CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	Combining and choosing case base maintenance algorithms
Authors	Cummins, Lisa
Publication date	2013
Original Citation	Cummins, L. 2013. Combining and choosing case base maintenance algorithms. PhD Thesis, University College Cork.
Type of publication	Doctoral thesis
Rights	© 2013, Lisa Cummins. - http://creativecommons.org/licenses/by-nc-nd/3.0/
Download date	2026-11-15 02:39:12
Item downloaded from	https://hdl.handle.net/10468/1154

UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

Combining and Choosing Case Base Maintenance Algorithms

Lisa Cummins



A thesis submitted to the National University of Ireland
in accordance with the requirements for the degree of
Dodoctor of Philosophy in the Faculty of Science.

Research Supervisor: Dr. Derek Bridge
Head of Department: Prof. Barry O'Sullivan

Department of Computer Science
National University of Ireland, Cork

May 2013

Contents

1 Introduction **1**
- 1.1 Case-Based Reasoning . . . 1
- 1.2 Maintenance of CBR Systems . . . 2
- 1.3 Research Objectives . . . 5
- 1.4 Thesis Overview . . . 5
- 1.5 Contributions of this Thesis . . . 7

2 Case Base Maintenance Algorithms **10**
- 2.1 Introduction . . . 10
- 2.2 Case-Based Classification . . . 10
- 2.3 Case Base Maintenance Algorithms . . . 11
  - 2.3.1 Noise Reduction Algorithms . . . 12
  - 2.3.2 Redundancy Reduction Algorithms . . . 13
- 2.4 The Competence Model . . . 15
  - 2.4.1 Coverage, Reachability, Dissimilarity and Liability Sets . . . 15
  - 2.4.2 The Classifies and Misclassifies Functions . . . 16
- 2.5 Noise Reduction Algorithms . . . 17
  - 2.5.1 RENN . . . 17
  - 2.5.2 BBNR . . . 18
  - 2.5.3 Comparison of Noise Reduction Algorithms . . . 19
- 2.6 Redundancy Reduction Algorithms . . . 20
  - 2.6.1 RC . . . 20
  - 2.6.2 ICF . . . 21
  - 2.6.3 CRR . . . 23
  - 2.6.4 Comparison of Redundancy Reduction Algorithms . . . 24
- 2.7 Empirical Evaluation . . . 25
  - 2.7.1 Experiment Objective . . . 26
  - 2.7.2 Test Data . . . 26
  - 2.7.3 Experiment Methodology . . . 27
  - 2.7.4 Results . . . 27
- 2.8 Conclusions . . . 28

3 Maintenance by a Committee of Experts **29**
- 3.1 Introduction . . . 29
- 3.2 Related Work . . . 30
  - 3.2.1 Ensembles . . . 30
  - 3.2.2 Distributed Case-Based Reasoning . . . 31

3.2.3 Case Base Maintenance . . . 32
3.3 The MACE Approach . . . 32
3.4 Evaluation . . . 33
3.4.1 Experiment Objectives . . . 33
3.4.2 Test Data . . . 34
3.4.3 Experiment Methodology . . . 34
3.4.4 Results . . . 34
3.4.5 Noise Reduction Results . . . 36
3.4.6 The Special Case of Spam . . . 38
3.5 Conclusions . . . 39

4 Case Base Measures **41**
4.1 Introduction . . . 41
4.2 Multiple Classes and Symbolic Attributes . . . 42
4.2.1 Multiple Classes . . . 43
4.2.2 Symbolic Attributes . . . 43
4.3 Complexity Measures . . . 44
4.3.1 Measures of Overlap of Attribute Values . . . 44
4.3.2 Measures of Separability of Classes . . . 49
4.3.3 Measures of Geometry, Topology and Density of Manifolds . . . 52
4.4 Criteria for Complexity Measures . . . 54
4.5 Evaluation . . . 56
4.5.1 Experiment Objectives . . . 56
4.5.2 Test Data . . . 56
4.5.3 Experiment Methodology . . . 56
4.5.4 Comparing Measures . . . 56
4.5.5 Predicting Classifier Error . . . 57
4.5.6 Effect of Maintenance Algorithms on Measure Values . . . 62
4.6 Conclusions . . . 70

5 Case Base Maintenance as Meta-Case-Based Classification **71**
5.1 Introduction . . . 71
5.2 Maintenance Algorithm Selection as Meta-Case-Based Classification . . . 72
5.2.1 Meta-Problem Description . . . 72
5.2.2 Meta-Solution . . . 73
5.2.3 Performing Meta-Case-Based Classification . . . 74
5.3 Evaluation . . . 75
5.3.1 Holdout Experiments . . . 75
5.3.2 Leave-One-Out Experiments . . . 77
5.4 Previous Work . . . 78
5.5 Conclusions . . . 79

6 Incremental Meta-Case-Based Classification **81**
6.1 Introduction . . . 81
6.2 Incremental Case Base Maintenance Algorithms . . . 82
6.2.1 Incremental Competence Model . . . 82
6.2.2 RENN$_i$ . . . 83
6.2.3 BBNR$_i$ . . . 83

6.2.4 RCR_i . . . 84
6.2.5 $ICFR_i$ . . . 84
6.2.6 CRR_i . . . 85
6.3 Incremental Computation of Case Base Measures . . . 86
6.3.1 Measures of Overlap of Attribute Values . . . 87
6.3.2 Measures of Separability of Classes . . . 88
6.3.3 Measures of Geometry, Topology and Density of Manifolds . . . 90
6.4 Incremental Meta-Case-Based Classification . . . 91
6.5 Evaluation . . . 91
6.5.1 Experiment Objective . . . 91
6.5.2 Experimental Methodology . . . 91
6.5.3 Results . . . 92
6.6 Conclusions . . . 93

7 Conclusions and Future Work **95**
7.1 Conclusions . . . 95
7.2 Future Work . . . 96

A Correlation Tables **99**

List of Figures

1.1 Aamodt and Plaza's CBR Application Cycle, from [1] 2
1.2 The CBR Application and Maintenance Cycles, from [30] 4
1.3 The CBR Application and Maintenance Cycles, from [75] 5

2.1 RENN Problem Situation 18
2.2 Problem Situation with RENN but not with BBNR 20
2.3 RENN vs BBNR 20
2.4 ICF Problem Situation 22
2.5 Redundancy Algorithms Comparison 24

3.1 The MACE approach, defined by an EBNF grammar 32
3.2 Results for twenty-five different datasets, highlighting the Pareto front . . . 36

4.1 Case base with neat boundary 46
4.2 Problems with the definition of $Overlap(a)$ 47
4.3 Linearly Separable Boundary with a high N_1' value 50
4.4 Mean classifier error against N_1' for twenty-five datasets 60
4.5 Interpretation of graphs in this analysis 63
4.6 Changes in accuracy and complexity: atomic noise reduction algorithms . . 65
4.7 Changes in accuracy and complexity: atomic redundancy reduction algorithms 67
4.8 Changes in accuracy and complexity: composite algorithms 69

5.1 Maintenance algorithm selection as meta-case-based classification 72

List of Tables

2.1 Coverage and Liability Sets for the Case Base shown in Figure 2.3 21
2.2 BBNR Algorithm for the Case Base shown in Figure 2.3 21
2.3 Redundancy Coverage Sets for the Case Base shown in Figure 2.5 25
2.4 Redundancy Reachability Sets for the Case Base shown in Figure 2.5 25
2.5 Details of the datasets used in experiments 26
2.6 Datasets with Missing Values 27
2.7 Results for existing algorithms 27

3.1 Atomic case base maintenance algorithms, and classic composites 30
3.2 Top five algorithms ordered by harmonic mean over twenty-five different datasets 35
3.3 Results for different uses of noise reduction algorithms 37
3.4 Top five algorithms ordered by within-class error rate for the spam datasets 39

4.1 The case base complexity measures that we have considered in this work . . 42
4.2 Classification error 59
4.3 Classifier correlation coefficients 60
4.4 Percentage of cases deleted, change in accuracy, and change in normalised complexity measures: mean (and standard deviation) over twenty-five datasets. (In this analysis, a positive change in a complexity measure implies lower complexity.) 64

5.1 Results of Classic Composites and Meta-CBR using Holdout Validation . . 76
5.2 Results of Classic Composites and Meta-CBR using Leave-One-Out Cross-Validation 78

6.1 Results of Incremental Algorithms using Leave-One-Out Cross-Validation . 93

A.1 Correlation coefficients of measures and classifiers over all twenty-five datasets 99
A.2 Correlation coefficients of measures and classifiers over fourteen Boolean classification datasets 100

Declaration

This dissertation is submitted to University College Cork, in accordance with the requirements for the degree of Doctor of Philosophy in the Faculty of Science. The research and thesis presented in this dissertation are entirely my own work and have not been submitted to any other university or higher education institution, or for any other academic award in this university. Where use has been made of other people's work, it has been fully acknowledged and referenced.

Signature: _____

Date: _____

Acknowledgements

There are many people that I would like to thank for making this work possible. Firstly, I would like to thank my PhD supervisor, Dr. Derek Bridge. His support and guidance have been invaluable throughout this research. In particular, his encouragement and patience during my writing-up period is very much appreciated.

I would like to thank the staff in the Computer Science Department who helped me during my PhD, particularly the systems administration staff who were always willing to organise lab machines for me to run my experiments on. I am also grateful to my colleagues in 2-08, particularly Aidan Waugh and Cathal Hoare, for their interesting research discussions, and, more importantly, for helping to make student life entertaining and enjoyable.

I am grateful to Mehmet Göker and his colleagues in Pricewaterhouse-Coopers in San Jose, California, for hosting me for two summer internships during my PhD. I learned a great deal and met many very talented people during my time there. I would also like to thank Belén Díaz Agudo for hosting me for a summer at the Universidad Complutense de Madrid, and to the GAIA group, particularly Juan Antonio Recio-García and Guillermo Jiménez-Díaz, for making me feel so welcome while I was there.

I would like to acknowledge the financial support of Science Foundation Ireland, whose funding made this research possible.

I would like to thank my wonderful friends, who have helped me through the ups and downs of my PhD and who have always been there to encourage me when I needed it.

Finally, I would like to thank my family, who have always been behind me, providing their unwavering support to make this possible.

Associated Publications

The publications that are related to this thesis are listed below:

- L. Cummins and D. Bridge. Maintenance by a committee of experts: The MACE approach to case-base maintenance. In L. McGinty and D. C. Wilson, editors, *Case-Based Reasoning Research and Development (Proceedings of the 8th International Conference on Case-Based Reasoning)*, LNAI 5650, pages 120–134. Springer, 2009

- L. Cummins and D. Bridge. On dataset complexity for case base maintenance. In A. Ram and N. Wiratunga, editors, *Case-Based Reasoning Research and Development (Proceedings of the 9th International Conference on Case-Based Reasoning)*, LNAI 6880, pages 47–61. Springer, 2011

- L. Cummins and D. Bridge. Choosing a case base maintenance algorithm using a meta-case base. In M. Bramer, M. Petridis, and L. Nolle, editors, *Research and Development in Intelligent Systems XXVIII (Proceedings of AI-2011, The 31st SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence)*, pages 167–180. Springer, 2011

Abstract

Case-Based Reasoning (CBR) uses past experiences to solve new problems. The quality of the past experiences, which are stored as cases in a case base, is a big factor in the performance of a CBR system. The system's competence may be improved by adding problems to the case base after they have been solved and their solutions verified to be correct. However, from time to time, the case base may have to be refined to reduce redundancy and to get rid of any noisy cases that may have been introduced.

Many case base maintenance algorithms have been developed to delete noisy and redundant cases. However, different algorithms work well in different situations and it may be difficult for a knowledge engineer to know which one is the best to use for a particular case base. In this thesis, we investigate ways to combine algorithms to produce better deletion decisions than the decisions made by individual algorithms, and ways to choose which algorithm is best for a given case base at a given time.

We analyse five of the most commonly-used maintenance algorithms in detail and show how the different algorithms perform better on different datasets. This motivates us to develop a new approach: maintenance by a committee of experts (MACE). MACE allows us to combine maintenance algorithms to produce a composite algorithm which exploits the merits of each of the algorithms that it contains. By combining different algorithms in different ways we can also define algorithms that have different trade-offs between accuracy and deletion.

While MACE allows us to define an infinite number of new composite algorithms, we still face the problem of choosing which algorithm to use. To make this choice, we need to be able to identify properties of a case base that are predictive of which maintenance algorithm is best. We examine a number of measures of dataset complexity for this purpose. These provide a numerical way to describe a case base at a given time. We use the numerical description to develop a meta-case-based classification system. This system uses previous experience about which maintenance algorithm was best to use for other case bases to predict which algorithm to use for a new case base.

Finally, we give the knowledge engineer more control over the deletion process by creating incremental versions of the maintenance algorithms. These incremental algorithms suggest one case at a time for deletion rather than a group of cases, which allows the knowledge engineer to decide whether or not each case in turn should be deleted or kept. We also develop incremental versions of the complexity measures, allowing us to create an incremental version of our meta-case-based classification system. Since the case base changes after each deletion, the best algorithm to use may also change. The incremental system allows us to choose which algorithm is the best to use at each point in the deletion process.

Chapter 1

Introduction

1.1 Case-Based Reasoning

Experience is something that we gain as we go through life. We use this experience to help us to deal with situations that occur each day. For example, a doctor uses his experience of previous patients and the treatments that worked for them to treat the symptoms of a new patient. A mechanic uses his experience of previous engine problems and how he fixed them to fix a new engine problem. A chef uses his experience of what recipes worked well in the past when putting together a new dish.

Case-Based Reasoning (CBR) is a branch of Artificial Intelligence which uses past experiences to solve new problems. CBR is used to solve problems in domains where similar problems have similar solutions. It stores past experiences as cases, where each case contains attributes of a problem and the solution to that problem. The cases are collected to form a case base. A solution to a new problem is found by comparing the attributes of the problem to the cases in the case base and finding the case(s) that are most similar to that problem. The solutions of these similar case(s) are adapted to find a solution to the new problem [42, 77].

A CBR system has four knowledge containers [76]. The vocabulary describes problems and solutions in the domain. The similarity measure contains knowledge about how cases are compared to each other. The case base contains the set of previously-solved problems described by the vocabulary. The adaptation knowledge describes how a retrieved solution is adapted to fit a new problem. This combined knowledge is used to complete the CBR process.

Traditionally, the CBR process to solve a new problem, known as the query, is represented as a cycle [1, 42], as shown in Figure 1.1. The cycle has four steps: *Retrieve*, *Reuse*, *Revise* and *Retain*. The steps work as follows:

Retrieve The case base is searched and a measure of similarity is used to retrieve the most similar case or cases to the query. Since similar problems have similar solutions, these cases are the most useful to use to find a solution to the query.

Reuse The retrieved cases are adapted to create a solution to the query. This takes into account any differences between the query and the retrieved cases and adapts the solution so as to deal with these differences. This step provides a suggested solution to the problem.

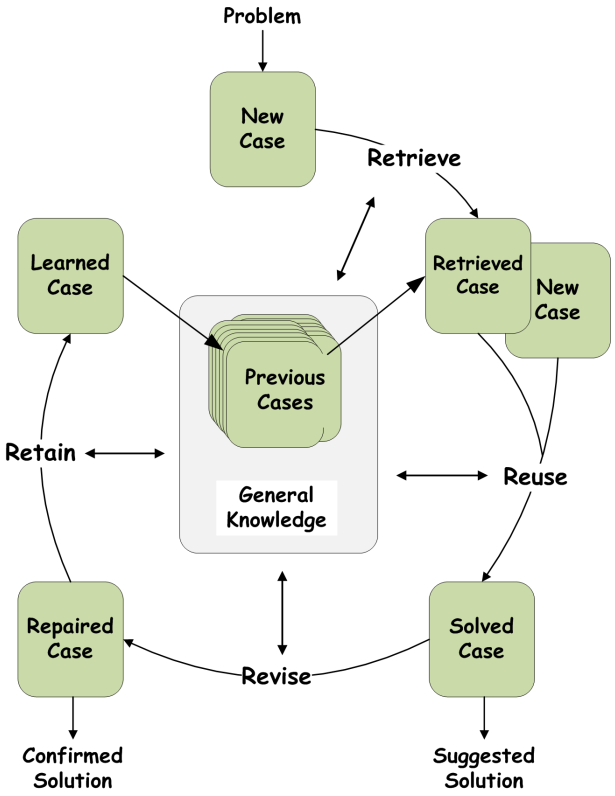


Figure 1.1: Aamodt and Plaza's CBR Application Cycle, from [1]

Revise The suggested solution is tested to see if it successful. In some situations, this may be done by applying it to the problem in the real world; in other situations it may be tried as a simulation. The feedback received from testing the solution may indicate that the solution needs to be revised to work properly for the problem.

Retain Optionally, a new case is created from the description of the query and the revised solution. This new case can be added to the case base so that it can be used in future problem-solving. This allows the CBR system to gain new experience as it solves more problems.

1.2 Maintenance of CBR Systems

CBR systems gain in experience as new cases are added to the case base. These new cases increase the system's coverage of the domain and help it to solve new problems. This is especially useful in a changing domain, as the system will learn the changes as experience is added to the case base. System efficiency may also be improved because less adaptation will be required since the case base will provide better coverage of the problem space [42]. However, as a case base grows in size, a number of problems occur.

Firstly, the cost of retrieval will increase because there are more cases to compare to the problem to find a solution. A problem known as the *utility problem* [26, 53, 59, 83] occurs when the cost of retrieval outweighs the benefits of less case adaptation. Rather

than improving system efficiency, adding new cases to the saturated case base will cause system efficiency to decrease.

Secondly, some cases in the case base will be redundant in the sense that the CBR system could correctly solve the same range of problems without them. These cases add to the cost of retrieval but are not useful. Additionally, changes in the domain may mean that some of the older cases will now be noisy, and these may cause a decrease in the accuracy of the CBR system. This can also happen if cases added to the case base were not checked for correctness before their addition, allowing incorrectly labelled cases to be added to the case base.

To help address these problems, maintenance is performed on the knowledge containers. One way of incorporating maintenance into the CBR cycle was proposed at the CBR workshop at IJCAI'99 [92]. Two additional steps were added to the original four by the workshop participants: *Review* and *Reflect*. The *Review* step is inserted before the *Retain* step and its function is to assess the quality of a case before adding it to the case base. This ensures that only valuable cases are added into the case base. The *Reflect* step looks at areas such as attribute weighting, similarity metrics and case coverage and decides whether or not work needs to be done to improve these.

However, since maintenance can be computationally expensive, it will itself degrade system efficiency if it is performed during every application cycle. To solve this, Göker et al. introduce a separate cycle known as the maintenance cycle [30], which is further developed by Reinartz and his co-authors [75]. This maintenance cycle can be run at regular time intervals, when a particular condition is satisfied (such as case base size reaching a predefined limit) or when the knowledge engineer decides that maintenance is necessary [46].

Göker et al.'s maintenance cycle is shown in Figure 1.2. In this model, the *Retrieve*, *Reuse* and *Revise* application cycle steps are performed as normal every time a new problem arises. If the solution generated by the system is not correct, the end users generate a new solution. The next step is the *Recycle* step, which has two functions. Firstly, it completes the application cycle by putting the generated solution to use. Secondly, if a new solution has been generated by the end user, it stores this case in a case buffer and sends it to the maintenance cycle. The maintenance cycle consists of two steps, *Retain* and *Refine*. These steps work as follows:

Retain The new cases are checked for quality by the CBR administrator. The cases should have a solution that is correct, relevant and applicable to the problem.

Refine The knowledge containers are refined to optimise case base performance. The goal of this step with respect to the case base is to keep it correct, with maximal coverage and no redundant cases. Cases that were checked for quality in the *Retain* step are now checked to see if their inclusion in the case base would cause redundancy or inconsistency. If not, they are added to the case base.

Changes in the domain and the inclusion of these new cases may affect the quality of the other knowledge containers, causing a need for maintenance of these also. This maintenance is also performed in the *Refine* step.

In [75], Reinartz et al. propose another variation of the six-step model, as shown in Figure 1.3. The *Retrieve*, *Reuse* and *Revise* steps are performed as normal with the additional function of collecting data that can be used by the maintenance cycle, such as mean retrieval time and how often a case is rejected or accepted as a solution to a new problem.

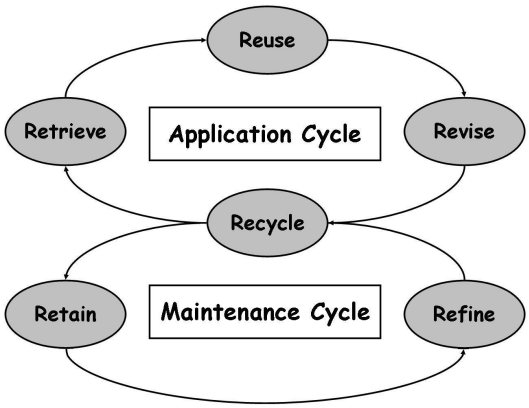


Figure 1.2: The CBR Application and Maintenance Cycles, from [30]

Similarly to Göker et al., they move the original *Retain* step to the maintenance cycle, along with two extra steps: *Review* and *Restore*. The three steps work as follows:

Retain New cases are added to the case base. The indexing structures are also modified to make changes to the similarity measures. The impact of these changes is measured before putting the new similarity measures to use in the application cycle.

Review The knowledge containers are checked (including the new cases submitted for addition to the case base) to see if they meet quality criteria. Reinartz et al. describe in detail these criteria for the case base but not for the vocabulary, similarity or adaptation knowledge containers. The quality criteria for the case base are as follows: correctness, consistency, uniqueness, minimality and incoherency [74]. Follow-on work [80] also includes measures such as case base density, distribution and coverage as measures of case base quality.

Restore Operators are used to change the system to get back to a desired quality level if the *Review* step has indicated that this is necessary. These operators include the addition, removal, specialisation and generalisation of cases, as well as operators to change parts of cases and to merge cases in a number of different ways. The maintenance algorithm employed will decide which of these operators to use to restore system quality.

Maintenance of the CBR system could be done manually by a case base administrator, who can decide which knowledge containers need to be maintained and how to maintain them. However, much research has been done into developing algorithms to automatically analyse CBR systems, recommend maintenance operations and carry them out. These can either be run independently or they can help the administrator to make maintenance decisions.

CBR maintenance involves maintaining each of the different knowledge containers. Much work has been done on maintaining the similarity measure by looking at maintenance of case-indexing, which allows retrieval to be performed more efficiently [11, 24, 25, 33, 61] and of attribute-weighting, which determines the importance of each of the attributes in retrieval [7, 13, 62, 94, 102]. Research has also been done into maintaining the adaptation knowledge [41, 47, 82]. However, most of the maintenance research has been centered on maintaining the case base, known as case base maintenance (CBM). Our work also focuses on this area.

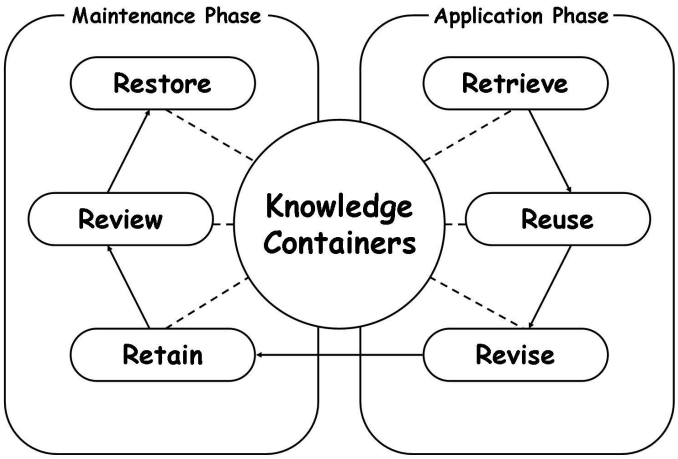


Figure 1.3: The CBR Application and Maintenance Cycles, from [75]

Case base maintenance has mainly focused on structured CBR, where problems are described by a number of attribute-value pairs. This is the area that we work in also. However, some maintenance work has also been done in the areas of textual CBR [34] and conversational CBR [3, 93].

1.3 Research Objectives

Our aim in this work is to help a case base administrator to choose which maintenance algorithm is best to use to delete cases from his case base. We have three main objectives:

1. To show that different case base maintenance algorithms have different biases and that no single algorithm works best for all case bases.

2. To look at how to combine the deletion decisions made by commonly-used case base maintenance algorithms to exploit the strengths of each of the algorithms in different situations.

3. To find ways to choose which algorithm is best for a given case base at a particular point in time.

1.4 Thesis Overview

In Chapter 2, we present a review of the current state of case base maintenance research. From this research, we pick out five case base maintenance algorithms that we believe to be representative and study these in further detail. Two of these are noise reduction algorithms, BBNR [19] and RENN [88], while three reduce redundancy, CRR [19]; ICF [9] and RC [58]. We examine these five algorithms in detail. We reveal each algorithm's biases and present scenarios which show the different deletion decisions made by the different algorithms. These biases indicate to us that different algorithms may work better in different situations. We also examine problems that can occur in a number of the case base maintenance algorithms and we give examples of situations where these problems may present

themselves. As a final step in our examination of these five commonly-used case base maintenance algorithms, we compare their performance on twenty-five classification datasets. We record how much each algorithm deletes and the classification accuracy of the resulting case bases after deletion. The deletion and classification accuracy results for the twenty-five datasets show that different algorithms perform well on different datasets. This evaluation confirms our hypothesis that, because of their different biases, no single algorithm is the best to use for all case bases.

