, C., Merz, C.: UCI repository of machine learning databases (2000) http://www.ics.uci.edu/∼mlearn/MLRepository.html.