In Chapter 3, we look at ways to combine the deletion decisions of the different case base maintenance algorithms so as to exploit the merits of each of the algorithms. We present our approach to case base maintenance: Maintenance by a Committee of Experts, or MACE. In MACE, we use an ensemble approach to combine the deletion decisions of current maintenance algorithms to benefit from the strengths of each of the algorithms in the combination. We combine the algorithms in two different ways: in sequence or by a committee vote. We present a grammar that shows how these two ways of combining algorithms work. In our evaluation of our MACE approach, we present the issue that case base maintenance is a multi-objective problem, where we want to delete as much of the case base as possible without sacrificing classification accuracy. Often when a maintenance algorithm deletes a large proportion of the case base, the resulting case base will have low classification accuracy due to the large amount of coverage that has been lost. Conversely, when a maintenance algorithm seeks to retain or improve the current classification accuracy, it often does not delete many cases. We examine two ways for case base administrators to be able to take both factors into account when evaluating case base maintenance algorithms: the harmonic mean and the Pareto front. We use these in our evaluation of our MACE algorithms. In evaluating MACE, we show that our novel composite maintenance algorithms perform well when compared with the classic maintenance algorithms. However, we again face the problem that no single algorithm gives the best performance over all case bases.

To address the problem of deciding which maintenance algorithm to use for a particular case base, we need to have a way of capturing the state of that case base at that point in time. In Chapter 4, we examine seventeen case base complexity measures, which provide us with ways of describing the case base numerically. The majority of these are based on a survey of complexity measures by Ho and Basu [37, 38]. Their measures have since been implemented by Orriols-Puig et al. [64] in DCoL, the Data Complexity Library which is an open-source C++ implementation of thirteen measures based on those in the Ho and Basu survey. We use these thirteen measures, and we take four more from the CBR literature: we define two from Massie et al.'s complexity profile measure [55]; we use Fornells et al.'s separability emphasis measure [22]; and finally we define another from Smyth & McKenna's notion of case base competence [85]. We examine each of these measures in detail, revising five of the original Ho and Basu measures to make them more fit for purpose. We determine which of the measures can apply to multi-class classification problems and to case bases with symbolic-valued attributes, both of these being common in CBR, and we adjust the measures to work in both of these situations where possible. We compute these measures for the twenty-five classification datasets and examine how well they correlate with each other. We find a number of situations where the correlation between two measures is surprising: a number of pairs of measures that do not seem alike correlate well while some measures that appear similar do not correlate as well as we would expect. We also look at a number of different classifiers and we evaluate the measures to see which are predictive of classifier accuracy. Finally, we examine how the measure values would be affected by running each

of the case base maintenance algorithms on the datasets. We find that RENN in particular does not perform as expected. While a noise reduction algorithm would be expected to cause a decrease in case base complexity, RENN instead often causes an increase in complexity. We find that BBNR does a better job than RENN at decreasing complexity and maintaining or increasing classification accuracy.

Chapter 5 presents our approach to choosing which maintenance algorithm to use for a given case base. We introduce meta-case-based classification for case base maintenance. This uses past experience of good maintenance decisions to predict what will work in the future. Each of the past case bases is made into a meta-case using the complexity measures as meta-attributes and the best maintenance algorithm for that case base as the meta-solution. To predict the best maintenance algorithm for a new case base, we treat it as a query to the meta-case-based classification system. We compute the measures to provide its description, and we use the previous experience to predict which maintenance algorithm is best to use based on this description. We evaluate our meta-case based classification approach using the twenty-five datasets and show that it performs well in comparison to the original case base maintenance algorithms. We also show how feature selection can be used to pick out which measures are best to use as meta-attributes.

In Chapter 6 we examine a more fine-grained approach to choosing a maintenance algorithm. As cases are removed during maintenance, the state of the case base changes. As a result, the best maintenance algorithm for the case base may also change. We present an approach to deal with this by developing an incremental version of each of the case base maintenance algorithms to allow them to suggest one case for deletion at a time. We also develop incremental versions of the case base complexity measures which adjust the measure values as cases are deleted from the case base. We run an incremental version of our meta-case-based classification system. Given a case base that requires maintenance, we query a meta-case base whose case solutions are incremental CBM algorithms. It chooses an algorithm to run, and we use this algorithm to delete a case from the case base. We update the incremental measure values and repeat the process until the predicted algorithm has no case for deletion. Using this incremental system, we try to make the best maintenance decision at every point in the maintenance process. We present an evaluation of all of the incremental case base maintenance algorithms, including our incremental meta-case-based classification system. From this evaluation, we show that our incremental system is competitive with the other incremental algorithms.

Chapter 7 concludes the thesis. We summarise our findings and identify possible areas for future work that arises from the research.

1.5 Contributions of this Thesis

There has been much research into the area of case base maintenance and there have been many algorithms developed to automatically delete cases. A case base administrator has no way of knowing which maintenance algorithm to use to delete cases from his case base. In this thesis, we are concerned with helping a case base administrator to choose which maintenance algorithm will work well to maintain his case base. The main contributions of this work are as follows:

- We implement five of the most commonly-used case base maintenance algorithms and make them available as open source in the jColibri framework

(http://gaia.fdi.ucm.es/projects/jcolibri/). We examine each of these five algorithms in detail to reveal their biases.

- In our detailed analysis of the maintenance algorithms, we discover a number of problems that could cause the algorithms to give unexpected results on particular case bases. In some situations, RENN will delete all of the cases in a case base. BBNR can also do this, but the circumstances under which it will occur are more limited. We give an example of a situation where the problem occurs for RENN but not for BBNR. Alternatively, in some situations ICF may not delete any cases, even though there are redundant cases present. We give examples of small case bases where each of these problems occur.

- We compare the deletion decisions of the maintenance algorithms. Sometimes BBNR and RENN identify different cases as being noisy. We show a scenario where the algorithms differ in their choice of a noisy case to delete. Similarly, we compare the cases that CRR, ICF and RC identify as being redundant, and we show how their deletion decisions differ on a small case base.

- We evaluate the performance of the five case base maintenance algorithms on twenty-five classification datasets. In this evaluation, we show that no single algorithm works well over all datasets. This motivates the rest of our work on combining and choosing maintenance algorithms so that we can decide which maintenance algorithm is best for a given case base at a particular point in time.

- We present our MACE approach to case base maintenance, which aims to combine the deletion decisions of individual maintenance algorithms to exploit the merits of each of the algorithms. We describe MACE using an EBNF grammar. We define sequences and committees and show how they work to form a MACE algorithm.

- We look at how to numerically describe a case base so that we can make decisions about which maintenance algorithm is best to use for that case base. We examine a number of case base complexity measures for this purpose. We develop ways to adjust the case base complexity measures to be able to deal with multiple classes and symbolic attributes. Both of these are common in CBR but a number of the complexity measures that we examine cannot be used for case bases with one or both of these features.

- We work with a number of complexity measures developed by Ho and Basu [37, 38]. We modify the definitions of five of of their measures to make them more fit for purpose. We also work with four new measures that we have based on ideas in the CBR literature. Additionally, we propose criteria by which complexity measures can be judged. This allows other measures to be evaluated for their usefulness in deciding whether a case base is easy or difficult for a classifier.

- We evaluate the measures to show when they agree and disagree on the complexity of a case base and to examine which measures are predictive of classifier error. Finally, we show the effect that case base maintenance has on the complexity measures.

- We use the complexity measures to give us a numerical description of a case base at a given point in time. Feature selection provides us with a way to choose which complexity measures are best to use to predict a maintenance algorithm.

- Using the numerical case base descriptions, we develop a meta-case-based classification approach to choosing the best maintenance algorithm to use for a case base at a given point in time, based on the experience of good maintenance decisions for other case bases. We present an empirical evaluation of this meta-case-based classification system for CBM, showing that our approach is competitive with the original maintenance algorithms.

- We develop incremental versions of the five main case base maintenance algorithms. In contrast to the maintenance algorithms from which they are derived, which propose the deletion of *batches* of cases, each time one of our incremental algorithms is run it proposes what it regards as the *single* best case to delete next.

- We also develop incremental versions of the complexity measures, which adjust the measure values as cases are deleted from the case base. These incremental measures, along with the incremental algorithms, allow us to develop our incremental meta-case-based classification system for maintenance. This system suggests the best incremental algorithm to use for a case base at a given time, which allows us to always use the best maintenance algorithm, even as the case base changes through the removal of other cases. Finally, we present an evaluation of the incremental algorithms and our incremental meta-CBR system.

Chapter 2

Case Base Maintenance Algorithms

2.1 Introduction

There have been many algorithms developed to automatically remove noise or redundancy (or both) from case bases. In this chapter, we will review these algorithms and examine what they aim to achieve. Since the algorithms work in different ways to achieve noise or redundancy reduction, different algorithms should work better on different case bases. To illustrate this, we look in more detail at five of the maintenance algorithms which we believe to be representative of the work that has been done on case base maintenance. We show examples of times when one algorithm works better than another and situations where one or more of the algorithms will not work as they are designed to. This motivates our later work on ways to combine the existing maintenance algorithms and ways a case base administrator can choose one that works well for his particular case base.

Case-Based Reasoning can be used in many different problem areas such as diagnosis, planning and design. Much of the work on case base maintenance has been done in the area of classification, which is also the area that we will concentrate on. In the next section, we give an introduction to case-based classification and to the notation that we use to describe it both in this chapter and throughout the thesis.

2.2 Case-Based Classification

Classification is the task of deciding which of a predefined, finite set of classes an object belongs to. When we perform case-based classification, we use the cases in the case base to make this decision. We denote the case base by CB and the number of cases that it contains by $|CB|$. Each case in CB contains a problem description and a solution. The problem description is usually made up of a number of attributes.

We use a_i to denote an attribute, with $i \in \{1, ..., n\}$ where n is the total number of attributes. For each attribute a_i, we let case x's value for that attribute be denoted by x_{a_i}. Attributes may be Boolean-valued, numeric-valued or symbolic-valued. We assume that a distance function (or similarity function) can be defined on the domains of each attribute. Where the attribute is numeric-valued, we denote the maximum possible value of attribute a_i by $max(a_i)$, and its minimum possible value by $min(a_i)$. If we do not know the maximum or minimum possible values for a given attribute, we find its maximum and minimum values contained in the case base, and use these instead.

In case-based classification, the solution of each case will be a class label. We denote x's

class by x_c. The set of all classes contained in the domain is denoted by C, and we denote a class label by c_j, where $j \in \{1, ..., |C|\}$. If there are just two possible classes, we refer to the task as *Boolean classification*. In this case, we designate the two classes by c^+ and c^-. If, however, there are more than two classes, we refer to the task as *multi-class* classification.

A case is described using the above notation as $x = \langle x_{a_1}, \ldots, x_{a_n}, x_c \rangle$.

As well as knowing the maximum and minimum values of an attribute, for some of the measures we need to know the maximum and minimum values of that attribute among cases of a particular class. For an individual attribute a, let $min(a, c)$ be the minimum value of a that occurs in cases labelled by c; and similarly for $max(a, c)$.

When faced with a new query, denoted by q, the case-based classifier works to assign a class label to q. To classify q using the case base CB, we first compute the similarity of each case in the case base to q using the global similarity measure shown in Equation 2.1.

$$Sim(x, q) = \frac{\sum_{i \in n} w_{a_i} \times sim_{a_i}(x_{a_i}, q_{a_i})}{\sum_{i \in n} w_{a_i}} \tag{2.1}$$

Each attribute has a weight, w_{ai}, and the global similarity measure uses these weights to compute a weighted average of the local similarity measures, which are applied to each attribute in turn. We have different local similarity measures for symbolic attributes and numeric attributes. Equation 2.2 gives the local similarity measure for symbolic attributes and Equation 2.3 shows the local similarity measure for numeric attributes.

$$sim_{a_i}(x_{a_i}, q_{a_i}) = \begin{cases} 1 & \text{if } x_{a_i} = q_{a_i} \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

$$sim_{a_i}(x_{a_i}, q_{a_i}) = 1 - \left(\frac{|x_{a_i} - q_{a_i}|}{max(a_i) - min(a_i)} \right) \tag{2.3}$$

We perform k-Nearest Neighbour (k-NN) classification by retrieving the k cases that are most similar to the query. In the case of a tie for the top k cases (that is, if the case at position k has the same similarity to q as the case at position $k + 1$), we randomly pick from the tied cases until we have k similar cases. Another possibility is to retrieve all of the tied cases. In this situation, more than k cases will be chosen. Sarah Jane Delany, whose work we will examine later in this chapter, uses this approach. For this reason, our results will not always match her results.

Using the k cases, we take each class c_j in turn and see how many of the k cases have c_j as their class. We take each of these as a vote for c_j and weight the vote by the similarity of that case to q. We sum the votes for c_j to get the total vote for that class. The class with the highest total vote is the predicted class for q.

We refer to the predicted class for q as q_{c_p}. We may also know q's actual class value, for example if we are performing experiments to see how well the classifier works. We denote q's actual class value by q_c. If $q_{c_p} = q_c$, we say that q has been correctly classified. Otherwise, it has been misclassified.

Case-based classification is used in many of the case base maintenance algorithms. We now give an overview of these algorithms and how they work.

2.3 Case Base Maintenance Algorithms

Work on the maintenance of datasets pre-dates research into the area of CBR. Much of the work on the editing of datasets to remove noise and redundancy originated in the instance-

based learning community [29, 35, 79, 88, 89, 97, 98, 99]. However, the techniques used in this area can be applied to modern case bases. We present these instance-based maintenance algorithms along with the algorithms that have since been developed specifically for use in the area of CBR. To avoid confusion, we use CBR terminology throughout our review of the literature.

Case Base Maintenance algorithms are often divided into two types: those that delete noisy (or harmful) cases and those that delete redundant cases. Noise reduction algorithms improve solution quality by removing cases that are considered to have a negative effect on the accuracy of the classification system. They do this by causing misclassification when they are retrieved as nearest neighbours to a query. Redundancy reduction algorithms improve system efficiency by removing cases which do not contribute to the range of problems that the classifier can solve. Many case base maintenance algorithms are composites that use a noise reduction phase to clean up the case base before identifying redundant cases.

2.3.1 Noise Reduction Algorithms

In 1972, Wilson [97] put forward his noise-reduction algorithm, known as the Edited Nearest Neighbour Rule (ENN). He considers a case to be noisy if it has a different class to most of the cases around it. The algorithm retrieves the k nearest neighbours of each case (usually $k = 3$) and removes the case if it does not agree with the majority of these neighbours. By doing this it removes the cases that Wilson considers to be noisy. It also removes instances close to the boundary between different classes since these will be surrounded by cases of a different class. This makes the boundaries between cases smoother, which aims to improve the accuracy of predicted solutions in these areas.

Tomek [88] extended ENN with the Repeated ENN (RENN) and All-kNN algorithms. RENN applies the ENN algorithm repeatedly until no more instances can be removed. All-kNN does the same but increments the value of k after each iteration. These allow for more noisy cases to be deleted by removing cases that ENN may have missed on a single pass.

In their Blame-Based Noise Reduction (BBNR) algorithm, Delany and Cunningham [19] take a different approach to removing noisy cases. Instead of looking for cases which are misclassified, they aim to identify cases which cause other cases to be misclassified. This is because a misclassified case may not be a noisy case but could be misclassified due to its noisy neighbours. BBNR first classifies each case in the case base and keeps track of which cases it classifies correctly and incorrectly. Each of the cases that causes at least one misclassification is then removed if its removal does not cause cases that were previously correctly classified to be misclassified. This allows noisy cases to be deleted without causing any additional misclassification.

Delany [18] aims to improve noise reduction in her RDCL algorithm. This algorithm computes a competence model and identifies the reachability (R), dissimilarity (D), coverage (C) and liability (L) sets of each case (which we will discuss in more detail in Section 2.4.1. These sets then give information about the cases. If a case has a non-empty reachability set, then that case has been correctly classified by the case base, while a non-empty dissimilarity set indicates that the case was misclassified. If a case has a coverage set, then that case is useful for classification. However, if a case has a non-empty liability set, that case is harmful and causes damage.

Delany identifies eight types of case profile depending on the non-emptiness or otherwise of the four sets and she looks empirically at what happens when cases with certain profiles are deleted. She finds that by removing DL cases (which are misclassified and also causing

others to be misclassified) and DCL cases (which do the same as DL but also help in classifying others correctly), she achieves consistent improvements in accuracy.

2.3.2 Redundancy Reduction Algorithms

There are two types of redundancy reduction algorithms: incremental algorithms which begin with an empty case base and add useful cases from the original cases, and decremental algorithms which begin with the original case base and remove redundant cases. We note that we use two meanings of the word 'incremental' in our word. In Chapter 6, we use it to mean that we delete cases one at a time from the case base. However, here it means to build a new case base from the existing one, with residual cases (ones not used to build the new case base) being deleted. In this situation, several cases (a batch) may be deleted at once.

Incremental Redundancy Reduction Algorithms

In 1968, Hart developed his Condensed Nearest Neighbour Rule (CNN) [35] to remove redundancy. CNN begins with an empty edited case base and adds every case from the original case base that cannot be classified correctly by the edited case base. The problem with this is that the order of the cases being presented to CNN has a great influence on the resulting case base. Cases that are presented early have a high chance of being misclassified because the case base contains few cases and so these cases have a high chance of being included in the edited set. Also, CNN is very sensitive to noise, as noisy instances will often be misclassified by their nearest neighbours, and so will be included in the edited case base. This is a problem because it decreases the accuracy of the resulting case base, and it also increases its size as it will need to retain non-noisy cases that lie close to the noisy case.

Ritter et al. introduced an improvement to CNN in their Selective Nearest Neighbour Rule (SNN) [79]. This algorithm extends CNN by requiring that every case in the original case base must be closer to its nearest neighbour in the edited case base than to any case of a different class in the original case base. This guarantees that the edited case base will be minimal and also includes fewer boundary cases than CNN. However, it is quite a complex algorithm to implement, and is still sensitive to noise, as it picks instances closest to the boundary between classes to make up its minimal subset, and noisy instances will often lie close to the boundary.

An approach to dealing with the problem of case ordering is taken by Tomek [89]. He presents cases to the CNN algorithm in order of increasing distance to their nearest neighbour of a different class (nearest unlike neighbour, or NUN). In this way, cases that lie on class boundaries are likely to be considered first, and redundant cases will be presented later when they are less likely to be misclassified and included in the edited case base. This CNN-NUN algorithm reduces case base size more than the original CNN algorithm but is sensitive to noise.

Smyth and McKenna [86] present a number of metrics for ordering cases for presentation to CNN. Their most successful of these, according to experiments in [86], is known as relative coverage (RC). RC is an estimate of how much an individual case contributes to case base coverage. Cases that have high coverage are presented to CNN first, so a compact case base with high coverage will be built by RC-CNN. However, like CNN itself, RC-CNN is still sensitive to noise.

Leake and Wilson [48] present a similar metric RP, or relative performance. This measures what contribution a case makes to the adaptation performance of the system. Cases with high adaptation performance are presented to CNN first as these cases are most useful. Tests show that RP-CNN does have improved performance over RC-CNN. However, it does this at the expense of case base size: RC-CNN produces a smaller case base than RP-CNN.

McKenna and Smyth [58] go on to use their RC ordering metric in their own deletion algorithm. As each case is added to the new case base, the cases that it solves are removed from the old case base. Since cases that solve many other cases are considered first, large groups of cases are removed at once and a new case base is formed quickly because there are fewer cases to consider for inclusion each time. However, this aggressive deletion may mean that the resulting case base loses some of the accuracy of the original case base.

Delany and Cunningham [19] use this same deletion technique but order the cases in the opposite way so that cases that solve few other cases are added to the new case base first. This algorithm is known as Conservative Redundancy Reduction (CRR). They believe that, by using this less aggressive approach, the resulting case base will not lose coverage.

Decremental Redundancy Reduction Algorithms

The first of the decremental redundancy reduction algorithms was the Reduced Nearest Neighbour Rule (RNN), introduced by Gates in 1972 [29]. In this algorithm, an edited case base is created that is initially a copy of the original. Each case is tested for removal in turn. If a case is temporarily taken out of the edited case base, and all of the cases in the original case base can still be correctly classified, then that case is removed permanently from the edited case base. Since the instance being removed does not itself have to be correctly classified, RNN can remove noise as well as redundancy from the set. RNN always produces a case base that is a subset of the case base that CNN produces, but it is computationally more expensive.

Wilson and Martinez [98, 99] introduce a number of redundancy reduction algorithms called DROP1-5. DROP1 is very similar to RNN. It looks at all of the cases which have a case x as one of their nearest neighbours. If these cases can be correctly classified without x then x is removed from the case base. Like RNN, this deletes noisy cases as well as redundant cases at the center of clusters. DROP2 aims to improve DROP1 by ordering cases by their distance from their nearest neighbour of a different class. Cases furthest from a case of another class are considered for removal first so that border points will be retained. However, this means that noisy cases, which will be considered to be border points, will be late in the ordering. Although the noisy case itself may be deleted at this stage, many of its neighbours would have been considered earlier and kept because the noisy case could not be correctly classified. For this reason, DROP3 introduces a RENN noise reduction pass before running the DROP2 algorithm. DROP4 modifies this by making the noise reduction pass less aggressive by introducing the additional rule that a misclassified case is only removed if its removal does not affect the classification of the other cases in the edited set. Finally, DROP5 extends DROP2 by makes an initial pass of the case base in ascending order of distance to the nearest neighbour of a different class to remove noise. DROP2 is then applied until no further changes can be made. The DROP algorithms give a good reduction in case base size while maintaining good accuracy.

In [9], Brighton and Mellish present their Iterative Case Filtering Algorithm (ICF). Like DROP3, ICF runs RENN to filter noise. It then examines each case in the case base in turn. If a case is correctly classified by more cases than it itself correctly classifies, it is

deleted. This removes cases that are not useful.

Smyth and Keane [84] introduce their Footprint Deletion Policy to delete redundant cases. They divide the case base up into different groups depending on how useful they are for classification. All cases from the least useful group are deleted before moving to cases in the next group. The number of cases to be deleted is determined by the user.

Racine and Yang [71, 72] address the problem of deleting redundant and inconsistent cases in mainly unstructured case bases whose cases consist of a problem description and solution description written in natural English. These are hard to deal with because two cases that are semantically the same may be described in very different ways. Racine and Yang use information retrieval techniques to try to identify similarities between two cases. They then either remove one of the cases or merge them to form a new case. They use triggers to identify inconsistent cases. These triggers contain combinations of keywords which do not make sense. If these keywords are found adjacent to each other in a case, or in the same field, the case can be considered to be inconsistent.

Massie and his co-authors [55] use a measure called case complexity to identify redundant cases. Cases with a very low value for their complexity measure are considered to be easy to solve and therefore redundant. To delete cases, a threshold value is decided upon and, in the Complexity Threshold Editing (CTE) algorithm, all cases with complexity values below this threshold are deleted. Like DROP3 and ICF, CTE runs a RENN noise reduction pass first.

Summary of Redundancy Reduction Algorithms

Incremental algorithms have the advantage over decremental algorithms that new cases can be added to the case base easily by just checking if they satisfy the rules of the algorithms. However, they are often sensitive to the order in which cases are presented to the algorithm. Decremental algorithms on the other hand have the advantage that they can suggest a single case for deletion more easily than incremental algorithms.

We have chosen a selection of the algorithms that we believe to be representative to study in further detail. We look at two noise reduction algorithms, BBNR [19] and RENN [88]. These are the two most commonly-used noise reduction algorithms in the literature and have very different approaches to removing noise. We look at three redundancy reduction algorithms, with each providing a different balance between deletion and accuracy: CRR [19] is a conservative algorithm which aims to preserve accuracy, RC-CNN (henceforth, just RC) [58] is an aggressive algorithm which deletes a high number of cases and ICF [9] provides a balance between the two.

Before we present our analysis of these algorithms, we will explain some of the concepts used by the algorithms.

2.4 The Competence Model

2.4.1 Coverage, Reachability, Dissimilarity and Liability Sets

Many of the case base maintenance algorithms use a competence model to decide which cases should be retained and which cases can be deleted to produce the best set of cases to solve new problems. The idea of a competence model was first introduced by Smyth and Keane [84]. They introduced two useful competence properties: the *coverage set* of a case x, which is the set of all cases that x can successfully classify (Equation 2.4), and

the *reachability set* of a case x, which is the set of all cases that can successfully classify x (Equation 2.5). We will talk more about how to define 'classifies' in the next section.

$$CoverageSet(x \in CB) = \{x' \in CB : Classifies(x, x')\} \tag{2.4}$$

$$ReachabilitySet(x \in CB) = \{x' \in CB : Classifies(x', x)\} \tag{2.5}$$

Delany and Cunningham [19] have also introduced the *liability set* of a case for use in their case base maintenance algorithms. The liability set of a case, x, is defined as the set of all cases where x contributes to their misclassification (Equation 2.6). Delany [18] extends the competence model further to include the *dissimilarity set* of a case x, which is the set of cases that misclassify x (Equation 2.7).

The liability set complements the coverage set, while the dissimilarity set complements the reachability set.

$$LiabilitySet(x \in CB) = \{x \in CB : Misclassifies(x, x')\} \tag{2.6}$$

$$DissimilaritySet(x \in CB) = \{x \in CB : Misclassifies(x', x)\} \tag{2.7}$$

In the next section we present the different definitions of the 'classifies' and 'misclassifies' functions.

2.4.2 The Classifies and Misclassifies Functions

Using the terms 'classifies' and 'misclassifies' brings generality to the competence model. The model can be used for any task where it can be defined whether or not a case classifies or misclassifies another. In our work, we are interested in how these terms are defined in the case of classification. We look at how they are defined for each of the maintenance algorithms that we have chosen to study in more depth.

'Classifies' has been defined in a number of different ways in the literature. Therefore, for each algorithm which computes a competence model, we use the classifies function as defined by the authors. Below we show these definitions for $Classifies(x, x')$.

Brighton and Mellish (ICF [9]) A case x' classifies a case x if it is of the same class as x and there are no cases between x and any other case x' of a different class. x is considered to classify itself and so is included in its own coverage and reachability sets.

Delany and Cunningham (BBNR, CRR [19]) A case x' classifies a case x if x is correctly classified by the case base, x' is returned as one of the neighbours of x, and x' has the same classification as x. x is not returned as one of its own nearest neighbours and so is not included in its own coverage and reachability sets.

McKenna and Smyth (RC [58]) 'Classifies' is not described in the presentation of this algorithm in the literature. We use the Delany and Cunningham definition as it seems to be the closest to what is shown for coverage and reachability sets in the Smyth and Keane [84] definitions on which the McKenna and Smyth model is based. However we now include x in its own coverage and reachability sets since it is considered by Smyth and Keane to classify itself.

Algorithm 1 Repeated Edited Nearest Neighbour Algorithm

```
CB ← Case Base
repeat
  changes ← false
  for each x ∈ CB do
    if x does not agree with the majority of its NN then
      flag x for removal
      changes ← true
    end if
  end for
  for each x ∈ CB do
    if x flagged for removal then
      CB ← CB − {x}
    end if
  end for
until not changes
return CB
```

'Misclassifies' is only used by Delany and Cunningham [19] in their BBNR algorithm. They define $Misclassifies(x, x')$ in the following way: a case x' is misclassified by the case base and x is returned as a neighbour of x' and has a different classification to x'. So x contributes to the misclassification of x'.

Now that we have introduced the idea of a competence model, and the different ways that it is implemented for the different maintenance algorithms that we are focusing on, we will present these algorithms in detail. There are a number of important features to look at for each maintenance algorithm which we will now talk about. Firstly, we look at how each of the algorithms works and what types of cases it aims to delete. We also examine any possible variations of the algorithms. Secondly, we try to find situations where the algorithm may not work as it should. Finally we look at each of the redundancy reduction algorithms to see if the noise reduction pass is useful and which noise reduction algorithm is best for each redundancy reduction algorithm. We use $k = 3$ in all of the examples that we present unless otherwise indicated, since this is the common value used for k in CBR literature.

2.5 Noise Reduction Algorithms

2.5.1 RENN

The Basic Algorithm

RENN [88] is a noise reduction algorithm that has been widely used by maintenance algorithms as a preprocessing stage to filter noise from the case base before redundant cases are removed. It considers a case to be noisy if its class does not agree with the class of the majority of its k nearest neighbours. This also means that RENN smooths the boundaries between classes by removing those cases which cross the boundary. The algorithm is described in Algorithm 1.

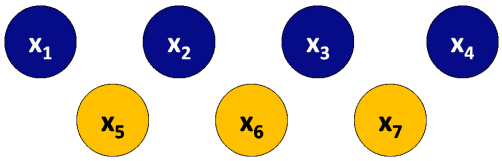


Figure 2.1: RENN Problem Situation

Variations of RENN

The original version of RENN considers a case to be noisy if it does not agree with the majority of its k-NN. However, a number of algorithms which use RENN as their noise reduction algorithm to pre-process the case base consider a case to be noise if it is *misclassified* by its k-NN. Since similarity-weighted voting is often used for classification in CBR, these two definitions may label different cases as being noisy. To maintain consistency with the other maintenance algorithms, we use similarity-weighted voting as our classification method in our experiments.

Problem Situations with RENN

Because RENN removes boundary cases, it can delete all of the cases in the case base if all cases are on the boundary of their class. A simple example of this is shown in Figure 2.1. In this example, the yellow or light grey circles are of one class, and the blue or dark grey circles are of another class.

In both situations, each of the cases has two similar neighbours of a different class and so it will be misclassified. So each case in turn will be flagged for deletion, leaving the case base empty after the first deletion pass is made.

2.5.2 BBNR

The Basic Algorithm

BBNR [19] also looks at cases that have been misclassified. However, instead of labelling these cases as noise, it examines which cases contributed to the misclassification. It then looks to delete these cases that cause other cases to be misclassified.

In this way cases which are correctly labelled but which do not help in the correct classification of other cases are also deleted. An example of this in the spam domain where BBNR has been successfully applied is a case that represents a spam email but looks like a legitimate email. The algorithm is described in Algorithm 2.

Problem Situations with BBNR

BBNR has the same problem situation as RENN but in more limited circumstances. If every case in the case base is misclassified, and every case contributes to the misclassification of some other case (every case is retrieved as a nearest neighbour for a case of a different class) then every case will have a liability set so all will be considered for deletion. Furthermore, since all cases are misclassified, no case will have a coverage set. So each case in turn will be deleted by the algorithm.

Like RENN, in the case base shown in Figure 2.1, all cases in the case base will be deleted by BBNR since the situation just described occurs. However, unlike RENN, the problem

Algorithm 2 Blame-Based Noise Reduction Algorithm

```
CB ← Case Base
/* Build case base competence model */
for each x ∈ CB do
   CSet(x) ← Coverage Set of x
   LSet(x) ← Liability Set of x
end for
/* Remove noisy cases */
CBSorted ← CB sorted in descending order of LSet(x) size
(for ties in LSet(x) size sort in ascending order of CSet(x) size)
x ← first case in CBSorted
while LSet(x) ≠ {} do
   CBSorted ← CBSorted − {x}
   misClassifiedFlag ← false
   for each x′ ∈ CSet(x) do
      if x′ cannot be correctly classified by CBSorted then
         misClassifiedFlag ← true
         break
      end if
   end for
   if misClassifiedFlag = true then
      CBSorted ← CBSorted ∪ {x}
   end if
   x ← next case in CBSorted
end while
return CBSorted
```

does not occur with the case base shown in Figure 2.2. Here, x_4 will not be retrieved as the nearest neighbour to any case and so will not have a liability set. So it will not be considered for deletion and will be left as the only case in the case base after BBNR is run.

2.5.3 Comparison of Noise Reduction Algorithms

Although both algorithms aim to remove noise from the case base, RENN and BBNR delete different types of cases. RENN deletes cases that have been misclassified. It assumes that a case which does not agree with its neighbours is a problem case. BBNR attempts instead to find the case (or cases) which cause the misclassification and delete these cases instead. An example of a small case base where BBNR and RENN delete different cases is shown in 2.3.

In this example, RENN will delete the case with the yellow/light grey class, x_4, since it will be misclassified. Each of the other cases will be classified correctly and so will not be removed.

BBNR computes the competence model shown in Table 2.1. Since x_1, x_2 and x_3 all have a liability set size of one and a coverage set size of two they will be ordered arbitrarily (for simplicity here we will assume that they are ordered numerically but the same analysis applies if they are ordered differently). We show how the algorithm works in Table 2.2.

- x_1 will be removed first since it is the first case in the list. So we attempt to classify

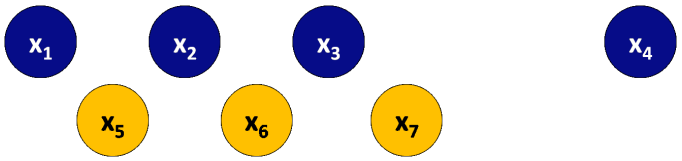


Figure 2.2: Problem Situation with RENN but not with BBNR

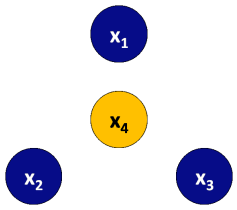


Figure 2.3: RENN vs BBNR

each of the cases in its coverage set: x_2 and x_3. Both x_2 and x_3's neighbours will be x_2, x_3 and x_4, giving both cases a correct blue/dark grey classification. Therefore x_1 will be removed from the case base.

- Next, x_2 will be removed. The cases in its coverage set are x_1 and x_3 and their neighbours will be x_3 and x_4 since these are the only cases left in the case base. x_3 will correctly classify itself, but x_1 is closer to x_4 than to x_3 so it will be incorrectly classified as yellow/light grey by x_4. Therefore x_2 will be reinserted into the case base.
- Finally, x_3 will be removed. The cases in its coverage set are x_1 and x_2 and their neighbours will be x_2 and x_4. x_2 will correctly classify itself, but x_1 is closer to x_4 than to x_2 so it will be incorrectly classified as yellow/light grey by x_4. Therefore x_3 will be reinserted into the case base.

BBNR deletes x_1 from the case base. It does this because x_1 contributes to the incorrect classification of x_4. This is different from the RENN approach, which directly removed the case that looked like it was noise (x_4).

2.6 Redundancy Reduction Algorithms

2.6.1 RC

The Basic Algorithm

McKenna and Smyth [58] looked at Hart's Condensed Nearest Neighbour Rule (CNN) [35], which begins with a new, empty case base and adds cases from the original case base in turn if they cannot be classified correctly by the new case base. However, they remark that, because the order of the cases added has a huge effect on the resulting case base, cases should be ordered intelligently before they are added.

McKenna and Smyth propose a number of different ordering measures which they use to achieve this. However, the one which they have found to be most successful is a metric they

Coverage Sets	Liability Sets
C-Set(x_1) = $\{x_2, x_3\}$	L-Set(x_1) = $\{x_4\}$
C-Set(x_2) = $\{x_1, x_3\}$	L-Set(x_2) = $\{x_4\}$
C-Set(x_3) = $\{x_1, x_2\}$	L-Set(x_3) = $\{x_4\}$

Table 2.1: Coverage and Liability Sets for the Case Base shown in Figure 2.3

x	CB	$x' \in$ C-Set(x)	NN(x')	Misclassified?	Replaced?
x_1	$\{x_2, x_3, x_4\}$	x_2	$\{x_2, x_3, x_4\}$	No	
		x_3	$\{x_2, x_3, x_4\}$	No	No
x_2	$\{x_3, x_4\}$	x_3	$\{x_3, x_4\}$	No	
		x_1	$\{x_3, x_4\}$	Yes	Yes
x_3	$\{x_2, x_4\}$	x_2	$\{x_2, x_4\}$	No	
		x_1	$\{x_2, x_4\}$	Yes	Yes

Table 2.2: BBNR Algorithm for the Case Base shown in Figure 2.3

call relative coverage (RC), an estimate of the competence contribution of an individual case as a function of the case's coverage set (Equation 2.8). The measure weights the contribution of each covered case by the degree to which these cases are themselves covered. This compensates for the fact that some of the cases that a case x covers may also be covered by other cases, and not uniquely by x.

$$RC(x) = \sum_{x' \in CoverageSet(x)} \frac{1}{|ReachabilitySet(x')|} \tag{2.8}$$

Smyth and McKenna then order the cases in descending value of RC, and run CNN to build a new case base. Cases with high relative coverage values will then be presented to CNN first, so a compact case base with high competence will be built by CNN. The cases that they cover will be removed from the original case base to avoid retaining redundant cases for the new case base. The algorithm can be seen in Algorithm 3.

Because it begins with the case that has the highest relative coverage and then deletes all of the cases that it covers before moving on to the next case, RC is an aggressive algorithm. It begins by retaining cases which are surrounded by many cases of the same class since these have high coverage. While this aggressive deletion will greatly reduce the size of the case base, it runs the risk of severely impacting accuracy if cases that are needed to make decisions are deleted.

2.6.2 ICF

The Basic Algorithm

The ICF algorithm [9] deletes a case if more cases solve it than it itself solves. This is calculated by means of the coverage and reachability sets. If a case has a reachability set size greater than its coverage set size then it is deleted. The algorithm makes multiple passes of the case base until there are no more cases with a larger reachability set than their coverage set. The algorithm is shown in Algorithm 4.

Algorithm 3 Relative Coverage Algorithm

$CB \leftarrow$ Case Base
$CBNew \leftarrow \{\}$ (New Case Base)
/* *Remove noisy cases* */
$CB \leftarrow \text{RENN}(CB)$
while $CB \neq \{\}$ **do**
  /* *Build case base competence model* */
  for each $x \in CB$ **do**
    $CSet(x) \leftarrow$ Coverage Set of x
    $RSet(x) \leftarrow$ Reachability Set of x
    $RC(x) \leftarrow \sum_{x' \in CSet(x)} \frac{1}{|RSet(x')|}$
  end for
  /* *Remove redundant cases* */
  $x \leftarrow$ case in CB with highest $RC(x)$ value
  $CBNew \leftarrow CBNew \cup \{x\}$
  $CB \leftarrow CB - CSet(x)$
end while
return $CBNew$

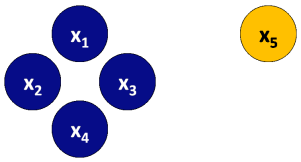


Figure 2.4: ICF Problem Situation

Because ICF removes cases that solve many other cases, it removes internal cases and retains boundary cases. While it makes multiple deletion passes through the case base, it only deletes one case at a time and so is not as aggressive as RC.

Problem Situations with ICF

Under certain circumstances, ICF will not delete any case from a group of redundant cases. This happens when the cases are bounded by a case of a different class, but this case is further from all of the cases of the same class than each of them is from each other. When this happens, they are all contained in each other's coverage and reachability sets. As a result, none of the cases has a reachability set that is bigger than its coverage set, so none of them is deleted. An example can be seen in Figure 2.4.

Here, each of cases x_1, x_2, x_3 and x_4 cover each other. This is because there are no cases of a different class closer to any of the cases than they are to each other. So each of the cases has each of the others, plus the case itself, in both its coverage and reachability set. As a result, all of the cases have a coverage set size of four and a reachability set size of four. So when ICF checks to see if the reachability set size is greater than the coverage set size, none of these cases qualify and so they are not removed even though in this situation it is clear that redundancy exists among the cases with the blue/dark grey class.

Algorithm 4 Iterative Case Filtering Algorithm

```
CB ← Case Base
/* Remove noisy cases */
CB ← RENN(CB)
repeat
  /* Build competence model */
  for each x ∈ CB do
    CSet(x) ← Coverage Set of x
    RSet(x) ← Reachability Set of x
  end for
  /* Remove redundant cases */
  progress ← false
  for each x ∈ CB do
    if |RSet(x)| > |CSet(x)| then
      flag x for removal
      progress ← true
    end if
  end for
  for each x ∈ CB do
    if x flagged for removal then
      CB ← CB − {x}
    end if
  end for
until not progress
return CB
```

2.6.3 CRR

The Basic Algorithm

CRR [19] is a redundancy reduction algorithm which works much like RC but has a different way of ordering the cases for deletion. It looks to retain cases that are near to class boundaries, which it achieves by looking at the size of the coverage set of a case. Cases near class boundaries will have few neighbours of their own class and so will have smaller coverage sets than cases situated far from a class boundary. Cases close to the boundary are added to the edited case base first. As in the RC algorithm, for each case added, the cases that it covers are removed from the training set. This algorithm is shown in Algorithm 5.

Because CRR begins at the case boundaries and adds cases with smallest coverage sets first, it gives a conservative reduction in case base size. If it began with the cases with biggest coverage sets first and then removed those coverage sets each time, it would delete many more cases. Delany and Cunningham hope that by taking the more conservative approach, accuracy will be maintained while still removing redundant cases.

Like the other redundancy reduction algorithms, CRR is run with a noise reduction pass first. It uses BBNR as its noise reduction algorithm. When the two are put together, the composite algorithm is known as Case Base Editing, or CBE.

Algorithm 5 Conservative Redundancy Reduction Algorithm

```
CB ← Case Base
CBNew ← {} (New Case Base)
/* Build case base competence model */
for each x ∈ CB do
   CSet(x) ← Coverage Set of x
end for
/* Remove redundant cases */
CBSorted ← CB sorted in ascending order of CSet(x) size
x ← first case in CBSorted
while CBSorted ≠ {} do
   CBNew ← CBNew ∪ {x}
   CBSorted ← CBSorted − CSet(x)
   x ← next case in CBSorted
end while
return CBNew
```

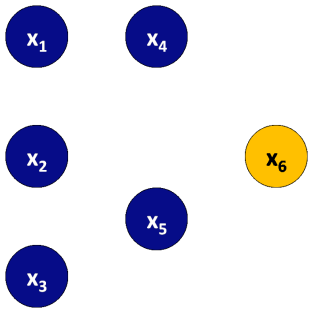


Figure 2.5: Redundancy Algorithms Comparison

2.6.4 Comparison of Redundancy Reduction Algorithms

There are two ways in which the redundancy reduction algorithms differ: the number and type of cases they delete. We have already seen that the RC algorithm is the most aggressive, followed by ICF and then the most conservative, CRR. We will now look at the types of cases that they delete. CRR and ICF both work by retaining cases on the boundaries of classes, while RC retains important cases in the middle of a class group, known as internal cases, and removes the redundant cases around these important cases. We can see an example of the cases that each algorithm deletes using Figure 2.5. We can also see the competence models generated by each of the algorithms in Tables 2.3 and 2.4.

CRR looks for the case with the smallest coverage set. It adds x_6 to its new case base since it has a coverage set size of zero. Next will be the boundary case x_4, since it only covers x_1 and x_5 and all of the other cases cover at least three other cases. So x_4 is added to the new case base and x_1 and x_5 are deleted from the original case base. So now remaining are x_2, which covers cases $\{x_1, x_3, x_4, x_5\}$ and x_3, which covers cases $\{x_1, x_2, x_5\}$. Since x_3 has the smaller coverage set, it is added to the new case base, and the cases that it covers (of which only x_2 remains) are removed from the original case base. So CRR deletes x_1, x_2 and x_5.

ICF looks for cases that have a bigger reachability set than coverage set to delete. These

Case	RC CSet	ICF CSet	CRR CSet
x_1	$\{x_1, x_2, x_3, x_4\}$	$\{x_1, x_2, x_3, x_4\}$	$\{x_2, x_3, x_4\}$
x_2	$\{x_1, x_2, x_3, x_4, x_5\}$	$\{x_1, x_2, x_3, x_4, x_5\}$	$\{x_1, x_3, x_4, x_5\}$
x_3	$\{x_1, x_2, x_3, x_5\}$	$\{x_1, x_2, x_3, x_5\}$	$\{x_1, x_2, x_5\}$
x_4	$\{x_1, x_4, x_5\}$	$\{x_1, x_2, x_3, x_4, x_5\}$	$\{x_1, x_5\}$
x_5	$\{x_2, x_3, x_4, x_5\}$	$\{x_1, x_2, x_3, x_4, x_5\}$	$\{x_2, x_3, x_4\}$
x_6	$\{x_6\}$	$\{x_6\}$	$\{\}$

Table 2.3: Redundancy Coverage Sets for the Case Base shown in Figure 2.5

Case	RC RSet	ICF RSet
x_1	$\{x_1, x_2, x_3, x_4\}$	$\{x_1, x_2, x_3, x_4, x_5\}$
x_2	$\{x_1, x_2, x_3, x_5\}$	$\{x_1, x_2, x_3, x_4, x_5\}$
x_3	$\{x_1, x_2, x_3, x_5\}$	$\{x_1, x_2, x_3, x_4, x_5\}$
x_4	$\{x_1, x_2, x_4, x_5\}$	$\{x_1, x_2, x_4, x_5\}$
x_5	$\{x_2, x_3, x_4, x_5\}$	$\{x_2, x_3, x_4, x_5\}$
x_6	$\{x_6\}$	$\{x_6\}$

Table 2.4: Redundancy Reachability Sets for the Case Base shown in Figure 2.5

cases are x_1 and x_3, both of which have a coverage set size of four and a reachability set size of five. When the algorithm makes its next pass through the data all remaining cases have the same coverage and reachability set sizes so no more cases are deleted. As with CRR, the boundary case x_4 remains.

The RC algorithm computes relative coverage values for each of the cases, and looks for the case with the highest RC value, which is the internal case x_2. So x_2 is added to the new case base, and all cases from x_2's coverage set (x_1, x_2, x_3, x_4 and x_5) are removed from the original case base. The only case left is x_6 so this is added to the new case base on the next loop. So x_1, x_3, x_4 and x_5 are removed by RC, including the two boundary cases x_4 and x_5.

This small example best shows the properties of RC that we have described. It aggressively deletes all but one of the blue/dark grey cases. Additionally, it retains the internal case x_2 and deletes the boundary cases. CRR is more aggressive than expected, deleting three out of the five blue/dark grey cases. It retains x_4, which is on the class boundary, but deletes x_5, which is also on the boundary. ICF, however, retains both of these boundary cases. It is quite conservative in this example, deleting only two cases.

2.7 Empirical Evaluation

In this section, we present an empirical evaluation of the maintenance algorithms. We compare the performance of the composite algorithms, which contain a noise reduction pass followed by a redundancy reduction pass, since both ICF and RC include the RENN noise reduction pass as part of their algorithm description. Since we are using noise reduction for ICF and RC, it is fair that we use it for CRR too, so we run the CBE algorithm in place of CRR.

Dataset Name	Cases	Attributes	Classes	Accuracy (%)
Balance	625	4	3	85.12
Breast Cancer Diagnostic	569	30	2	96.90
Breast Cancer Prognostic	198	33	2	71.58
Breathalyser	127	5	2	71.60
Credit Approval	690	15	2	86.92
Dermatology	366	34	6	97.75
Glass Identification	214	9	7	69.05
Haberman's Survival	306	3	2	69.51
Heart Disease Cleveland	303	14	5	53.22
Hepatitis	155	19	2	80.63
Ionosphere	351	33	2	86.71
Iris	150	4	3	97.00
Lenses	24	4	3	72.50
Liver Disorders	345	6	2	64.20
Lung Cancer	32	56	3	48.00
Pima Indians Diabetes	768	8	2	70.78
Post-Operative Patient	90	8	3	64.71
Spam1	1000	699	2	93.35
Spam2	1000	699	2	94.3
Spam3	1000	699	2	98.25
Spam4	1000	699	2	97.05
Spam5	1000	699	2	94.8
Teaching Assistant Evaluation	151	5	3	55.33
Wine	178	13	3	96.67
Zoo	101	16	7	91.50
Average over twenty-five datasets	-	-	-	80.30

Table 2.5: Details of the datasets used in experiments

2.7.1 Experiment Objective

The objective of this experiment is to compare the performance of the maintenance algorithms over a large number of datasets to examine if one works for all of them or if different algorithms work better in different situations.

2.7.2 Test Data

To evaluate the case base maintenance algorithms, we used twenty-five datasets from the UCI repository [27]. As well as the UCI datasets, we used the Breathalyser dataset described in [21]. Finally, we also used five email datasets for predicting spam as used in [19]. Each email is represented by a vector of Boolean-valued attributes extracted from the structure and text of the original email. The details of all of the datasets are provided in Table 2.5.

In choosing datasets for evaluation we wanted to ensure that we had a good mixture so as to try to reveal any possible weaknesses in the different algorithms. We have datasets of varying sizes, with different numbers of attributes and different numbers of classes. The datasets also have varying amounts of noise and redundancy. For example, the Breathalyser

Dataset Name	Full Size	Missing Values	Resulting Size
Breast Cancer Prognostic	198	4	194
Credit Approval	690	37	653
Dermatology	366	6	358
Heart Disease Cleveland	303	6	297
Hepatitis	155	75	80
Lung Cancer	32	5	27
Post-Operative Patient	90	3	87

Table 2.6: Datasets with Missing Values

	25 datasets		Breathalyser		Credit		Lenses	
Algorithm	Del	Acc	Del	Acc	Del	Acc	Del	Acc
ICF	78.76	73.58	77.53	74.00	84.02	83.38	44.38	52.50
RC	88.75	75.09	87.66	66.80	87.84	86.38	86.25	55.00
CBE	55.31	77.67	61.95	71.20	55.90	83.62	68.13	65.00

Table 2.7: Results for existing algorithms

data is very noisy, while the Balance data is noise-free.

A number of the datasets had missing values in their data. In most cases we are unable to determine the semantics of a missing value. For example, in a product recommender system missing query attributes typically represent "don't care" statements of the customer, while missing case attributes correspond to unknown or non-existing properties of the products. Hence we are unable to choose a similarity measure that respects that semantics without access to domain experts who could give us this information. To deal with this, we removed any case that had any missing values. Table 2.6 shows which datasets had missing values and how many cases were removed from each of these datasets.

2.7.3 Experiment Methodology

For evaluation, we performed repeated holdout on each of the datasets. Each dataset was divided randomly into three splits: a 60% training set, a 20% test set and the final 20% is discarded for these experiments. This final 20% was required for evaluation of systems that we will describe later in Chapter 5. We used the same splits here so that the results would be comparable. We created ten different splits of the data.

For each maintenance algorithm, we used the training set as our case base and ran the algorithm on this training set. We recorded the percentage of cases deleted by the maintenance algorithm. We also recorded the accuracy of the resulting case base by using the test set as queries and recording the percentage correctly classified. Finally, we recorded the accuracy before performing any maintenance to provide a benchmark figure. Table 2.5 contains the benchmark accuracies.

2.7.4 Results

Table 2.7 shows the accuracy and deletion results for the five maintenance algorithms that we have been focusing on for a sample of the datasets.

If we consider the results averaged over the twenty-five datasets, we see that RC is the most aggressive (it deletes 88.75% of cases on average) and CBE is the most conservative (55.31%). With more than a 30% difference in deletion, we would expect that CBE would produce much more accurate case bases than RC. However, RC is only slightly behind CBE in accuracy, with values of 75.09% and 77.67% respectively. We also see an unexpected result when we compare ICF and RC. ICF deletes 78.76, 10% less than RC. We would therefore expect ICF to give a higher accuracy result than RC. Instead, it is slightly less accurate, giving 73.58% accuracy on average. However, since these results are averaged over the twenty-five datasets, they hide details about individual datasets. For example, although RC beats ICF in the average deletion and accuracy results, this is not necessarily the case for each dataset.

If we look at the results from some of the individual datasets, we can pick out places where each of the different algorithms results in the highest accuracy. ICF has the highest accuracy on the Breathalyser dataset (74%), RC on the Credit dataset (86.38%), and CBE on the Lenses dataset (65%). On all three datasets, RC deletes most. ICF deletes more than CBE on two of the datasets but the reverse is true for the Lenses dataset.

We can see from these examples that there is no one maintenance algorithm that performs best for all datasets. Because of their different biases, each of the algorithms works well in some situations, but not in others. This raises the question of whether combining the deletion decisions of the different algorithms can result in an algorithm which produces better deletion results. We will examine this in the next chapter.

2.8 Conclusions

In this chapter, we looked at case base maintenance algorithms. We firstly gave a general overview of the research done into maintenance and the different algorithms that have been developed to reduce noise and redundancy in case bases. We then chose five representative algorithms to study in further detail. We looked at how these five algorithms work and at possible situations where some of them may not work as they should. We also found situations where the different algorithms delete different cases. We then went on to compare the algorithms empirically by running them on twenty-five datasets and looking at how much they deleted and how they changed the classification accuracy of the case base. We saw that, on average, RC deleted the most cases, while CBE resulted in the dataset with the best accuracy. However, when we looked at the results of the individual datasets, we saw that all algorithms had the winning accuracy for at least one dataset.

All of this simply serves to confirm what is well-known: in case base maintenance, there is no clear 'winning' algorithm. The differences in performance of the algorithms are caused by their having different biases. As a result, the question naturally arises whether novel composites, that combine algorithms with different biases, can do better than the original maintenance algorithms. We examine this in the next chapter.

Chapter 3

Maintenance by a Committee of Experts

3.1 Introduction

We have looked at a number of the most widely-used case base maintenance algorithms and the types of cases that they delete. Each of the algorithms has its own advantages and each works well in different situations. To exploit the merits of the different algorithms we examine how we could combine their deletion decisions.

As we saw in the previous chapter, many of the redundancy reduction algorithms are already composed of two algorithms: a noise reduction phase followed by a redundancy reduction phase. In Table 3.1, we show the five algorithms that we have been examining in detail and their component algorithms. We introduce some new terminology to refer to the different parts of each of the algorithms. For example, the ICF algorithm is made up of the RENN noise reduction part and a redundancy reduction phase. We call its redundancy reduction phase ICFR and we use RENN→ICFR to refer to the algorithm as a whole, meaning that RENN is run followed by ICFR.

In Table 3.1, we also introduce a number of terms that we use to describe certain groups of maintenance algorithms. Atomic algorithms are the noise or redundancy reduction phases of the original maintenance algorithms. RENN and BBNR are the atomic noise reduction algorithms, while CRR, ICFR and RCR are the atomic redundancy reduction algorithms. Composite algorithms are made up of more than one algorithm. These usually consist of a noise reduction algorithm followed by a redundancy reduction algorithm. The classic composite algorithms are the composite algorithms that we looked at in detail in the previous chapter: RENN→ICFR (or ICF), RENN→RCR (or RC) and BBNR→CRR (or CBE).

Examining these composite algorithms raises a number of questions. What would happen if we ran the redundancy reduction phase of ICFR, for example, without the noise reduction preceding it? Or how would it perform if we used BBNR to reduce noise instead of RENN before running ICFR? Additionally, is noise reduction followed by redundancy reduction always the best way to run a composite algorithm? What would the result be if we ran ICFR followed by RENN instead of preceding it? To answer questions like these, we looked at the literature for ways to combine components of a system to produce a better system overall.

There are a number of research areas which are concerned with combining systems in

Name used by us	Name in the literature	Description
Atomic noise reduction algorithms		
RENN	RENN	Repeated Edited Nearest Neighbour [88]
BBNR	BBNR	Blame-Based Noise Reduction [19]
Atomic redundancy reduction algorithms		
ICFR	—	Redundancy reduction phase of ICF [9]
RCR	—	Redundancy reduction phase of RC [58]
CRR	CRR	Conservative Redundancy Reduction [19]
Classic composite algorithms		
RENN→ICFR	ICF	Brighton & Mellish's Iterative Case Filtering [9]
RENN→RCR	RC	McKenna & Smyth's algorithm [58]
BBNR→CRR	CBE	Delany & Cunningham's Case Base Editing algorithm [19]

Table 3.1: Atomic case base maintenance algorithms, and classic composites

some way to produce a result which has input from the different systems. Ensembles are used in machine learning to combine the results of different classifiers [20, 70]. They are also used in CBR to produce better results by combining retrieval results [17, 45, 63] or adaptation results [100]. Similarly, distributed CBR [68] deals with the use of multiple case bases and how the system works in combining these.

In this chapter we first examine in more detail the related work in this area. Following this, we present our MACE approach: Maintenance by a Committee of Experts. We combine maintenance algorithms in different ways and we experiment with different combinations and orderings of the component algorithms to see what works best. We then describe our experiments and present our results before concluding the chapter.

3.2 Related Work

3.2.1 Ensembles

The goal of an ensemble is to produce a classifier with the strengths of each of the individual classifiers that it contains so that it has higher accuracy than any of the component classifiers [20, 70]. New problems are classified by each of the classifiers in the ensemble and the predicted class is given by a vote of their classifications. There are a number of different ways of constructing ensembles, with the most common being bagging [8] and boosting [28]. Both approaches manipulate the cases in the case base to generate the different classifiers for the ensemble.

*B*ootstrap *agg*regat*ing*, or bagging, samples with replacement from the case base until it has a set of the same size as the training set. Because the cases are being replaced each time, the sample may not contain all cases, and it may contain some cases more than once. A classifier is constructed from the sample set of cases. This procedure is repeated a predetermined number of times, T, with T classifiers being created. When a new query is being classified, each classifier classifies the query and a majority vote by the classifiers decides the result.

The most common boosting algorithm is known as AdaBoost [28]. This algorithm assigns a weight to each of the cases. The higher a case is weighted, the more influence that

case has on the learned classifier. The weights are adjusted through a series of iterations. It begins with a classifier that it calls a weak classifier. During each iteration, it uses this classifier to classify each of the cases, and adjusts the weights, so that misclassified cases carry a higher weight. As with bagging, a majority vote is taken during classification, but the vote is now weighted by the learned weights.

Ensembles have been shown to be more accurate than their constituent classifiers if two conditions are met: each of the constituent classifiers must have an accuracy of greater than 50% (better than a random choice), and the classifiers contained in the ensemble must be diverse [20].

Ensembles have been used in CBR in a number of ways to produce better classification results. Cunningham & Zenobi [17] identify a shortcoming of a case representation based on a single vector of attribute-value pairs: the set of attributes chosen may not be the optimal subset for that domain. They use ensembles to overcome this by using several individual classifiers based on different attribute subsets. Experiments with four different datasets show that the ensemble classifiers produce better accuracy than a classifier based on the best possible subset of the attributes.

Leake & Sooriamurthi [45] propose that, when multiple external case bases are available to a system, there is a benefit to keeping them separate rather than combining them to form one large case base. When solving a new query, they use some criteria to decide which case base to dispatch the case to. In their experiments, they use the average distance of a query to its nearest neighbours to decide this: if the query is sufficiently far from its nearest neighbours in the main case base, it is dispatched to an external case base. If it is closer to its nearest neighbours in the external case base than it is to its nearest neighbours in the main case base, the external case base is used to solve the case. Otherwise it is solved by the main case base. They also extend this to use cross-case base adaptation, where they adapt any cases solved by the external case base using the original case base for this adaptation. This allows them to combine the results of two different sources to give a better result. In this way they can make use of all information available about the domain.

Plaza & Ontañón [69] use ensembles in multi-agent CBR systems. They use a collection of agents with uncorrelated case bases to improve the accuracy of any individual agent. If an agent has a problem that it cannot solve, it dispatches the problem to the other agents and they solve the problem using their own case bases. These agents then send back their solutions to the original agent, who uses a voting mechanism to decide on the final solution. Plaza & Ontañón present a number of different voting mechanisms that could be used.

3.2.2 Distributed Case-Based Reasoning

Distributed CBR [68] looks at how CBR systems can work with multiple case bases and/or multiple agents. Leake & Sooriamurthi's [45] work described above is an example of a distributed CBR system with multiple case bases. The work by Plaza & Ontañón [69] described above deals with both multiple case bases and multiple agents. Another example of this type of system is McGinty & Smyth's Collaborative Case-Based Reasoning (C-CBR) architecture [57], which consists of a collection of CBR agents which have different problem solving experiences. They use C-CBR in the area of personalized route planning to generate routes that conform to a user's implicit travel preferences. If a user asks for a route that he has no prior experience of, he can use the experience of a similar user who has experience of the route. This collaboration means that all of the users can find routes that they will like, even in unfamiliar territory.

<*System*> ::= <*AtomicAlgorithm*> | <*Sequence*> | <*Committee*>
<*AtomicAlgorithm*> ::= <*AtomicNoiseReductionAlgorithm*> |
<*AtomicRedundancyReductionAlgorithm*>
<*AtomicNoiseReductionAlgorithm*> ::= 'RENN' | 'BBNR'
<*AtomicRedundancyReductionAlgorithm*> ::= 'ICFR' | 'RCR' | 'CRR'
<*Sequence*> ::= <*System*>'→'<*System*>
<*Committee*> ::= '{' <*System*> <*System*>$^{+}$ '}' $\langle$*VotingRule*$\rangle$
<*VotingRule*> ::= 'S' | 'M' | 'U'

Figure 3.1: The MACE approach, defined by an EBNF grammar

3.2.3 Case Base Maintenance

The work in case-based ensembles and in distributed CBR tends to imply the use of multiple case bases, with goals such as improved accuracy, efficiency, or personalisation. Brodley & Friedl use multiple case bases (in fact, multiple folds of the same case base) in their approach to case base maintenance [10]. They split the case base and use a classifier that is trained on one part to classify the remaining part; they repeat this for each of several splits; and they then remove any cases that were misclassified.

This use of multiple case bases, however, is not a feature of our MACE approach: what we are combining are different *algorithms*. Closest to our work is arguably Wiratunga et al [100]. We are using a committee of maintenance experts, whereas they use a committee of adaptation experts. In the next section we will describe this MACE approach.

3.3 The MACE Approach

In the MACE approach, we consider each atomic algorithm to be an expert that recommends cases for deletion. The MACE approach then defines different ways in which these atomic algorithms combine to form novel composite case base maintenance algorithms. The easiest way to show how MACE forms these composites is by giving a grammar. This grammar is shown in EBNF in Figure 3.1.

The grammar's start symbol is <*System*>, which has three expansions:

<*System*> **::=** <*AtomicAlgorithm*> In the simplest case, a maintenance system comprises just one of the atomic algorithms. The atomic algorithms are here divided into the noise reduction algorithms (RENN, BBNR), and the redundancy reduction algorithms (ICFR, RCR, CRR).

<*System*> **::=** <*Sequence*> A composite maintenance system may put systems into sequence, denoted by writing an arrow between them. An example is a classic composite such as BBNR→CRR. When we write this, we mean that the algorithm comprises two phases, where the second (CRR) is executed after the first (BBNR). This rule allows us to create all of the classic composites but novel composites that have not previously been tested too, such as ICFR→BBNR. (The recursion in the grammar also allows the possibility of sequences that comprise more than two algorithms, although this is a degree of freedom that we shall not explore in this work.)

<*System*> **::=** <*Committee*> Another way to create a composite is to form a committee (from which the MACE approach takes it name). A committee comprises a set of two or more systems, which we write between curly braces. Each system within a committee is executed, but cases are not deleted. If a member of a committee would ordinarily delete a case, we treat this instead as a vote for the deletion of that case. Committees must therefore also have a voting rule, which we write in superscript following the closing curly brace, which determines how votes are tallied. We explain the voting rules in the next paragraph. An example of a committee is {BBNR, RENN}^{U} where each of BBNR and RENN separately proposes a set of cases to delete; and the committee deletes each case for which the voting is unanimous (U).

We define three voting rules for committees:

Single (S): If any member of the committee votes to delete case x, then the committee deletes x.

Majority (M): If more than half of the members of the committee vote to delete x, then the committee deletes x.

Unanimous (U): If all of the members of the committee vote to delete x, then the committee deletes x.

Single voting allows us to define committees that can aggressively delete large parts of a case base, especially if the committees' constituents have very different biases. For example, {BBNR, RENN}^{S} deletes all the cases that BBNR identifies as noisy, plus all the cases that RENN identifies as noisy. On the other hand, with unanimous voting, we can define very conservative committees. For example, if {BBNR, RENN}^{U} deletes a case, we can be fairly confident that the case is noisy since both algorithms agree.

The real power of the grammar, however, comes from the mutual recursion in the rules. We will illustrate this with just three examples.

Since a sequence comprises two systems, and a system can be a committee, the grammar allows for sequences of committees. An example is {BBNR, RENN}^{S}→{ICFR, RCR, CRR}^{U}, which first runs an aggressive noise reduction committee, followed by a very conservative redundancy reduction committee.

Similarly, since a committee comprises two or more systems, and a system can be a sequence, the grammar allows for committees of sequences. An example is {RENN→ICFR, RENN→RCR, BBNR→CRR}^{M}, which uses majority voting of the three classic algorithms.

Finally, since a committee comprises two or more systems, and a system can be another committee, the grammar allows for committees with sub-committees. An example is {{RENN, BBNR}^{U}, RCR}^{S}, in which RCR votes alongside a noise sub-committee.

3.4 Evaluation

3.4.1 Experiment Objectives

The aim of this evaluation is to test out our MACE algorithms in a number of different situations. We firstly see how they perform in comparison to the maintenance algorithms that we tested in the previous chapter. Secondly, we look at the area of noise reduction. In this part, we want to examine whether the noise reduction pass that most case base maintenance algorithms do is necessary or useful, or if it would be better to perform redundancy

reduction first and then follow with noise reduction after. Finally, we look at the special case of spam and look at which MACE algorithms perform well on just the spam datasets. The objective here is to see if the algorithms developed specifically for spam datasets (BBNR and CRR) perform better than the others.

3.4.2 Test Data

We used the same datasets for these experiments as we used for our experiments in Chapter 2.

3.4.3 Experiment Methodology

The MACE grammar defines an infinite set of maintenance algorithms (sub-committees of sub-committees; sequences of committees of sequences; and so on). In our experiments, we restrict ourselves to some of the more 'sensible' composites that it defines. We put a number of restrictions on the sequences and committees that we created. We limited the sequences to ones that comprise either two atomic algorithms or one atomic algorithm and one committee, and we ensured that a sequence contained distinct algorithms (e.g. BBNR→BBNR is excluded). We similarly excluded duplicates from committees, and kept the length to five or less. We allowed sub-committees, but not sub-sub-committees, and a committee that contained a sub-committee could only contain one other component (which was allowed to be an atomic algorithm, a sequence or a sub-committee). With all of these restrictions in place, we created 307 algorithms from the grammar for experimentation. We then ran experiments as described in the previous chapter with these 307 algorithms and recorded the accuracy and percentage deletion for each algorithm as previously described.

3.4.4 Results

General Results

In this section, we compare overall performance, averaged over the twenty-five datasets. But there is an immediate problem: case base maintenance is a multi-objective problem. We wish to optimise both the percentage of cases deleted, but also the accuracy of the final case base. This is not possible: algorithms that do well on one of the criteria do not necessarily do well on the other. It comes as no surprise, for example, that our experimental results show that committees with many members and single voting delete many cases, but at a severe cost in accuracy; the opposite is the case with committees with few members and unanimous voting. Case base administrators must strike a balance between accuracy and deletion. They need ways of seeing the trade-offs, so they can make informed decisions.

In some situations it may be possible to weigh up both factors easily, for example in the last chapter where we were only comparing three algorithms to each other. However, if there are a large number of algorithms, it is not possible to compare both factors at once for all of the algorithms. In this case, we need some other way to present the results so as to be able to compare the algorithms more easily. We look at two ways of presenting this.

Harmonic Mean

We could present the arithmetic mean of the percentage of cases deleted and the case base accuracy. But this can be misleading. The arithmetic mean in the case of an algorithm

Algorithm	Deletion (%)	Accuracy (%)	Harmonic mean
{RENN, BBNR, RCR}^{S}	90.06	75.84	82.34
RENN→RCR	88.70	76.05	81.89
{RENN, RCR}^{S}	88.13	75.22	81.17
BBNR→RCR	85.17	77.01	80.88
{{RENN, BBNR}^{U}, RCR}^{S}	83.49	78.43	80.88

Table 3.2: Top five algorithms ordered by harmonic mean over twenty-five different datasets

with very high accuracy and very low deletion will be similar to the arithmetic mean in the case of an algorithm with medium accuracy and deletion. To avoid this, we instead use the harmonic mean (Equation 3.1). The harmonic mean penalises large differences between two values so that a high mean is only produced if both individual values are high. In this way we can find algorithms which have a high value for both accuracy and deletion.

$$HarmonicMean(Acc, Del) = \frac{2 \times Acc \times Del}{Acc + Del} \tag{3.1}$$

Table 3.2 shows the algorithms with the top five harmonic means. We can see that these algorithms have high values for both accuracy and deletion.

In the previous chapter, we showed that the average accuracy over the twenty-five datasets before deletion is 80.3% (Table 2.5). These algorithms perform well when compared to this benchmark accuracy. We can see that the algorithm with the best accuracy, {{RENN, BBNR}^{U}, RCR}^{S}, only causes a 2% drop in accuracy while deleting 83.49%. This seems to be a reasonable compromise.

Additionally, we can see that novel MACE algorithms are performing well. In the top five, there are three committees, all of which are quite conservative. There is also one novel sequence, BBNR→RCR, and one classic, RENN→RCR. Interestingly, all five are some combination of RCR and a noise reduction algorithm.

A problem with this way of presenting the results, however, is that it does not give the case base administrator much sense of what trade-offs can be made. For example, what does he do if he is not happy to see accuracy fall by 2%? In the following section we present a way to view the trade-offs between accuracy and deletion and use this to help in choosing which algorithm to use.

Pareto Front

Another way to handle a multi-objective problem is to compute the Pareto front. The Pareto front contains algorithms that are not dominated by any other algorithms. We take one maintenance algorithm to dominate another if and only if it both deletes more and is more accurate. This finds a set of algorithms which are not bettered by other algorithms and which are all either equal to one another or incomparable with one another (e.g. because one of them deletes more, but the other has higher accuracy).

In Figure 3.2, we plot all 307 of our algorithms. The percentage of cases deleted is on the x-axis, and the percentage accuracy is on the y-axis. Each point represents one algorithm: red squares are algorithms on the Pareto front; blue diamonds are algorithms that are dominated by those in the Pareto front.

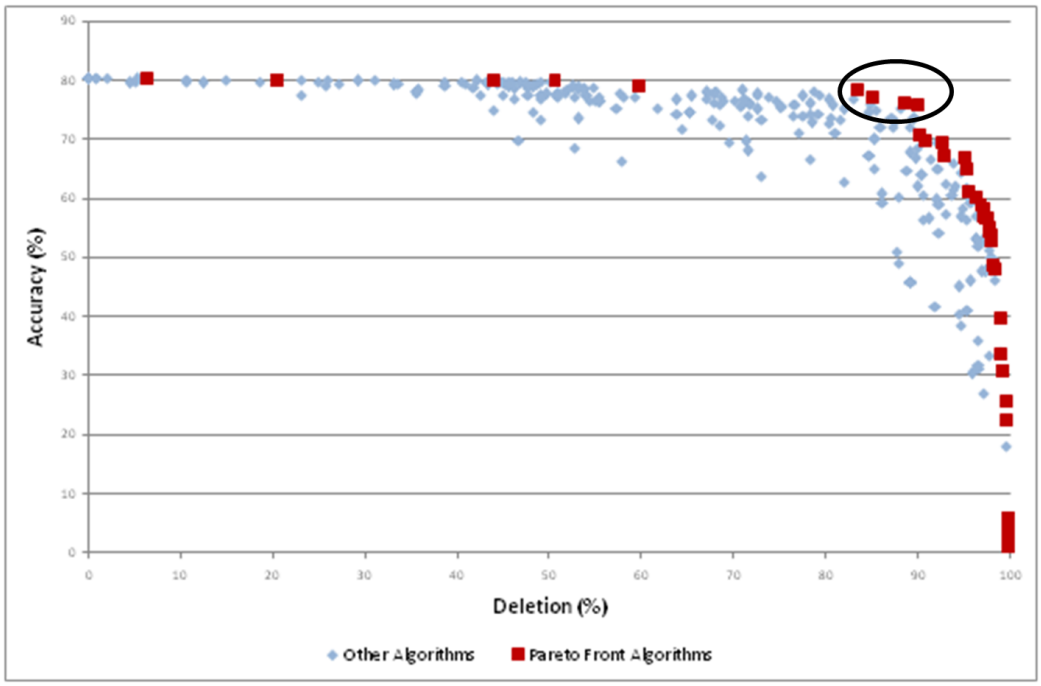


Figure 3.2: Results for twenty-five different datasets, highlighting the Pareto front

The Pareto front in Figure 3.2 contains algorithms that delete everything and hence have extremely low accuracy, and vice versa. However, it also contains algorithms with a good balance between the two. These are very easy to identify on the graph because we can see where accuracy begins to fall rapidly. This is the point where 80% or more of the case base is deleted. We have circled these algorithms on the graph. The four algorithms we have circled are: $\{\{\text{RENN, BBNR}\}^U, \text{RCR}\}^S$, BBNR→RCR, RENN→RCR and $\{\text{RENN, BBNR, RCR}\}^S$. Similarly to the best algorithms ordered by the harmonic mean, we can see that sequences and committees containing RCR and the noise algorithms perform strongly.

The advantage of the graph is that a case base administrator can investigate the compromises that need to be made. For example, if she wants to delete 50% of the case base, the graph shows that this can be done while retaining 78% accuracy; but if she wants to delete 90%, then accuracy drops to 75%. Similarly, if she wants to keep accuracy above 79%, the most that she can delete is about 45%. Of course, to be truly useful, the administrator needs a graph that shows the Pareto front for her particular case base, not an average over twenty-five datasets.

3.4.5 Noise Reduction Results

The idea of using an initial noise reduction phase followed by a redundancy reduction phase probably originates with Wilson et al. [98]. All three classic case base maintenance algorithms follow suit. Here, inspired by the numerous alternatives that the MACE approach defines, we wanted to determine how beneficial a noise reduction phase is. Accordingly, in Table 3.3, we compare all five atomic algorithms, all three classic composite algorithms, algorithms in which atomic redundancy reduction algorithms are paired with noise reduction algorithms that they have never previously been paired with (e.g. BBNR→RCR), and algorithms in which the noise reduction phase comes after the redundancy reduction

	25 datasets		Breathalyser		Lenses	
Algorithm	Del (%)	Acc (%)	Del (%)	Acc (%)	Del (%)	Acc (%)
No Deletion	-	80.30	-	71.60	-	72.50
RENN	23.19	77.60	25.84	68.80	36.25	52.50
BBNR	23.11	80.06	25.97	71.60	41.25	60.00
ICFR	61.81	75.33	61.30	66.00	40.00	67.50
RCR	77.54	77.36	77.40	70.80	63.75	72.50
CRR	35.93	78.29	41.17	72.00	32.50	72.50
RENN→ICFR	78.44	74.40	77.53	74.00	44.38	52.50
RENN→RCR	88.71	76.05	87.66	66.80	86.25	55.00
RENN→CRR	59.39	77.18	66.62	67.20	71.25	55.00
BBNR→ICFR	75.32	75.47	72.08	67.20	70.00	55.00
BBNR→RCR	85.17	77.01	85.19	66.80	80.00	65.00
BBNR→CRR	54.80	78.76	61.95	71.20	68.13	65.00
ICFR→RENN	85.39	64.95	87.79	50.80	79.38	45.00
RCR→RENN	91.52	66.69	92.60	52.00	92.50	45.00
CRR→RENN	63.96	74.34	72.73	62.40	75.63	47.50
ICFR→BBNR	81.63	73.25	80.52	59.20	76.88	57.50
RCR→BBNR	89.74	73.53	87.79	69.20	91.88	45.00
CRR→BBNR	59.83	79.11	64.68	70.00	75.63	62.50

Table 3.3: Results for different uses of noise reduction algorithms

phase, rather than before it (e.g. RCR→RENN). In addition to averages over all twenty-five datasets, we look separately at the Breathalyser dataset, which is known to be very noisy (since it was collected in Dublin pubs!), and the Lenses dataset, which is known to be noise-free. As our earlier table, Table 2.5, shows, without maintenance the accuracy of these datasets is 80.3%, 71.6% and 72.5%, respectively.

Firstly we look at the atomic algorithms and their performance over the datasets. We would expect that noise reduction algorithms would improve classification accuracy on the datasets, especially on the noisy Breathalyser data. However, we see that the atomic *noise* reduction algorithms (RENN and BBNR) never actually improve accuracy, even on the Breathalyser dataset (although BBNR does maintain the same accuracy here while deleting 25.97%). We also see that they both delete a large proportion of the Lenses dataset even though this is noise-free with RENN deleting 36.25% and BBNR deleting 41.5%. They both cause large accuracy drops as a result. The accuracy before deletion is 72.5%, but RENN and BNRR only achieve 52.5% and 60% respectively. The noise reduction algorithms, therefore, do not perform as they are intended to.

Unexpectedly, all three atomic redundancy reduction algorithms have higher accuracy than the noise reduction algorithms on the Lenses dataset. In fact, CRR has a higher accuracy than RENN across the table, and only loses to BBNR on the results averaged over the twenty-five datasets. RCR has an accuracy of 70.8% on the Breathalyser dataset, which is similar to that of both RENN (68.8%) and BBNR (72.6%). However, while maintaining this accuracy, it also deletes over 50% more of the case base. The fact that the redundancy reduction algorithms often work better to preserve accuracy emphasises the fact that the noise reduction algorithms are not always working as they are supposed to.

Secondly, we look at the changes in accuracy and deletion when each of the noise reduction algorithms is run before each of the redundancy reduction algorithms. On the Lenses dataset, there is a large drop in accuracy for each of these algorithms. This is not surprising given the fact that the atomic noise reduction algorithms performed so badly on this dataset. For the Breathalyser dataset, only RENN→ICFR increases accuracy (from 71.6% before deletion to 74%). It does so while deleting over 77% of the dataset, so it performs particularly well here. BBNR→CRR also performs well with only a small drop of 0.4% in accuracy, but the other algorithms of this type experience larger drops in accuracy. In the results averaged over the twenty-five datasets, changes in accuracy are quite small. Using BBNR in the noise reduction phase gives slightly less loss of accuracy than using RENN.

When we look at the deletion results, we can see that these composites do delete more than their constituents on their own, as we would expect. In most situations, deleting more cases lowers accuracy. But there are exceptions where accuracy improves (e.g. RENN→ICFR on the Breathalyser dataset) and where accuracy falls only a little while the case base shrinks a lot (e.g. BBNR→CRR on the Breathalyser dataset). Interestingly, these good results come from 'classic' algorithms (RENN→ICFR, BBNR→CRR). But there are novel combinations that are doing well on the average results, e.g. RENN→CRR, which might be worthy of investigation on other individual datasets.

Finally, we look at the effect of switching around the sequences so that the redundancy reduction algorithm comes before the noise reduction algorithm. The results here are very consistent. In all situations, the resulting algorithm deletes a greater amount than it does in the conventional ordering. On the other hand, the resulting algorithm is always less accurate than its original with three exceptions: CRR→BBNR is more accurate than BBNR→CRR on the averaged data (with accuracy values of 79.11% and 78.76% respectively), RCR→BBNR (69.2%) is more accurate than BBNR→RCR (66.8%) on the Breathalyser data, and ICFR→BBNR (57.5%) is more accurate than BBNR→ICFR (55%) on the Lenses dataset.

In summary, we have discovered a number of situations where our analysis of the performance of the algorithms does not line up with what we would expect from the case base maintenance algorithms. We have shown that the noise reduction phase that is used by so many algorithms is not always useful, and in some cases may be quite detrimental to the accuracy of the algorithm. Additionally, the three classic composite algorithms do not necessarily use the best algorithm for their noise reduction phase, and that the best algorithm to use can change depending on the dataset.

However, we have also shown that if we want to use both noise and redundancy reduction algorithms, it is better for accuracy preservation if we run noise reduction before redundancy reduction. This is in line with what has been proposed in the literature.

3.4.6 The Special Case of Spam

Spam-filtering is a task with special characteristics [19]. Of particular relevance here are the facts that spam is heterogeneous (hence, it is a disjunctive concept), and there is a high cost of false positives. We decided, therefore, to look separately at how the maintenance algorithms perform on our five spam datasets.

As well as recording the percentages of accuracy and deletion for each of the algorithms, we also recorded the rate of false positives, the rate of false negatives and the within-class error rate (the average of the other two rates) [19]. Table 3.4 shows the best five algorithms, ordered by increasing within-class error rate.

Algorithm	Del	Acc	FP Rate	FN Rate	Err Rate
BBNR	5.84	96.16	2.64	5.02	3.82
{{CRR, RCR, ICFR}^{U}, BBNR}^{S}	35.58	95.86	2.42	5.84	4.14
{RCR, BBNR}^{U}	4.54	95.64	2.46	6.26	4.36
{BBNR→CRR, BBNR→ICFR, ICFR→BBNR, CRR→BBNR, BBNR→RCR, RCR→BBNR}^{U}	28.88	95.60	2.44	6.30	4.38
{CRR→BBNR, BBNR→RCR}^{U}	38.14	95.56	2.22	6.66	4.44

Table 3.4: Top five algorithms ordered by within-class error rate for the spam datasets

We can see that the algorithms that do well on the spam datasets are quite different from the algorithms that have done well on other datasets. BBNR performs very strongly, with the lowest rate of error overall. It also provides the noise reduction component of all of the sequences and committees in the top five; RENN is not contained in any of the top five. Since BBNR was developed specifically to remove noise from spam datasets, this result is not surprising. However, it does confirm the strength of the algorithm for this domain.

It is also interesting to note that CRR, the algorithm developed specifically to remove redundancy from spam datasets, does not perform as strongly. It is contained in three of the top five algorithms, but appears less often than RCR. Also, the 'classic' BBNR→CRR composite comes only in 62nd place.

This indicates that, while the BBNR part of the composite algorithm is well suited to spam datasets, CRR is not as well suited. In fact, it appears that no single redundancy reduction algorithm on its own deals well with the spam datasets. The committees in the top five all contain at least two atomic redundancy reduction algorithms, if not all three. This may be due to the fact that spam is heterogeneous; it is more difficult to be sure that spam cases are redundant because they are distributed quite widely across the case base.

3.5 Conclusions

In this chapter, we have examined how to combine case base maintenance algorithms to take advantage of what each of the algorithms does well. We have presented our MACE approach to maintenance, which provides ways to put together these combinations of algorithms in sequences, where the algorithms are run one after the other, and in committees, where each of the algorithms is run and their results are combined by the algorithms voting for which cases to delete.

In our evaluation of the MACE approach, we have identified the fact that case base maintenance is a multi-objective problem, where we want to delete as much as possible, while still keeping accuracy as high as possible. As a result, it is hard to decide which algorithms are better than others since one may delete more than another but cause a bigger accuracy drop. We have presented two ways of dealing with this multi-objective problem: the harmonic mean of accuracy and deletion, and the Pareto front of the algorithms.

In evaluating the algorithms, we have looked at a number of areas. Firstly, we examined the results of our MACE algorithms. We found that some of our newly-developed algorithms are competitive with the classic algorithms. Three out of the top five algorithms (averaged over the twenty-five datasets and ordered by the harmonic mean) were novel

MACE algorithms.

Secondly, we examined the noise reduction pass performed by many of the redundancy reduction algorithms to assess how effective it is. In this evaluation, we looked at the averaged results over the twenty-five datasets, and we also took two single datasets to examine: the Breathalyser dataset, which is noisy, and the Lenses dataset, which is noise-free. We found that running the noise removal algorithms on the datasets did not give us the results that we expected. Neither algorithm improved accuracy on any of the datasets that we examined, although BBNR did maintain the same accuracy on the Breathalyser dataset. We then looked at the results of pairing up each noise reduction algorithm with each redundancy reduction, and we examined the effect of reversing the usual ordering, i.e. running redundancy reduction before noise reduction. In this analysis, we found that a number of sequences worked well, but neither noise algorithm consistently improved the performance of the redundancy reduction algorithms over all of the datasets. We also found that running the noise removal algorithms before the redundancy removal algorithms gave less deletion but higher accuracy results.

Finally, we looked at the area of spam filtering, since this is a task with special characteristics. We were especially interested in how BBNR and CRR performed in this domain since they were developed specifically for it. We found that the algorithms that did well on the spam datasets were quite different to the algorithms that performed well in our earlier analysis. BBNR in particular performed very strongly, confirming its usefulness for this domain. The redundancy removal results, however, were not as consistent. The winning algorithms all contained at least two atomic redundancy reduction algorithms, if not all three. From our analysis, CRR is not as well-suited to spam filtering as BBNR.

In our evaluation, we notice that no single maintenance algorithm stands out as being the best algorithm to use for all case bases. What works well for a noisy case base does not work so well for one that is noise-free, and the algorithms that work best for spam filtering are different algorithms to those that work best for other domains. This brings us to the conclusion that we need to be able to measure different properties of a case base in order to be able to know which algorithm would be good to use. In the next chapter we will examine a number of properties of a case base that may be useful in this way.

Chapter 4

Case Base Measures

4.1 Introduction

In the previous chapter, we learned that the best maintenance algorithm to use for a case base is dependent on characteristics of that case base. In this chapter we look at different case base characteristics which can give information about the complexity of that case base. Although our aim is to use the characteristics to choose the best case base maintenance algorithm, we also look at how these characteristics could be helpful in classification. If we can decide whether a case base is complex or not based on its characteristics, this may be able to give an indication of which is the best classifier to use for that case base. Additionally, we know that classifiers have different biases [60] and cannot work well across all case bases [101, 81]. When testing new classifiers, we should use a selection of datasets with different characteristics to reveal when the new classifier works well, and when it works less well. Often in CBR, researchers experiment on only one or two datasets. Others seek greater rigour by using a lot of datasets, often selected to give a mix: ones with only numeric-valued attributes; ones with only symbolic-valued attributes; ones with both types of attribute; ones with missing values; ones with no missing values; ones with only two class labels; ones with more than two class labels; ones with few attributes; ones with many attributes; ones with few cases; and ones with many cases. But these selection criteria at best have only an indirect influence on classification difficulty. Maciá et al. for example, show empirically that the numbers of attributes and cases do not of themselves correlate with classification accuracy [51]. In this chapter we show that characteristics of the case base such as the complexity of its class boundaries and its coverage of the domain work well to predict the classification accuracy of that case base.

Ho and Basu [37, 38] have looked at a number of different ways of measuring the complexity of a dataset. Their complexity measures have since been implemented by Orriols-Puig et al. [64] in DCoL, the Data Complexity Library which is an open-source C++ implementation of thirteen measures based on those in the Ho and Basu survey.[1] We apply these measures to CBR to compute the complexity of a case base. Additionally, we introduce a number of measures that have been specifically developed within CBR: we define two from Massie et al.'s complexity profile measure [55]; we use Fornells et al.'s separability emphasis measure [22]; and we define another from Smyth and McKenna's notion of case base competence [85].

Table 4.1 summarizes these seventeen measures. We have made changes to a number

[1] `http://dcol.sourceforge.net/`

Table 4.1: The case base complexity measures that we have considered in this work

Measure	Description	Multi-Class	Symbolic
F_1	Maximum Fisher's Discriminant Ratio	*Not DCoL*	No
F_2'	Volume of Overlap Region	Yes	Yes
F_3'	Maximum Attribute Efficiency	Yes	Yes
F_4'	Collective Attribute Efficiency	Yes	Yes
N_1'	Fraction of Cases on a Boundary	Yes	*sim*
N_2	Ratio of Average Intra/Inter Class Distance	Yes	*sim*
N_3	Error Rate of a 1-NN Classifier	Yes	*sim*
L_1	Minimised Sum of Error Distance of a Linear Classifier	*Not DCoL*	*Not DCoL*
L_2	Training Error of a Linear Classifier	*Not DCoL*	*Not DCoL*
C_1	Complexity Profile	Yes	Yes
C_2	Similarity-Weighted Complexity Profile	Yes	Yes
N_5'	Separability Emphasis Measure	Yes	*sim*
L_3	Nonlinearity of a Linear Classifier	*Not DCoL*	No
N_4	Nonlinearity of a 1-NN Classifier	Yes	No
T_1'	Fraction of Maximum Covering Spheres	Yes	Yes
T_2	Number of Cases per Attribute	Yes	Yes
T_3	Dataset Competence	Yes	Yes

of Ho and Basu's measures to make them more fit for purpose. We use a prime symbol to show the measures that we have changed, e.g. Ho and Basu's F_2 measure becomes our F_2' measure. In Sections 4.3.1, 4.3.2 and 4.3.3, we present the measures in more detail and explain any changes that we have made.

The table divides the measures into three groups. These groups are explained in Section 4.3. Additionally, we have two columns in the table which show whether or not the measures can deal with multi-class case bases and symbolic attributes. We discuss these two issues in detail in Section 4.2.

Another area of work that looks at measuring an aspect of case bases is checking whether or not similar problems have similar solutions [43, 56]. This is the basic assumption used in CBR, so the more that it holds true in a domain, then the more suitable CBR is for that domain. Lamontagne [43] introduces a case cohesion measure for textual case bases, which measures to what extent cases that are close to a case x have similar solutions to x. Massie et al. [56] go one step further by identifying cases which do not satisfy the CBR assumption as noise. While these measures are good indicators of performance for CBR systems, they cannot determine whether or not a case base will be considered to be complex for a different type of classifier. We want to be able to check the quality of a case base independent of the reasoner used. As a result we do not include these measures in our analysis.

4.2 Multiple Classes and Symbolic Attributes

Ho and Basu's original measures were designed to work only with Boolean classification and numeric attributes. This is too limiting for the kinds of domains in which CBR is used, where multi-class classification and symbolic-valued attributes are common. Oriolls

attempts to address both of these issues using the suggestions by Ho et al. [39]. In our discussion of the measures, we examine how well each of the measures deals with multi-class case bases and with symbolic attributes. We also discuss any ways in which we have adapted the measures to deal with either or both of these. In Sections 4.2.1 and 4.2.2, we describe general approaches to dealing with both issues which can be used as a fall-back when measure-specific extensions are not available. Following this we look at each of the complexity measures in detail.

4.2.1 Multiple Classes

Many of the measures have been developed to deal only with case bases with two classes. We have extended these to work with multi-class case bases. One-versus-all (OVA) [78] is a general approach that we use for many of these measures when there is not an obvious way to include multiple classes in the measure computation. For $|C|$ classes, the problem is decomposed into $|C|$ Boolean classification problems, where each of these Boolean classification problems discriminates one of the classes from the other $|C| - 1$ classes treated as a single class.

To compute a measure using OVA, we take each class c in turn. We convert the case base into a two-class case base by isolating all cases with class c, and treating the rest of the cases in the case base as having a second class, c'. Since the case base now has two classes, we can compute the original measure on it. We do this for each of the classes in turn and then take an average of the measure values to obtain a measure value for the multi-class case base. This algorithm is shown in Algorithm 6.

Algorithm 6 OVA Computation Algorithm

```
CB ← Case Base
C ← Classes in Case Base
totalMeasureVal ← 0
for each c ∈ C do
  CBEdited ← CB
  for each x ∈ CBEdited do
    if x.c ≠ c then
      x.c ← c'
    end if
  end for
  totalMeasureVal ← totalMeasureVal + computeMeasure(CBEdited)
end for
averageMeasureVal ← totalMeasureVal / |C|
return averageMeasureVal
```

4.2.2 Symbolic Attributes

Ho and Basu do not deal with symbolic attributes. In DCoL, for many of the measures, each of the symbolic values is assigned an integer and the measures are then applied in the same way as for the numeric attributes. This is fine for a Boolean-valued attribute, and it is legitimate in the case where the symbolic values have some ordering and where the integers reflect this ordering. But when these conditions do not hold, it is clearly not

correct. Consider three cases with one symbolic-valued attribute each: $\langle v_1, c^+ \rangle$, $\langle v_2, c^+ \rangle$ and $\langle v_3, c^- \rangle$. It is clear that the attribute is highly discriminatory: values of v_1 or v_2 predict class c^+; a value of v_3 predicts class c^-. If we assign $v_1 \mapsto 1$, $v_2 \mapsto 2$ and $v_3 \mapsto 3$, then the Fisher's discriminant ratio of the attribute is 36 (see Section 4.3.1). But if the mapping of integers to nominal values is $v_1 \mapsto 1$, $v_2 \mapsto 3$ and $v_3 \mapsto 2$, which increases the mean and variance of the attribute values that occur in class c^+, then the ratio is lower (it is 1.0), implying that the case base is harder. Of course, the complexity of the original case base has not changed, only the coding. Therefore, we do not use this approach on symbolic-valued attributes, unless the values have a natural ordering to them. When there is not a natural ordering to the values, and the complexity measure requires to compute similarity, then we use Equation 2.2, which gives us a way of dealing with symbolic attributes for some of the measures. However, we are still left with a number of the measures that we simply cannot compute on certain case bases.

We use these general ways to deal with multi-class case bases and symbolic attributes for any of the measures where there is not a better way to deal with these issues. We now go on to present the measures in detail in the next section.

4.3 Complexity Measures

Each of the complexity measures focuses on some particular aspect of the case base. Ho and Basu [37] divide the measures into three categories depending on what they focus on:

Measures of Overlap of Attribute Values. These measures look at how effective the attributes are for discriminating between the classes, both singly and compositely over a number of attributes. They examine the range of attribute values within the classes and the overlap of values between different classes.

Measures of Separability of Classes. These measures look at how separable the classes are by examining the shape of the class boundary and the distances between the cases that belong to the different classes. They look at each case as a whole rather than at the attributes separately.

Measures of Geometry, Topology and Density of Manifolds. These measures assume that a class is made up of one or more manifolds that define the shape of the class. They look at the position, shape and overlap of these manifolds to estimate how well the classes are separated, and how well the domain is covered by the cases contained in the case base.

We use these categories to group the measures. We now present each of the measures in detail. We look at how each measure deals with multi-class case bases and symbolic attributes. Additionally, we look at problem situations that occur with some of the measures and our solutions to these problems.

4.3.1 Measures of Overlap of Attribute Values

F_1: Maximum Fisher's Discriminant Ratio

For a Boolean classification problem, Fisher's Discriminant Ratio for a given attribute a, $f(a)$, is an indication of the extent to which the values of the attribute in cases labelled by one of the classes overlap with the values of the attribute in cases labelled with the other

class. Equation 4.1 shows how $f(a)$ is computed for attribute a and classes c^+ and c^-, where μ_{a,c^+} and σ_{a,c^+} denote the mean and variance respectively of the values of attribute a in CB that occur in cases labelled by class c^+, and μ_{a,c^-} and σ_{a,c^-} are defined analogously.

$$f(a) = \frac{(\mu_{a,c^+} - \mu_{a,c^-})^2}{{\sigma_{a,c^+}}^2 + {\sigma_{a,c^-}}^2} \tag{4.1}$$

If the two means are similar and the variances are high, for example, it is likely that there is a lot of overlap between those values of a that appear in cases labelled by c^+ and those labelled by c^-; hence a does not discriminate well. On the other hand, if the means are far apart and the variances are low, for example, then it is likely that overlap in values is lower; and the attribute can discriminate between the two classes. Attributes that discriminate best have high discriminant ratios. In DCoL, the implementation uses a vectorial form defined in [52] instead of the above equation.

To measure the complexity of a case base, the maximum Fisher's Discriminant Ratio over the attributes is used, as shown in Equation 4.2. This gives the discriminant ratio of the attribute that discriminates best. High values here mean that the case base is regarded as having low complexity (because there is an attribute that discriminates well); low values mean that the case base has high complexity (because no attribute discriminates well).

$$F_1 = max_{i=1...n} f(a_i) \tag{4.2}$$

Multiple Classes Although there are several ways of extending the measure to multi-class classification, including one described in [52], DCoL provides an implementation only for Boolean classification. To deal with multi-class case bases, we use OVA to compute $f(a)$ for each attribute a. We then take the maximum of these values as the case base complexity, as in Equation 4.2. We note, however, that there is a risk that the averaging will suppress large differences in the discriminatory power of an attribute for different classes.

Symbolic Attributes Since the mean and variance are numeric calculations, there is no easy way to extend F_1 to the case of symbolic-valued attributes.

F_2': Volume of Overlap Region

This measure, which is again defined for Boolean classification and numeric-valued attributes, looks at the overlap of the attribute values of the cases of each class. If there is a large overlap, then the case base is considered to be more complex than a case base with a small overlap, as it is easier to distinguish between the classes.

For each attribute a, the overlap $Overlap(a)$ is the amount by which the values in the range $min(a, c^+)$ to $max(a, c^+)$ overlap with values in the range $min(a, c^-)$ to $max(a, c^-)$, as shown in Equation 4.3. The range $Range(a)$, which will be used to normalize the overlap, is the difference between the maximum and minimum values of a in the case base, as shown in Equation 4.4.

$$Overlap(a) = \min(max(a, c^+), max(a, c^-)) - \max(min(a, c^+), min(a, c^-)) \tag{4.3}$$

$$Range(a) = \max(max(a, c^+), max(a, c^-)) - \min(min(a, c^+), min(a, c^-)) \tag{4.4}$$

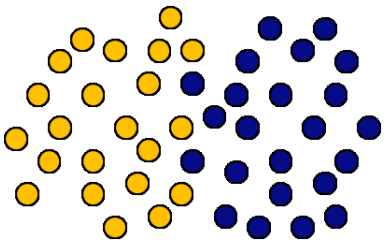


Figure 4.1: Case base with neat boundary

To compute a complexity measure for the case base as a whole, Ho and Basu use the product of the normalised overlaps, as shown in Equation 4.5.

$$F_2 = \prod_{i=1}^{n} \frac{Overlap(a_i)}{Range(a_i)} \tag{4.5}$$

Problems with this Measure

Negative Overlap. In the description of this measure, Ho and Basu say that it will give a value of zero if at least one attribute exists for which the value ranges of the two classes have no overlap. However, this is not the case.

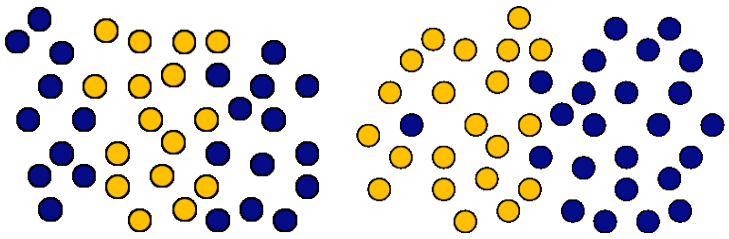
Consider the situation where for c^+ attribute a's values range from 10 to 20, i.e. $min(a, c^+) = 10$ and $max(a, c^+) = 20$; and its values for c^- range from 20 to 30, i.e. $min(a, c^-) = 20$ and $max(a, c^-) = 30$. In this situation, $o(a) = 0$, hence $F_2 = 0$. Yet there clearly is an overlap here; there is a value of this attribute which does not discriminate, namely 20.

This could easily be fixed, but the problem goes deeper. Suppose instead that $min(a, c^+) = 10$ and $max(a, c^+) = 20$ but $min(a, c^-) = 30$ and $max(a, c^-) = 40$. Now there is no overlap and a is most definitely discriminatory. Yet now $o(a) = -10$, whose effects on F_2 will depend on whether there are other negative numbers in the product. DCoL tries partly to deal with negative values of F_2 by taking F_2's absolute value. However, this compounds the problem. If the value of F_2 is large, we cannot tell whether this is because there is no discriminating attribute or because there is an attribute that is very discriminating in this way and so has a large negative value for its overlap.

One way to fix this problem is to set the overlap value for an attribute to be 0 if there is no overlap. Because the attribute overlap values are multiplied together when computing F_2, this will give the overall measure a value of 0 if any attribute discriminates the classes. However, the problems with this complexity measure do not stop there.

Overlap Calculation. The overlap is taken to be the difference between the lowest of the maximum attribute values and the higher of the minimum attribute values. This is assuming that, in an attribute that is highly discriminatory, all the values of a that are associated with c^+ are contiguous, and those for c^-, while they may overlap by a small amount with those for c^+, are also contiguous, as shown in Figure 4.1.

However, consider Figure 4.2(a). In this example, a is also highly discriminatory: medium values predict class yellow or light grey; other values predict blue or dark grey. But, because the maximum and minimum values of a for class yellow (light grey) fall inside



(a) One class split up into different regions of the case base

(b) Noisy case in the middle of cases of a different class

Figure 4.2: Problems with the definition of $Overlap(a)$

the maximum and minimum values for class blue (dark grey), $Overlap(a)$ will incorrectly be high.

Noisy cases may also cause a problem for the overlap calculation. Consider the noisy blue (dark grey) case in Figure 4.2(b). This case extends the minimum value of a for the blue (dark grey) class, causing $Overlap(a)$ to have a much higher value than the actual overlap between the classes.

This measure may also not be reliable for other perfectly common but not necessarily complex case bases, for example case bases for disjunctive concepts.

Because of the problems with this measure, we have chosen to replace it with one of our own. $Overlap(a)$ is redefined as $Overlap'(a)$ which simply enumerates values of a which appear both in cases labelled by c^+ and in cases labelled by c^-. We present this in Equation 4.6. We normalize by the number of distinct values that a has in the case base, as shown in Equation 4.7. We then redefine F_2 as F_2' in Equation 4.8.

$$Overlap'(a) = \{v : \exists x \in CB, x' \in CB, x_a = v \wedge x'_a = v \wedge x_c \neq x'_c\} \tag{4.6}$$

$$Range'(a) = \{v : \exists x \in CB, x_a = v\} \tag{4.7}$$

$$F_2' = \prod_{i=1}^{n} \frac{|Overlap'(a_i)|}{|Range'(a_i)|} \tag{4.8}$$

This equation works with discrete values. This means that it may not work very well for real-valued attributes whose values may be very precise. For these, it could be improved by rounding the values to a certain number of decimal places agreed with the knowledge engineer. For example, for an attribute that has values where a change of 1 is considered small, the knowledge engineer may decide that values can be rounded to one decimal place and overlaps computed on these rounded values.

Multiple Classes DCoL extends F_2 to work for multiple classes by looking at each possible pair of classes and computing the measure for the cases of those two classes. Their measure value for the case base as a whole is the sum of the absolute value of these class measures. This causes two problems. We have already discussed the problem of using absolute value to remove the negative values and how we have dealt with this issue. The other problem is that by summing the individual values, the maximum measure value for the case base is now greater than 1. This makes it hard to compare measure values for different case bases.

Our new measure, F_2', has the advantage that it works unchanged for Boolean and multi-class classification.

Symbolic Attributes When computing F_2, DCoL assigns integer values to the symbolic attribute values. We have already described why this is almost always incorrect in Section 4.2.2. Since F_2' uses an equality measure to compare attribute values, it works equally well for numeric and symbolic attributes.

F_3': Maximum Attribute Efficiency

The attribute efficiency measure is similar to F_2' in that it computes how much the attributes discriminate based on the overlap in their values. However, instead of taking the measure to be the value of the overlap, attribute efficiency counts the number of cases that fall outside the overlap. The count, as a fraction of the total number of cases, is the discriminative power of this attribute. This is presented in Equations 4.9 and 4.10.

$$AttributeEfficiency(a) = \frac{\sum_{x \in CB} OutsideOverlap(x, a)}{|CB|} \tag{4.9}$$

$$OutsideOverlap(x, a) = \begin{cases} 1 & \text{if } x_a \notin Overlap'(a) \\ 0 & \text{otherwise} \end{cases} \tag{4.10}$$

The complexity is then the maximum attribute efficiency over all the attributes as shown in Equation 4.11. We designate it F_3', rather than F_3, to emphasize that it uses our new definition of overlap.

$$F_3' = \max_{i=1...n} AttributeEfficiency(a_i) \tag{4.11}$$

A high value for this metric means low complexity because it means that the attribute discriminates well between the classes.

Problems with this Measure

Overlap Calculation. To define what it means to be outside the overlap requires a definition of overlap. Ho and Basu use $Overlap(a)$ as defined in Equation 4.3, but we have seen that this has problems and limitations. Hence, in our definition of what it means to be outside the overlap we again replace $Overlap(a)$ by $Overlap'(a)$.

Multiple Classes and Symbolic Attributes Since the F_3 measure uses the same overlap computation as the F_2 measure, it suffers from the same weaknesses that we described for F_2. Our F_3' measure works for both multi-class case bases and for symbolic attributes because it uses our modified overlap measure that we use in F_2'.

F_4': Collective Attribute Efficiency

Orriols-Puig et al. extend maximum attribute efficiency to collective attribute efficiency [64], which computes the collective discriminative power of all of the attributes (F_4).

To compute F_4, the most discriminative attribute is determined using F_3. The cases discriminated by this attribute are then removed from the case base. In other words,

each case whose value for this attribute is a value that only co-occurs with one class label is removed from the case base. The process is repeated on this reduced case base: of the remaining attributes, the one with highest efficiency is determined and cases that are discriminated by this attribute are removed. This continues until all of the cases have been discriminated by some attribute, all of the attributes have been used for discrimination, or the remaining attributes cannot discriminate the remaining cases.

The metric, F_4 is the proportion of the case base that has been discriminated, i.e. the number of cases removed in total by this process as a proportion of the original case base size. Again high values mean low complexity.

Problems with this Measure

Overlap Calculation. Since F_4 again uses an overlap computation, it suffers from the same problems as F_2 and F_3. We again use our revised definition of overlap and call our version of the measure F_4' to differentiate it from the original measure.

Multiple Classes and Symbolic Attributes F_4' works for both multiple classes and for symbolic attributes in the same way as F_3' since it uses F_3' in its computation.

4.3.2 Measures of Separability of Classes

The measures in this category rely on computing distances between cases. A substantial literature shows how to define distance (or similarity) in the case of symbolic-valued attributes, as well as numeric-valued ones (e.g. [65, 96]). With these definitions of distance (or similarity), almost all of the measures in this category can all be used on case bases, irrespective of the type of their attributes. The only exceptions to this are L_1 and L_2, for reasons explained in their descriptions. Equally, all of the measures in this category except L_1 and L_2 can be used for both Boolean classification case bases and multi-class case bases without any modifications.

However, some of the measures in this category and the next category make use of a classifier, or techniques used within classifiers, which raises questions about whether as measures of case base complexity they are independent enough of the classifiers whose performance they purport to predict.

N_1': Percentage Points on Boundary

This measure, N_1, examines the boundaries between classes and calculates an estimate of the length of these boundaries [37, 38]. To compute it, we build a minimum spanning tree (MST), connecting all the cases with minimum overall distance between the cases. Within this tree, we count the number of cases that are directly connected to cases of a different class. These are assumed to be the cases that lie on the class boundaries. We express this count as a fraction of the total number of cases. A high value implies a complex case base, in which many cases lie on the boundaries.

Problems with this Measure As pointed out by Ho and Basu [38], this measure can give a high value in two possible situations. It gives a high value if the boundary is very complicated and cases of different classes are interleaved. This correctly tells us that the boundary is hard to learn. However, the measure will also give a high value for a case

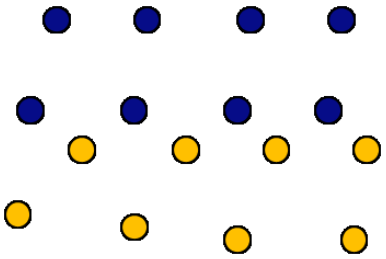


Figure 4.3: Linearly Separable Boundary with a high N_1' value

base that is linearly separable, hence not complex, but the distances between cases on the boundaries are smaller than distances between cases of the same class. An example of this is shown in Figure 4.3.

A problem that is not discussed in [37, 38, 64] is that MSTs are not unique. The particular MST that an algorithm finds often depends on the order in which cases are presented to it. If cases further from boundaries are presented first, the complexity will be lower than when boundary cases are presented first. To investigate this, we ran the measure on different orderings of the case bases. We found that the different order did cause a change in the results. In our version of this measure, N_1', we shuffle the case base ten times, compute the measure on each ordering and report the mean of the results.

N_2: Average Intra/Inter Class NN Distance Ratio

Similarly to N_1', this measure looks at the distance between cases of different classes. For each case, x, we compute the distance to its nearest neighbour of the same class (the intra-class distance, $IntraClassDist(x)$) and the distance to its nearest neighbour of a different class (the inter-class distance, $IntraClassDist(x)$). We then compute N_2, the average intra/inter class distance ratio, as shown in Equation 4.12.

$$N_2 = \frac{\sum_{x \in CB} IntraClassDist(x)}{\sum_{x \in CB} InterClassDist(x)} \tag{4.12}$$

This measure gives an indication of how cases are spread throughout the space. A low value indicates that cases of the same class lie closely together with greater distance to cases of different classes. This type of case base would be considered to be less complex than one where the value is high, in which inter-class distances are smaller relative to intra-class distances.

N_3: Error Rate of 1-NN Classifier

This measure looks at how close cases of different classes are to each other. It computes the leave-one-out error rate of the one-nearest-neighbour classifier on the case base. A high value means that the majority of cases in the case base are closer to a case of a different class than to a case of their own class. This indicates a case base with a complex class boundary.

N_5': Separability Emphasis

N_5' is the first of the measures that we took from the CBR literature rather than from from [38] or [64]. It is, however, based on two of Ho and Basu's measures [38].

Fornells et al. propose a visualization that they refer to as a complexity map [22]. Each case base is a point on a graph, where the x-axis is F_3, the maximum attribute efficiency, and the y-axis is the product of boundary length (N_1) and the ratio of average intra/inter class distance (N_2). The product is taken in order to "emphasize their behaviours" [22, p.421]. We will also use this product as a complexity measure, N_5, but we calculate the product using our revised version of N_1: $N_5' = N_1' \times N_2$.

L_1: Minimised Sum of Error Distance of a Linear Classifier

This is a measure of the linear separability of the case base. Ho and Basu propose using a linear programming formulation, where the objective function to be minimised measures error [38]. The value of this function is used as the case base complexity. In DCoL, Orriols-Puig et al. instead use a support vector machine (SVM), which is trained on the case base using the sequential minimised optimization (SMO) algorithm [64].

L_1 is then the sum of the absolute differences between the actual class of each case (coded as +1 or -1) and the value of the learned function on that case. A value of zero means that the case base is linearly separable and therefore not complex.

Multiple Classes Although SVMs have been extended to handle multi-class classification (e.g. [12]), the DCoL implementation only handles Boolean classification. We use the DCoL implementation, and for multi-class problems, we resort to OVA.

Symbolic Attributes SVMs do not handle symbolic values and so some encoding must be used if the measure is to be used on case bases that have symbolic attribute values. As in previous measures, the DCoL implementation assigns unique integers but, again, this will not be correct unless the distance function on this attribute can be modified to reflect the original semantics of the symbolic values. As a result, we do not have an implementation of this measure for case bases with symbolic attributes. One way that we could allow it to work with symbolic values would be to use a binary encoding, which would replace the symbolic-valued attribute by several Boolean-valued ones. However, this is an area for future work.

L_2: Training Error of a Linear Classifier

To compute this measure, we build the same linear classifier as in L_1 and we compute its error rate on the original case base. In other words, we compute the proportion of the training set that the classifier misclassifies. Again a value of zero means that the case base is linearly separable and therefore has low complexity.

Multiple Classes and Symbolic Attributes Since L_2 uses the same linear classifier as L_1, it suffers from the same limitations on both of these issues.

C_1: Case Base Complexity Profile

C_1 is the second of our measures that we have taken from the CBR literature. Massie et al. [54, 55] present a complexity measure that computes the complexity of a case by looking at the classes of its nearest neighbours. This measure looks at successively larger nearest neighbour sets by retrieving the k nearest neighbours of a case x for $k = 1$, and then the nearest neighbours for $k = 2$, and so on up until k equals some parameter K. For each set of k neighbours that it retrieves, it computes $P_k(x)$, the proportion of x's k neighbours that are of the same class as x. The different values of $P_k(x)$ for $k = 1 \ldots K$ are then averaged and this average is subtracted from one so that a high value of the measure indicates a high degree of complexity. The measure is shown in Equation 4.13.

$$\mathit{Complexity}(x) = 1 - \frac{\sum_{k=1}^{K} P_k(x)}{K} \tag{4.13}$$

The closer a neighbour is to a case x, the more influence it has on the measure, since it will be used to contribute to the computation of several values for $P_k(x)$ for successively large values of k. For example, x's nearest neighbour contributes to all values of $P_k(x)$ for $k = 1 \ldots K$.

Massie et al. use this case complexity measure to compute the complexity of a cluster of cases [54]. They do this by averaging the complexity of the cases in the cluster. We obtain the complexity of the case base in this way, as shown in Equation 4.14.

$$C_1 = \frac{\sum_{x \in CB} \mathit{Complexity}(x)}{|CB|} \tag{4.14}$$

C_2: Similarity-Weighted Case Base Complexity Profile

A potential weakness of Massie et al.'s measure is that, in the computation of any one value of $P_k(x)$, it does not take into account how far away each neighbour is from x. So, for example, when computing $P_2(x)$, both neighbours will contribute equally to the computation, even if one of them is very close to x and the other is quite distant.

In computing C_2, we make a simple modification to C_1 to deal with this issue. This modification is to weight the neighbours by similarity to x so that, in the example, the nearer of the two neighbours would contribute more. To compute $P_k(x)$, we no longer simply compute the proportion of x's k nearest neighbours of the same class as x. Instead, we compute the mean similarity to x of those neighbours with the same class as x. A high value of $P_k(x)$ now indicates to us not only that x has many neighbours of the same class as x, but that these neighbours also lie close to x in the case base.

We compute the complexity of each case and of the case base as a whole in the same way as we did for C_1 (Equations 4.13 and 4.14), substituting in our refined definition of $P_k(x)$.

4.3.3 Measures of Geometry, Topology and Density of Manifolds

L_3 and N_4: Nonlinearity of a Classifier

Ho and Basu's next two measures are based on a measure of nonlinearity of a classifier, originally proposed by Hoekstra and Duin [40]. They consider the nonlinearity of a classifier to be the probability that an arbitrary case, which is linearly interpolated between two randomly-selected cases of the same class, shares this class.

To measure this, a linear classifier is built from the case base. Then a test case base of new cases is created from the original case base. Each new case is obtained by: randomly drawing two cases, x and x', of the same class from the original case base; obtaining the attribute values for the new case by linear interpolation between the attribute values of x and x'; and assuming the new case has the same class as x and x'. Each case in the test case base is presented to the classifier, and the error rate is measured.

There is a family of measures possible by this approach, depending on what type of classifier is used. Ho and Basu propose using a linear programming formulation and 1-NN. We follow Orriols-Puig et al. who use an SMO SVM, as in Section 4.3.2, to give measure L_3, and a 1-NN classifier, to give measure N_4.

Multiple Classes Since N_4 uses a 1-NN classifier, it can deal with multi-class classification. L_3, however, inherits the limitations of the SVM classifier, already discussed in section 4.3.2. As with L_1 and L_2, we resort to OVA for multi-class problems.

Symbolic Attributes Irrespective of the classifier, if we are to create a test set by linear interpolation, we require that all attributes be numeric-valued. As before, DCoL assigns integer values to the symbolic values arbitrarily. Depending on how these values are assigned, different values for L_3 and N_4 can ensue. Hence, these measures are not suitable for case bases with unordered symbolic-valued attributes.

T_1': Fraction of Maximum Covering Spheres

Ho and Basu's T_1 measure looks at how well cases of each class cluster together, using the concept of adherence subsets as described by Lebourgeois and Emptoz [49]. The adherence subset of a case x is a hypersphere of cases around x but within a threshold distance of x and having the same class as x. Each hypersphere initially contains just x. It is grown incrementally by adding the next nearest neighbour until the next neighbour would be of a different class from x or the threshold would be exceeded.

If a case base can be represented using a small number of large adherence subsets, then cases of each class are highly-clustered. To compute this, the measure computes the adherence subsets of all cases and removes each adherence subset that is strictly included in another so that only the biggest adherence subsets are considered. The resulting number of adherence subsets is divided by the number of cases in the case base to give a normalised measure of how clustered the classes are, where low values indicate simple case bases.

We modify T_1 in two ways to get T_1', a simpler and more satisfactory measure. First, we set the threshold to zero, so x's adherence subset contains x itself and all its nearest neighbours up to, but excluding, the nearest unlike neighbour (whose class is different from x's). Second, T_1 uses *strict* containment but T_1' uses non-strict containment, so equal adherence subsets count as one. This is better, as can be seen by thinking about a case base in which all cases share the same class, which is a very simple case base. Since there are no unlike neighbours, each case x has the whole case base as its adherence subset. By T_1, no adherence subset is strictly contained within any other, so the number of maximal adherence subsets is $|CB|$ and this is divided by the number of cases (also $|CB|$), resulting in $T_1 = 1$, which, incorrectly, implies high (indeed, maximum) complexity. On the other hand, since in this example all the adherence subsets are equal, T_1' will ignore all but one of them, hence the number of maximal subsets is 1, which will be divided by $|CB|$, giving the lowest possible value, correctly implying low complexity.

Multiple Classes and Symbolic Attributes T_1 and T_1' can be used directly in cases of multi-class classification and, while Ho and Basu describe the case of Euclidean distances, we can extend the measure to other case bases by defining suitable distance measures on cases that include symbolic-valued attributes.

T_2: Number of Instances per Attribute

Ho and Basu's final measure, T_2, is the number of cases divided by the number of attributes, which gives an indication of case base sparseness. Sparse case bases are problematic for classifiers in general, since we cannot be sure how to generalize over under-represented regions of the problem space. They can also confound some of the other complexity measures, since a complex problem can appear simple if we sample too little of the problem space.

T_3: Case Base Competence

The last of the measures that we consider, T_3, is another that we define from ideas in the CBR literature.

Smyth and McKenna propose a measure of case base competence [85]. We note that this has not been proposed as a measure of complexity, and it may not prove to be useful as such. Informally, it is a prediction of the range of problems that a case base can solve. We consider here whether it can be used as a measure of case base complexity. We present it only in the context of k-NN classification.

Smyth and McKenna define the coverage set of a case x in a case base to be the set of all of cases x' in the case base each of which is correctly classified by its k neighbours, x is one of those neighbours, and x has the same class as x'. However, coverage sets alone cannot be used to calculate the competence of a case base because coverage sets can overlap. Cases form clusters around well-defined areas of competence in the case base, with large, dense clusters indicative of common problems, and smaller clusters (or even single cases) indicative of more unusual problems. Smyth and McKenna define these clusters as being the fundamental unit of competence within the case base and call them *competence groups*.

Two cases belong to the same competence group if they have shared coverage, i.e. the intersection of their coverage sets is non-empty; and competence groups are maximal in the sense that any case that shares coverage with another will be in the same competence group. Group coverage for a competence group is defined heuristically, being proportional to group size and inversely proportional to similarity within the group. Dataset competence is then calculated as the sum of the coverages of each of the competence groups. If we treat this as a complexity measure, high values imply good coverage and hence a case base with low complexity.

Multiple Classes and Symbolic Attributes As with the other measures that use k-NN as the basis for their measure computation, this measure is well-defined for multi-class classification and for symbolic-valued attributes.

4.4 Criteria for Complexity Measures

If we are to compare complexity measures, we need criteria or desiderata. A non-exhaustive list of desiderata might include the following:

Predictive We require the measures to be predictive of the different characteristics of case bases. For example, if we are comparing classifiers, we need measures that help us to select case bases that will reveal the behaviour of the classifiers under different conditions.

Independent of what is being analysed We must avoid measuring that which we are optimizing. It is preferable that measures that are computed using a classifier should not be used to analyse that same classifier. This may disqualify use of some measures, or some classifiers, in some analyses.

Widely applicable across case bases Multi-class classification is common, so we require measures that can be computed on multi-class case bases as well as Boolean classification case bases. In CBR especially, symbolic-valued attributes are also common, as are instances which have attributes for which the value is missing.

In Table 4.1, we have already indicated which measures can directly handle multi-class classification (where they cannot, the OVA method can be used). We have also indicated whether the method can handle symbolic-valued attributes. If a solution to this problem requires that distance (or similarity) functions be defined over the values of these attributes, the entry is *sim.* Where the measure can in principle handle multi-class classification or numeric-valued attributes but where DCoL does not handle them, the entry is *Not DCoL.* A careful analysis of the applicability of the measures to case bases with missing values has not yet been carried out, and so remains an issue for future work.

Cheap-to-compute Efficiency is important but not an over-riding factor. Dataset complexity measures are not usually part of the regular day-to-day operation of a classifier, therefore 'real-time' performance is not generally needed. Rather, they are usually part of an off-line, and typically one-off, analysis. (An exception to this is our own research into meta-case-based classification for making case base maintenance decisions, which we will present in the next chapter.)

We have not conducted our own analysis into the running-time and memory requirements of the algorithms that compute the complexity measures, but there is a table of computational costs in [64, p.9]. The table shows the measures in Ho and Basu's first category to be linear in $|CB|$, the number of cases; those in their second category are quadratic in $|CB|$; and those in their third category are quadratic (T_1 and T_1'), constant (T_2) or depend on the linear classifier, which in the case of an SVM trained using SMO is sub-quadratic. We believe that the measures from CBR (C_1, C_2, N_5' and T_3) are all quadratic in $|CB|$.

Incremental Similarly, there may be occasions where one has calculated the value of a measure, a case gets inserted into, or removed from, the case base, and we want to compute the new value of the measure, preferably without doing so 'from scratch', by exploiting the value already calculated. Some of the measures are amenable to this; others are not. We will discuss this issue in detail in Chapter 6.

Transparent and explainable The analytic value of a measure may depend in some circumstances on it being intelligible to those using it.

In the next section, we analyse the values of the complexity measures when run on different case bases and investigate which of the measures work better to predict whether a

case base is going to be easy or hard to classify. We also look at them from a maintenance point of view and investigate if there is a link between the complexity of a case base and the maintenance algorithm that works best for that case base.

4.5 Evaluation

4.5.1 Experiment Objectives

This experiments aims to investigate the usefulness of the case base measures. We firstly look at how well the measures correlate with each other to see if they agree on which datasets are complex and which ones are not. Our second objective is to test whether or not the measures agree with measured classifier accuracy of different classifiers on the datasets. If the measures are to be believed, they should indicate that datasets with high accuracy over a number of classifiers have low complexity, while datasets with generally low accuracy should be flagged as being complex. Finally, we look at how maintenance of the datasets affects the measures, the aim here being to see whether the algorithms generally increase or decrease complexity.

4.5.2 Test Data

In these experiments we again use the twenty-five datasets that we have used in the previous chapters. We run the measures on the 60% splits since these are the datasets that we use for our training sets.

4.5.3 Experiment Methodology

We calculate the measures for each of the ten splits of the data and average their results to get an overall dataset value for each measure. We then normalise the measure values to a [0,1] range for all of the measures so that the values would be comparable. Finally, for most of the measures, a low value of the measure indicates low complexity. However, a subset do the opposite, with a low value indicating high complexity. For ease of comparison, we inverted the measure values for F_1, F_3', F_4' and T_3 by subtracting each of their normalised values from 1. This allows us to compare the results more easily since low and high measure values now indicate low and high complexity respectively for all of the measures.

We now examine the results of the measures. We look at how well their results correlate with each other to see how much they agree in predicting whether a dataset is complex or not. We also look at how a number of different classifiers perform on the datasets and see if they do well on the datasets that the measures predict as being easy and vice versa. We finally examine the effect that maintenance algorithms have on a number of the measures and whether they increase or reduce the complexity of the datasets.

4.5.4 Comparing Measures

Firstly, we look at the measures and their values. We examine how well they correlate with each other and where the measures agree and disagree about which datasets they consider to be hard and easy.

There are a number of measures that we expect to correlate well because they are so alike. C_1 and C_2 are very similar variations of the same algorithm so it is not surprising that these have the highest correlation coefficient of any two measures at 0.98. Both also

correlate extremely well with N_1', N_3 and N_5', and each of these three measures correlates well with the other two. This is also to be expected since all of these measures use nearest neighbour techniques.

We would also expect the two attribute efficiency measures to correlate well since they are both based on the same computation. However, they only have a correlation value of 0.62. This indicates that the collective attribute efficiency does not often stop after the first attribute, since it would give the same value as the maximum attribute efficiency if it did. So it is not common for the case bases that we have examined to have one attribute that totally distinguishes between the classes.

There are also a number of measures which correlate particularly well that we would not necessarily have expected. F_4' correlates with N_4 with a correlation coefficient of 0.87. This is unusual since the two measures are very different. F_4' is the collective attribute efficiency, which examines the overlap between attribute values for different classes and which cases are outside this overlap. N_4 is the nonlinearity of the 1-NN classifier, which measures the 1-NN error of a test case base created by linear interpolation between pairs of cases in the original case base. These two measures appear to be very different, but yet their results show that they are computing a similar property of the case base.

We also examine measures that do not correlate well with the others. Two measures in particular, F_1 and T_1', do not correlate well with any of the other measures. Their highest correlation coefficients with any other measure are 0.37 and 0.38 respectively. This indicates that these measures do not often agree with many of the other measures on whether or not a case base is complex. It is interesting that F_1, the maximum Fisher's discriminant ratio, does not even correlate well with the other three measures that also calculate complexity based on how much attribute values overlap. T_1', the fraction of maximum covering spheres measure, is not similar to any of the other measure in its approach to computing complexity, so it is less surprising that it doesn't correlate well with them.

Taking specific pairs of measures, we see that the pairs with the least correlation to each other are N_1' and T_1' with a correlation coefficient of 0.005, C_1 and T_1' with a coefficient of 0.007 and L_2 and F_3' with a coefficient of 0.04. Since each pair consists of two very different measures, it is not very surprising that they do not agree with each other. A pair that we would expect to be quite similar but that do not correlate well are L_1 and L_3. L_1 measures the minimised sum of error distance of a linear classifier, while L_3 measures the nonlinearity of a linear classifier. Although the measures are in different categories, they both have the element of testing the error of a linear classifier so it would be expected that they would be measuring a similar property of the case base. However, they are actually very slightly negatively correlated, with a correlation coefficient of -0.01.

In summary, we can see that while all of the measures aim to compute case base complexity, some are more similar than others in what they measure. In the next section, we use a number of classifiers to see which datasets they find easy and hard for classification and we examine which of the measures agree with these classifiers.

4.5.5 Predicting Classifier Error

Classifier Accuracies

Since the measures aim to predict how easy or hard the datasets are for classifiers, we now compare their results with a number of commonly-used classifiers. To do this, we built each of seven Weka classifiers [32] on the twenty-five training sets. Since some of the complexity

measures are computed using nearest neighbour techniques or SVM classifiers, we made sure we included additional classifiers. Even so, there is a danger that the classifiers that we have chosen have insufficiently different biases, and we must view our results with this in mind. The seven classifiers were:

- Multilayer Perceptron: A neural net with one hidden layer.
- SMO: A support vector machine.
- J48: A C4.5 decision tree learner.
- IBk: A nearest neighbour learner.
- NaïveBayes: A Naïve Bayes classifier.
- AdaBoost: The AdaBoost classifier.
- OneR: Rule learner which creates rules based on a single attribute.

Each was built using its default Weka settings except for the perceptron. We found that it was better to use one hidden layer, instead of the default value of no hidden layers. We recorded the percentage error of each classifier on the held-out test sets. We found that three of the classifiers, Naïve Bayes, AdaBoost and OneR, were outperformed by the others across the datasets and so we omitted them from our final analysis. The results of repeating the splitting, training and testing ten times and averaging the percentage errors are shown for each of the four remaining classifiers in Table 4.2. We have also included a column in the table for the mean error over the four classifiers and the table is sorted in ascending order of this value. The lowest error value for each dataset is indicated in bold.

The classifiers tend to agree on which datasets are easy to classify (the low error ones, such as the Spam and Iris datasets) and which are difficult (the higher error ones, such as the Heart Disease and Lung Cancer datasets). We confirm this by correlating the error, Table 4.3, and observing that the correlation coefficients are between 0.93 and 0.98.

We can therefore take the mean error over the four classifiers as an indication of complexity. We can now see whether the complexity measures agree with these classification error results. We are aware of a major irony here. We have selected these twenty-five datasets in precisely the way we criticized in Section 4.1. But we are compelled to operate in this fashion. We cannot select them using dataset complexity measures, for to do so would mean that their selection was not independent of that which we wish to experiment with, which in this chapter are measures of dataset complexity themselves.

We have included the full correlation tables between the classifiers and the measures in Appendix A. Table A.1 shows the correclations based on all twenty-five datasets, and Table A.2 shows the correlatio coefficients based on just the fourteen Boolean classification datasets.

Comparison between Classifier Results and Measure Results

DCoL's Thirteen Measures Of the thirteen measures in DCoL, N_1, the fraction of instances on a boundary, computed from the MST, is one that other studies have found to correlate well with classifier performance [5, 6, 51]. With our modified version of this measure, N_1', we also find high correlations with the performance of our classifiers on our datasets: for example, the correlation coefficient between N_1' and the mean error is just

Table 4.2: Classification error

Dataset	NN	SVM	J48	IBk	Mean
Spam3	0.50	**0.40**	3.25	1.75	1.48
Spam4	**1.60**	1.95	6.20	3.05	3.20
Iris	**2.67**	4.00	5.00	**2.67**	3.58
Dermatology	3.38	**2.39**	6.34	3.10	3.80
Spam2	**2.95**	3.45	4.65	5.40	4.11
Breast Cancer Diagnostic	3.19	**2.57**	7.26	3.54	4.14
Spam5	**2.20**	2.70	7.65	5.50	4.51
Spam1	**2.15**	2.75	7.20	6.40	4.63
Wine	3.33	**1.67**	13.33	2.67	5.25
Zoo	8.50	**7.00**	12.50	10.50	9.63
Ionosphere	10.43	12.43	**10.29**	14.29	11.86
Credit Approval	15.38	14.08	14.23	**13.00**	14.17
Balance	**8.88**	13.44	22.40	16.88	15.40
Hepatitis	20.00	**18.13**	18.75	21.88	19.69
Breathalyser	**21.60**	26.40	22.80	28.00	24.70
Breast Cancer Prognostic	24.47	**23.16**	27.11	30.53	26.32
Pima Indians Diabetes	25.69	**23.53**	28.56	27.84	26.41
Haberman's Survival	26.23	**23.77**	25.25	31.15	26.60
Lenses	32.50	40.00	**22.50**	27.50	30.63
Post-Operative Patient	38.82	30.00	**29.41**	36.47	33.68
Liver Disorders	**33.77**	42.32	37.10	37.10	37.57
Glass Identification	**35.71**	53.57	36.43	36.67	40.60
Teaching Assistant Evaluation	**43.33**	45.67	45.33	43.33	44.42
Heart Disease Cleveland	46.10	**43.90**	49.49	46.95	46.61
Lung Cancer	58.00	50.00	**46.00**	56.00	52.50

under 0.96. We plot this in Figure 4.4: each point is a dataset; the x-axis is error and the y-axis is the value of N_1'. Recall from Section 4.3.2 however, that Ho and Basu point out that a high value for this measure can arise in complex datasets when the boundary is complicated but also in simple datasets that are linearly separable but where the distances between instances on the boundaries are smaller than distances between instances of the same class.

In fact, in our experiments, of the thirteen measures, it is N_3, the Error Rate of a 1-NN Classifier, that has highest correlation with mean classifier error. Its correlation coefficient with mean error is just over 0.96, and it correlates slightly better than N_1' with each of the individual classifiers other than the SVM. Of course, it has the unfair advantage that it is based on a classifier (1-NN) that is very similar to one that we are using to judge it (IBk). We already mentioned in the previous section that N_1' and N_3 correlate extremely well, so, if choosing one to use over the other, the fact that N_1' is less similar to the IBk classifier might be a reason to prefer it.

Of the thirteen, the measures that correlate next best with error are N_2, the Ratio of Average Intra/Inter Class Distance (coefficient 0.84), N_4, the Nonlinearity of a 1-NN Classifier (coefficient 0.77), and L_2, the Training Error of a Linear Classifier (coefficient 0.64), although we were able to compare N_4's and L_2's values with the error rate on only

Table 4.3: Classifier correlation coefficients

	SVM	J48	IBk
NN	0.95	0.95	0.98
SVM		0.93	0.95
J48			0.97

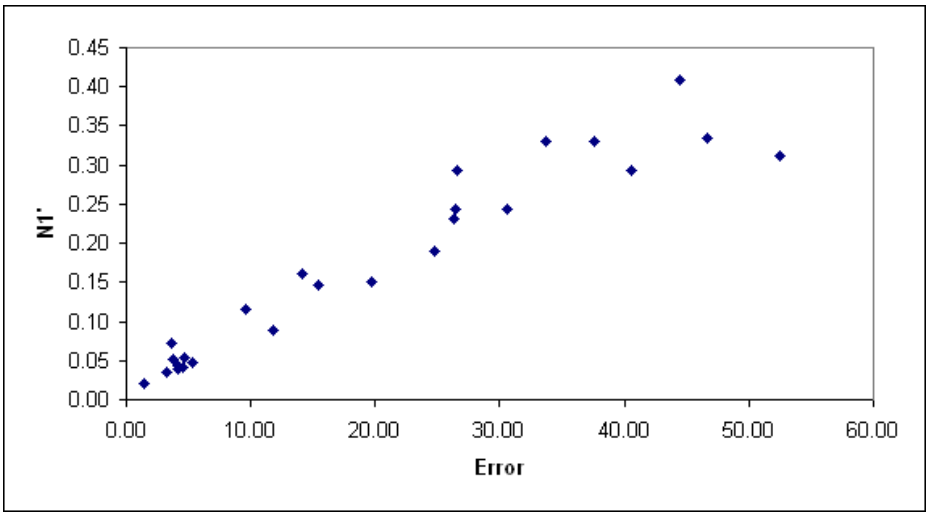


Figure 4.4: Mean classifier error against N_1' for twenty-five datasets

twenty-one of the datasets because they cannot handle the symbolic values in the other four datasets.

Of the remaining of the thirteen measures, three (F_4', L_3 and T_1') have coefficients around 0.5, and the rest have coefficients close to zero.

For four of the measures (see Table 4.1), we were resorting to the OVA method to handle multi-class classification. We were concerned that the averaging involved in doing this might be detrimental to the performance of these measures. So we looked separately at how the measures correlate with error for just the Boolean classification datasets: fourteen of the twenty-five (or thirteen in the case of those measures that cannot handle symbolic values). On just these datasets, the five best-performing measures though remain the same: N_1', N_2, N_3, N_4 and L_2. The correlation coefficient for N_1' and N_3 are the highest (0.98 and 0.97). N_4's correlation with mean error increased a little and, by contrast, N_2's fell a little. The three measures that use a linear classifier all require OVA to be able to work with multi-class classification. Each of these measures correlates much better with error for just the Boolean classification datasets. L_1's correlation coefficient increases from 0.05 to 0.34, L_2's coefficient increases from 0.64 to 0.96 and L_3's coefficient increases from 0.57 to 0.95. This suggests that the OVA method damages the measures and should not be used, with the consequence that certain measures will be confined to Boolean classification domains, unless some other way of extending them to multi-class classification can be found.

We should emphasize, however, that the goal here is not to find the one 'true' measure. There will not be such a measure. Any sense that a small number of measures adequately characterizes the datasets in our study is more likely due to using too few datasets. In [6], for example, while N_1 is found to be most useful, N_2 and the nonlinearity measures, L_3 and L_4, are also found to be useful. And, even though the measures of overlap of attribute

values, such as F'_4, do not correlate well with error in our study, and have not seen much support in the other studies that we have been citing, they might be relevant, for example, if judging the applicability of decision tree techniques to a dataset.

Complexity Profile and Similarity-Weighted Similarity Profile We turn now to the measures that we have based on work in the CBR research literature. The first of these, C_1, is the Complexity Profile, which we defined as the mean case complexity [55]. We computed this with its parameter $K = 10$. Although designed for a different purpose, we find that, of all our measures, this is the best correlated with classifier error, the correlation coefficient being 0.98. We should, however, add the caveat that we are using only four classifiers and twenty-five datasets. It would certainly be interesting to look at values of C_1 across a large set of specially-constructed artificial datasets, in the style of [38]. Additionally, as with N_3 in the previous section, C_1 uses nearest neighbour techniques in its computation so it has the advantage of being similar to the IBk classifier. Perhaps surprisingly, C_2, our modified version of C_1, which weights neighbours by their distance to each instance, does not correlate as well with mean error as C_1 does. Its correlation coefficient is 0.97, which is still better than all the others. And, where we look at Boolean classification only, C_1 and C_2 remain pre-eminent.

Separability Emphasis Measure N'_5 is also from the CBR literature [22]. It is the product of N'_1, which correlates very well with error, and N_2, which correlates reasonably well. It is no surprise that it correlates slightly less well with error than N'_1 and somewhat better than N_2. There seems then no reason to prefer this measure over others. Empirically, C_1, C_2, N_3 and N'_1 on their own are to be preferred.

Dataset Competence We also implemented from the CBR literature Smyth and McKenna's dataset competence, and treated this as a measure of dataset complexity, T_3. It was never designed as such, and it is therefore no criticism of the competence idea to find that it does not correlate with classifier error: it is inversely correlated and the coefficient is close to zero, -0.17.

Summary We can predict classifier error using any of C_1, N_3 and N'_1. C_2 and N'_5 also correlate well with classifier error but do not appear to be different enough from C_1 and N'_1 respectively. L_2 also correlates well with error when used on our Boolean classification datasets, but less well when used with the OVA method on multi-class classification and we have not been able to use it on the datasets that contain symbolic-valued attributes. We would need an implementation that overcomes the limitations of the one in DCoL to properly judge its applicability. Finally, N_4 is moderately predictive of classifier error, but also limited to datasets with only numeric-valued attributes.

What is interesting is that all of the measures mentioned in the previous paragraph except for N_4 are measures of the separability of classes, Ho and Basu's second category of measures. A similar result is found in [5]. It is convenient too that all but two of these measures can work with multi-class datasets and symbolic-valued attributes, so they are useful for CBR, where both of these are common.

Another observation is that some are not wholly independent of the classifiers whose performance one might want to predict. N_3, L_2 and N_4 make direct use of classifiers in their computation, either a 1-NN classifier or a linear classifier (in DCoL, this being an

SVM). It can be argued that C_1 and C_2 are using nearest neighbour techniques and are therefore not wholly independent of k-NN classifiers. Perhaps N_1' and N_2 are the ones that take approaches that are most different from the way that current classifiers operate.

4.5.6 Effect of Maintenance Algorithms on Measure Values

As our work centres on the area of case base maintenance, we wanted to examine the effect that running the different maintenance algorithms would have on the measures. To do this, we ran each of the atomic maintenance algorithms and each of the classic composites on the twenty-five datasets and recorded the percentage of cases deleted and the accuracy before and after maintenance. Additionally, we computed the values of each of the seventeen complexity measures on the case bases before and after maintenance.

We then chose three of the complexity measures to study in detail. We chose one from each of the categories, that would best reveal the changes that the maintenance algorithms bring about. We also chose measures that are computed in a manner that is, as much as possible, independent of the way the maintenance algorithms work and the way we measure changes in accuracy.

Measures for Detailed Analysis

Measures of Overlap of Attribute Values From our analysis in the previous section, we found that these measures are not predictive of classifier accuracy. But we might expect them to be much more revealing of the effects of maintenance, especially of noise reduction. An incorrectly-labelled case may decrease the discriminatory power of one or more attributes.

We choose to use F_3' for this analysis. We rule out F_1 because we cannot apply it to datasets that include symbolic values or multi-class classification and we avoid F_2' because we found that most of its values on most datasets, both pre- and post-maintenance, tend to be zero. F_4' would be just as good as F_3': there is not much to choose between them here.

Measures of Separability of Classes We exclude L_1 and L_2 because they cannot handle multi-class classification problems and symbolic-valued attributes and we exclude N_5' because it is simply the product of N_1' and N_2 and so produces values intermediate between the two. When examining C_1 and C_2, we found that, for all maintenance algorithms and about four-fifths of the datasets, C_1 increases after maintenance, suggesting that the resulting case base has become more complex, not less. In the roughly one fifth of datasets where this does not happen, the measure either remains the same or falls but only very slightly. In the case of C_2, the story is the same, an increase in complexity for the most part, but there are a few more exceptions, where complexity stays the same or decreases. Overwhelmingly, the other measures in this category show case bases becoming less complex after maintenance, and since C_1 and C_2 disagree, we exclude them.

This leaves N_1', N_2, N_3. We exclude N_3: it is computed using a 1-NN classifier, and therefore may not be independent enough of the maintenance algorithms since many of them use k-NN. The other two measures, N_1' and N_2 are reasonably independent of the maintenance algorithms. We choose N_1' since we found it to be more predictive of classifier accuracy than N_2.

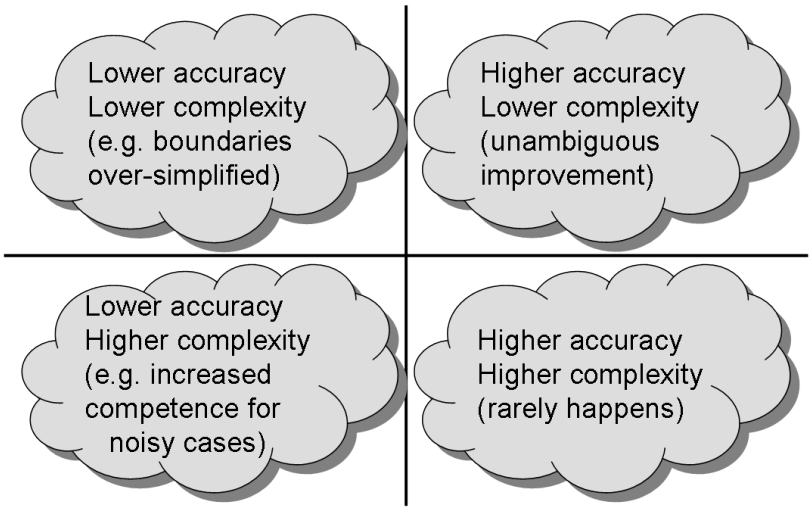


Figure 4.5: Interpretation of graphs in this analysis

Measures of Geometry, Topology and Density of Manifolds In this category, it is easy to choose T_1'. We exclude L_3 and N_4 because they cannot be computed on datasets with symbolic-valued attributes. T_2 is simply the number of instances in the dataset divided by the number of attributes: since the former will either stay the same or decrease and the latter will not change after maintenance, this measure is giving little insight beyond what we learn already from recording the percentage of cases deleted, so we exclude it. And, as well as noting in the previous section that T_3 was not predictive of classifier accuracy, we have not found it to be useful here either: for all case bases and all maintenance algorithms, it shows complexity increasing, rather than decreasing, after maintenance. This leaves T_1', which was not very predictive of classifier accuracy but it behaves as we would expect for maintenance, mostly showing a decrease in complexity after maintenance.

In summary, then, we will proceed to use F_3', N_1' and T_1' in the rest of our analysis.

Results

In this analysis, we make a small change to how we normalise our complexity values. We invert them so that a positive change means a lowering of complexity and vice versa. We do this for the benefit of graphs so that moving upwards and rightwards in the graphs is a good thing, corresponding to decreases in complexity and increases in classification accuracy respectively. We show this in Figure 4.5. The upper-right quadrant shows that the maintenance algorithm has unambiguously improved accuracy and complexity. The top-left quadrant signifies lower complexity but also lower accuracy. This might occur if the maintenance algorithm over-simplifies class boundaries. The bottom-left quadrant signifies higher complexity and lower accuracy. There are any number of reasons that this might happen but one is the deletion of correctly-labelled cases near incorrectly-labelled ones, so that the incorrectly-labelled case will be retrieved for a larger number of target problems. The bottom-right quadrant is the situation of lower accuracy and higher complexity. Fortunately, our graphs show this to be very rare. Of course, as we have already talked about, maintenance is a multi-objective optimization problem. Therefore losses in accuracy may be a price worth paying if the case base becomes more compact, and so it is not necessarily a 'bad thing' for a dataset and algorithm to be in a quadrant other than the top-right one.

Table 4.4: Percentage of cases deleted, change in accuracy, and change in normalised complexity measures: mean (and standard deviation) over twenty-five datasets. (In this analysis, a positive change in a complexity measure implies lower complexity.)

	% of cases deleted		Change in % accuracy		Change in F_3'		Change in N_1'		Change in T_1'	
BBNR	23.08	(15.06)	-0.24	(3.15)	31	(31)	6	(5)	4	(5)
RENN	23.17	(18.67)	-2.70	(6.74)	41	(34)	22	(18)	23	(19)
CRR	35.94	(7.86)	-2.00	(2.75)	24	(32)	-14	(7)	-9	(6)
ICFR	61.85	(22.15)	-4.97	(2.75)	26	(34)	-37	(14)	-26	(15)
RCR	77.59	(9.57)	-2.93	(2.43)	32	(34)	-27	(9)	-14	(18)
BBNR→CRR	59.38	(12.58)	-1.54	(3.37)	44	(35)	15	(17)	16	(18)
RENN→ICFR	78.41	(16.12)	-5.90	(6.64)	48	(33)	-5	(26)	10	(22)
RENN→RCR	88.71	(2.11)	-4.25	(6.85)	49	(34)	1	(20)	8	(24)

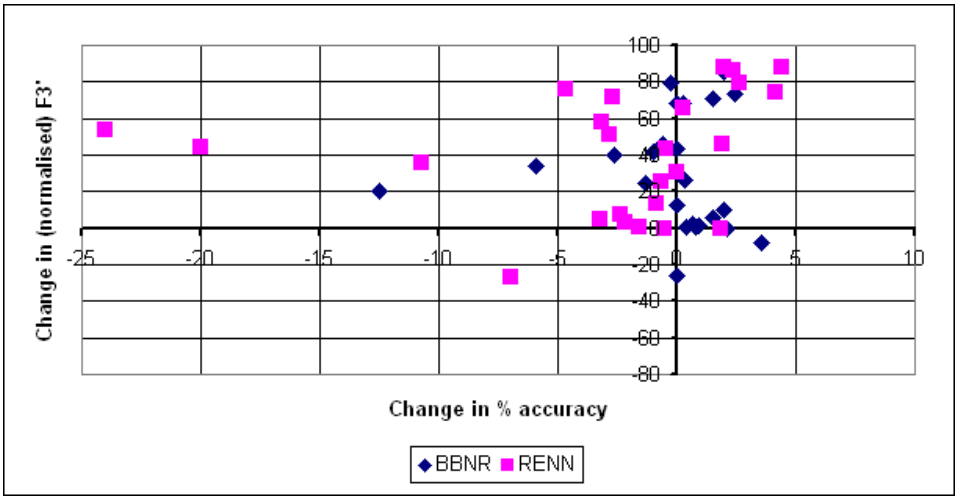
We summarize the results in Table 4.4, which shows averages over the twenty-five datasets, and reports the changes in the three measures that we have selected.

Atomic Noise Reduction Algorithms Figure 4.6 shows results for the atomic noise reduction algorithms, BBNR and RENN. The x-axis is the same in all three graphs in Figure 4.6. It is the change in accuracy on the held-out test set: positive values mean that accuracy is higher after maintenance than it was before maintenance. On the y-axis, we plot the change in the normalised values of the complexity measure: F_3' in Figure 4.6(a), N_1' in Figure 4.6(b), and T_1' in Figure 4.6(c). We repeat the point made earlier that, perhaps counter-intuitively, positive values here mean that the complexity has gone down after maintenance. Hence, the upper-right quadrant of every graph is where we might hope to be: increased accuracy and lower complexity. There are fifty points on each graph, representing what happens to each of the twenty-five datasets in the case of BBNR (diamonds) and in the case of RENN (squares).

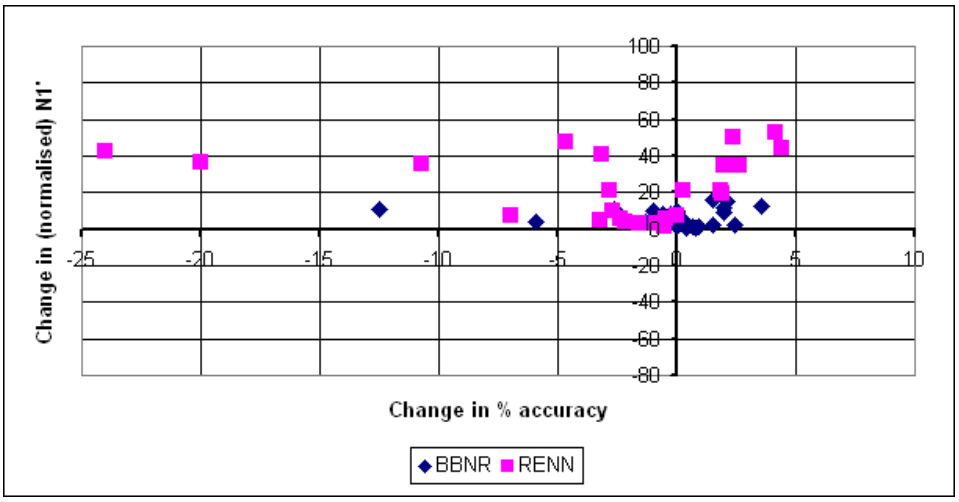
We might expect that a good noise reduction algorithm would either improve accuracy or, if there was no noise, would leave it unchanged. But we see that the algorithms sometimes improve accuracy and sometimes worsen it. BBNR seems to do better than RENN: in eighteen of the twenty-five case bases, BBNR improves accuracy or leaves it unchanged; in only nine of the twenty-five does RENN do the same. In the case of RENN, some quite severe falls in accuracy occur (20-25%). The two case bases on which accuracy falls most after applying RENN are Lung Cancer and Lenses.

We might think that the explanation would lie in the number of cases deleted. But, in fact, for most datasets, the two algorithms delete quite similar numbers of cases. The difference in the percentages deleted is less than or equal to 5% for twenty datasets. The mean percentage deleted over the twenty-five datasets by BBNR is 23.08%, where for RENN it is 23.17%. In only seven of the datasets does RENN delete more, although in three of these the difference in the percentage deleted is more than 10%.

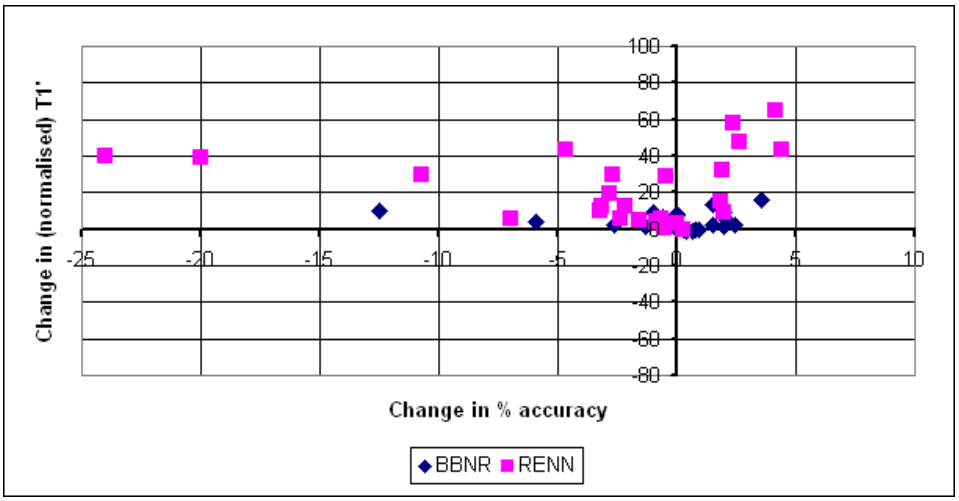
Both algorithms lower complexity, across all three complexity measures, on nearly all datasets. Let's consider F_3' first (Figure 4.6(a)). In an extreme example, if two cases have exactly the same attribute values but are labelled by different classes, then one of the cases is noisy (unless there are other attributes in the domain that would distinguish the two cases



(a) F_3': Maximum Attribute Efficiency



(b) N_1': Fraction of Instances on a Boundary



(c) T_1': Fraction of Maximum Covering Spheres

Figure 4.6: Changes in accuracy and complexity: atomic noise reduction algorithms

but which are not part of the dataset). We would hope that a noise reduction algorithm would delete at least one of these cases and ones like them and this should be reflected in an improvement to F_3'. Figure 4.6(a) shows that both algorithms do seem to be successful in doing this, and to the same degree. The average change in F_3' in the case of BBNR is 31 with standard deviation also 31, and in the case of RENN it is 41 with standard deviation 34.

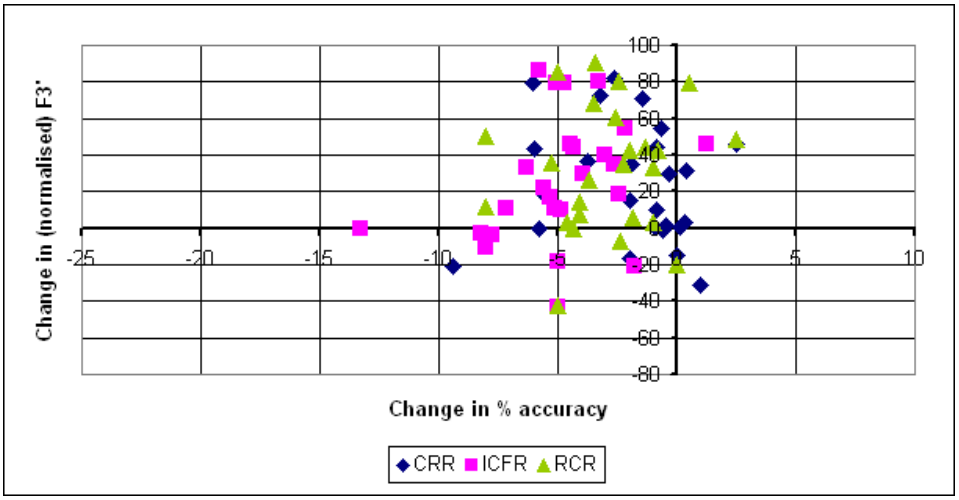
A noise reduction algorithm might be expected also to 'clean up' the boundary between classes: removal of a noisy case should simplify the boundaries. This is what N_1' is supposed to measure, and Figure 4.6(b) confirms that both algorithms succeed in improving this measure of complexity. RENN does this more aggressively than BBNR. Its improvements to N_1' are often greater: it improves N_1' by 22 on average (standard deviation 18) whereas BBNR improves it by 6 (standard deviation 5). But this may explain why RENN loses accuracy and therefore places many datasets in the upper-left quadrant: it may be over-simplifying class boundaries.

Finally, by removing noisy cases, a noise reduction algorithm should produce a case base in which clusters of like cases are fewer in number but larger in size. This is what T_1' is supposed to measure, and Figure 4.6(c) shows again that this is what is happening: the changes are positive (meaning lower complexity), or close to zero, across all twenty-five datasets. Again RENN may be too aggressive: its mean is 23 (standard deviation 19) compared to BBNR's mean of 4 (standard deviation 5).

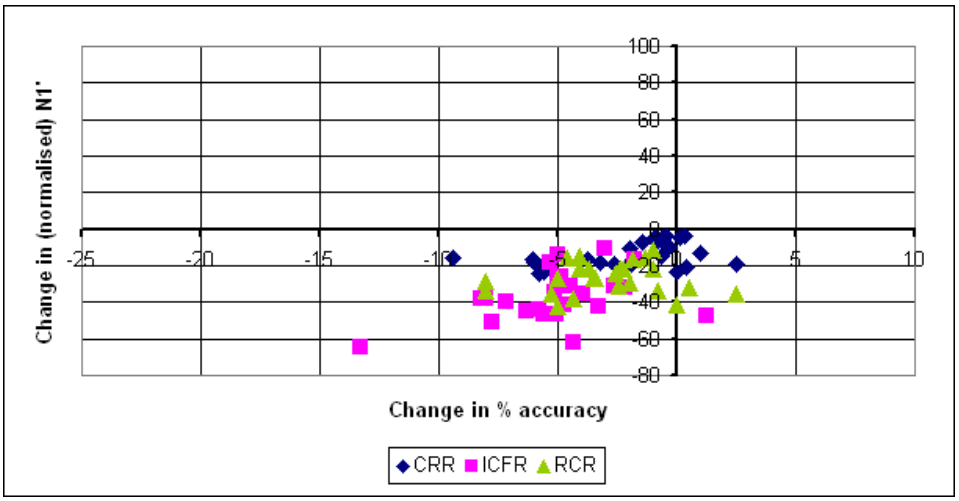
We conclude that, generally speaking, RENN's heuristic for identifying harmful cases is not as successful as BBNR's. It aggressively deletes cases, thereby over-simplifying the case base, achieving larger improvements in complexity but at the expense of classifier accuracy.

Atomic Redundancy Reduction Algorithms Figure 4.7 shows the performance of the three atomic redundancy reduction algorithms (CRR, ICFR and RCR). The big observation across the three graphs in Figure 4.7 is that most points are to the left of the origin on the x-axis, showing that they mostly worsen classification accuracy. Of course this may be because the algorithms are not being applied to the kinds of case bases that they expect, i.e. ones that have undergone noise reduction. Another observation is that CRR (the diamond) tends to give smaller decreases in accuracy (mean -2%, standard deviation 2.75) than RCR (the triangle, mean -2.93%, standard deviation 2.43), which in turn gives smaller decreases than ICFR (the square, mean -4.97%, standard deviation 2.75). This is only partly explained by looking at the percentage of cases deleted. CRR is conservative: it deletes the least and therefore does least damage to accuracy. Indeed it deletes 35.94% cases on average (standard deviation 7.86). Counter-intuitively, though, RCR deletes the most (mean 77.59%, standard deviation 9.57) with less damage to accuracy than ICFR, which deletes fewer cases on average (61.85%, standard deviation 22.15).

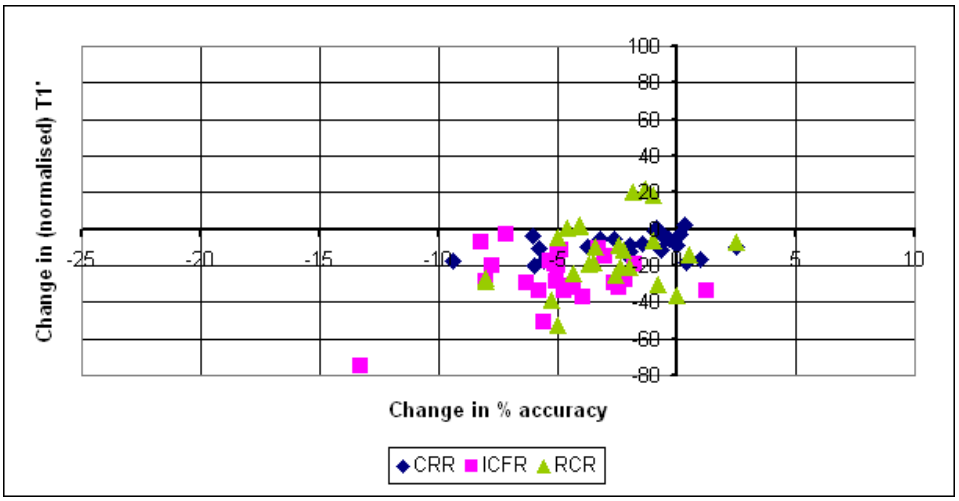
According to F_3', complexity becomes lower (Figure 4.7(a)), but in the case of both N_1' and T_1', complexity increases (Figures 4.7(b) and 4.7(c)). In the case of N_1' and T_1', most points are in the lower-left quadrants. One hypothesis is that noisy cases, which have not previously been removed by a noise reduction phase, remain untouched by the redundancy reduction algorithms, and that correctly-labelled cases are removed. Without these correctly-labelled cases, the noisy cases grow in competence. They will be among the neighbours of a wider range of target problems, leading to reduced classifier accuracy. And, with fewer correctly-labelled cases relative to incorrectly-labelled ones, boundaries are more complex and there are more clusters, and denser clusters, of noisy cases.



(a) F_3': Maximum Attribute Efficiency



(b) N_1': Fraction of Instances on a Boundary



(c) T_1': Fraction of Maximum Covering Spheres

Figure 4.7: Changes in accuracy and complexity: atomic redundancy reduction algorithms

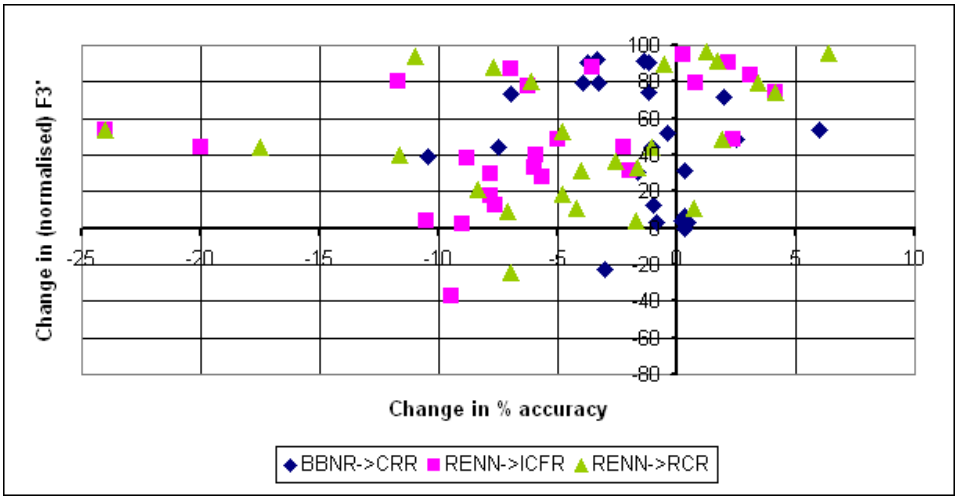
We can illustrate this with a simple thought experiment. Imagine four cases: x_1 is of class c^+, x_2 is of class c^-, and x_3 and x_4 are of class c^+. Assume that they are connected linearly in the order described in a minimum spanning tree. The value for N_1 is 0.75 (three of the four cases are connected to cases of a different class). Let's suppose that we think that x_2 is noisy by virtue of having a different class from its neighbours. If we run a redundancy reduction algorithm, without first removing noise, we are likely to remove either x_3 or x_4. In this case, N_1 goes down to 0.66 (two of the three remaining cases are now connected to cases of a different class). Suppose, instead, that x_2 is first removed by a noise reduction algorithm; N_1 is 0, and remains 0 after a redundancy reduction algorithm removes one or two of the remaining cases.

Classic Composite Algorithms Figure 4.8 shows the effects of the classic composite algorithms, designated here by BBNR→CRR (diamond), RENN→ICFR (square) and RENN→RCR (triangle). We see a much greater spread in the effects on classifier accuracy: sometimes it improves; more often, it does not. BBNR→CRR does the least harm and has the narrowest spread (mean -1.54%, standard deviation 3.37) compared to RENN→RCR (mean -4.25%, standard deviation 6.85) and RENN→ICFR (mean -5.9%, standard deviation 6.64), but it is also the ones that deletes the least — 59.8% compared to 78.41% and 88.71%. BBNR→CRR improves accuracy in nine of the case bases; RENN→ICFR and RENN→RCR do the same in six and seven of the case bases respectively, and their decreases in accuracy tend to be larger than BBNR→CRR's. Again, it is the Lung Cancer and Lenses case bases that see the greatest decreases in accuracy. This seems to parallel the results about noise reduction from Figure 4.6.

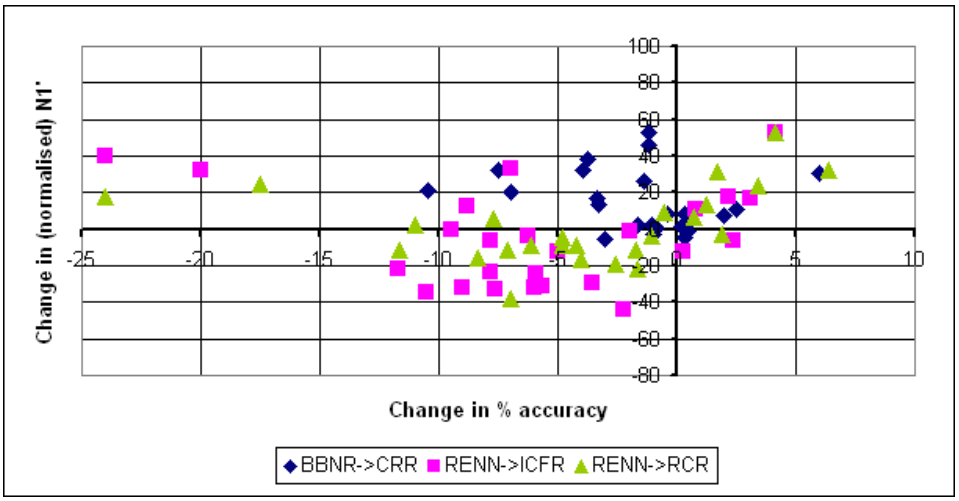
Of course, these composites run redundancy reduction after noise reduction, and we find that the redundancy reduction is having a substantial effect. On its own, BBNR deletes 23.08% on average, but BBNR→CRR deletes 59.38% (36.3 more). RENN deletes 23.17% on average, but RENN→RCR deletes 88.71% (65.54 more) and RENN→ICFR deletes 78.41% (55.24 more).

The BBNR→CRR composite seems to have a clear advantage over its constituents, BBNR and CRR. BBNR→CRR deletes on average about the same amount as the sum of the amount deleted by BBNR alone and CRR alone. But the decrease in mean accuracy is less than the sum of the decreases caused by BBNR alone and CRR alone. In the cases of RENN→ICFR and RENN→RCR, the percentages of cases deleted by the composites exceed the amounts deleted by their constituents but, unless these very high levels of deletion are desirable, it might be better to use their redundancy constituents alone, since these still delete quite a lot of cases but with a smaller decrease in mean accuracy.

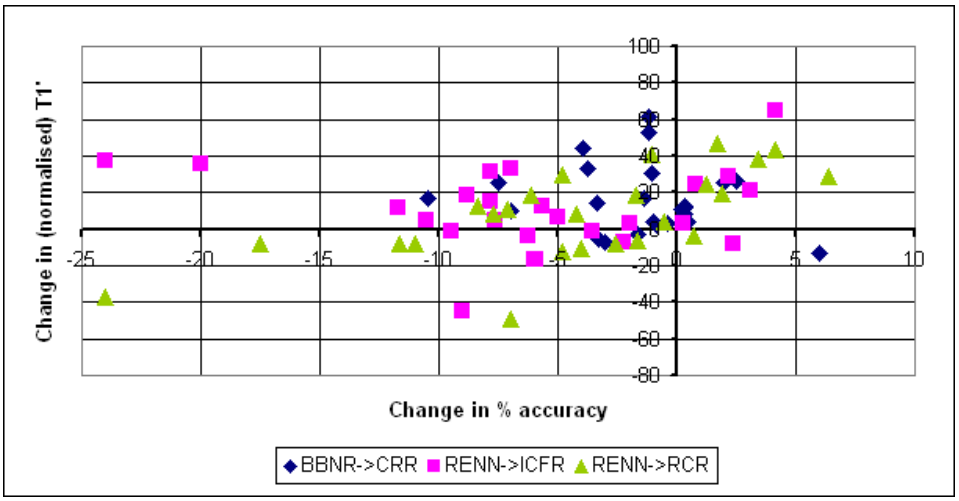
BBNR→CRR almost always lowers complexity: F_3' improves by 44 (standard deviation 35), N_1' by 15 (17) and T_1' by 16 (18). And it achieves these results with the least damage to accuracy. Results are more mixed for the other two composites. They both make improvements to F_3' of nearly 50 (standard deviation just over 30) and to T_1' of about 10 (standard deviation just over 20). But according to N_1', RENN→ICFR makes complexity worse (-5 with standard deviation 26) and RENN→RCR improves it but only by 1 (standard deviation 20). Although BBNR→CRR and RENN→ICFR are more similar in their redundancy reduction phases (both aim to delete 'interior' cases), RENN→ICFR and RENN→RCR have the same initial noise reduction phase, and this seems to make their behaviour more similar in these results. Again, unless RENN's high levels of deletion are especially desirable, it may be better to avoid it in favour of BBNR→CRR, or just ICFR



(a) F_3': Maximum Attribute Efficiency



(b) N_1': Fraction of Instances on a Boundary



(c) T_1': Fraction of Maximum Covering Spheres

Figure 4.8: Changes in accuracy and complexity: composite algorithms

or RCR on their own.

4.6 Conclusions

In this chapter, we have looked at a range of complexity measures. We have looked at some situations where there could be problems with these measures, and we have presented ways to handle these problems. Additionally, we have extended the measures so that they can deal with case bases with more than two classes, and with symbolic attributes where possible.

We ran these measures on our twenty-five datasets, and examined their results. We compared the measure results to each other, and looked at how well they correlated. Some of the measures had much in common, while others were very different. We found some cases where measures that we expected to be similar were not, and vice versa. The fact that some of the measures have an overlap with each other shows us that we might not need all of them to describe a dataset well.

We then went on to correlate the measure results with the classification results of a number of commonly-used classifiers. This was to show whether or not the measures were predictive of classifier accuracy. We found that many of the measures correlated well with the classifiers, with a number of the measures correlating extremely well with all of the them. This indicated that these measures could be useful in predicting classifier accuracy.

Finally, we looked at how maintenance algorithms affect the measures. We chose three of the measures to analyse in detail and we computed the measures on the twenty-five held-out datasets. We then ran each of the five atomic and three classic composite maintenance algorithms on the datasets and recorded the percentage of cases deleted, the change in accuracy, and the changes in the three complexity measures that we had selected. We analysed the results and found in particular that RENN may be too aggressive. It often deletes a large proportion of the case base, but sometimes at the price of large decreases in classifier accuracy. At the same time, complexity often increases. This suggests that RENN may be over-simplifying class boundaries. Composite algorithms that use RENN as their initial phase are affected in a similar fashion. Indeed, the problems are compounded, whether the subsequent redundancy reduction phase targets interior cases (as with ICFR) or boundary cases (RCR). The implications are that RENN needs much closer investigation. If it is important to delete a lot of cases, it may be better to use the ICFR or RCR redundancy reduction algorithms alone: fewer cases will be deleted, but the damage to accuracy may be lower.

We have found that BBNR may be doing a better job at identifying noise than RENN. It deletes as much on its own as RENN does, with less damage to accuracy.

In the previous chapter, we saw that different maintenance algorithms do well for different case bases. We now go on to ask the question whether or not the measures that we have looked at in this chapter could be used to predict a good maintenance algorithm for a case base. In the next chapter we will examine this question and try to choose a maintenance algorithm to use based on the information given by the complexity measures.

Chapter 5

Case Base Maintenance as Meta-Case-Based Classification

5.1 Introduction

We have shown in Chapter 2 that different maintenance algorithms work well for different case bases. There are many algorithms to choose from and we have increased that choice with our MACE approach, which produces an infinite amount of algorithms that can be used for maintenance. Additionally, a case base may need to use different algorithms at different times in its life cycle. For example, a case base developed from existing data may require noise removal in its early stages to make it cleaner, but redundancy removal as it matures and collects more experience. This presents us with a new problem: how do we choose a good maintenance algorithm to use for a given case base at a given time?

Imagine that we could collect experience across the CBR community to say which algorithm or algorithms worked well for a case base at a particular stage in its life-cycle. We could combine this experience to build a model and then use this model to make future maintenance predictions. In this chapter we set out to build such a model and use it to help us to solve the problem of choosing which maintenance algorithm to use for a particular case base.

In developing a model we have a number of options available to us. Firstly, we have the choice of whether to formulate the problem as one of regression or of classification. We could run different maintenance algorithms on case bases and measure values such as the percentage of deletion and the difference between the original case base classification accuracy and the accuracy of the case base after maintenance. If we stored these values we could then use regression to predict deletion and accuracy values for a given maintenance algorithm and case base. Alternatively, if we could make a judgement about which algorithm works best for a given case base, we could run different algorithms on different case bases and pick the best algorithm for each. We could then use classification to predict which algorithm is best to use for a given case base based on the algorithms that have worked well for other case bases.

Another choice is what type of system to use to make these predictions. We could use any classification or regression system, such as a neural net, a rule-based system, a decision tree or a CBR system.

Our goal in this chapter is to establish the feasibility of using a model to choose a maintenance algorithm for a given case base. It would not be possible for us to test all of

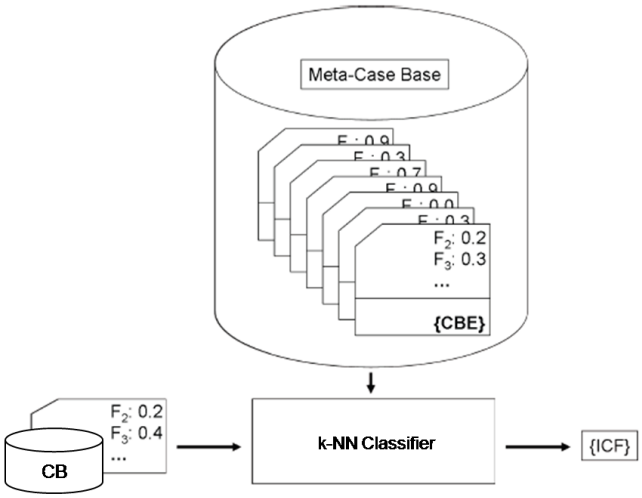


Figure 5.1: Maintenance algorithm selection as meta-case-based classification

the options that we have mentioned. Rather than comparing different ways of building this model, we chose to build one that kept closely in line with the rest of our work: case-based classification. We call this meta-case-based classification since we are performing case-based classification for a case-based reasoning task, namely case base maintenance.

In the next section we will describe our meta-case-based classification approach for case base maintenance. We then go on to describe our experiments and results. Following this we present a discussion of previous work in the area of meta-reasoning, and our conclusions.

5.2 Maintenance Algorithm Selection as Meta-Case-Based Classification

To perform meta-case-based classification, we need to assemble a case base to use for predicting maintenance algorithms. This is a case base whose cases describe other case bases (Figure 5.1). We call such a case base a *meta-case base*, and we refer to the cases it contains as *meta-cases*. Each meta-case has a *meta-problem description* and a *meta-solution*. The meta-problem description is made up of the values for *meta-attributes*. A meta-case's meta-solution contains the name or names of the best maintenance algorithms (of those tested) for the case base that this meta-case describes.

5.2.1 Meta-Problem Description

To create meta-attributes, we need to find a way to describe a case base as a series of attribute-value pairs. In the previous chapter, we examined seventeen measures that compute the complexity of a case base. These measures provide a description of the case base, so we can therefore use them as our meta-attributes. However, five of these measures cannot be computed for either multi-class classification datasets or for datasets with symbolic-valued attributes (or both). These measures are F_1, L_1, L_2, L_3 and N_4. We disregard these for use as meta-attributes in this chapter since both multi-class datasets and symbolic-valued attributes are common in CBR.

It is possible that not all of the measures may be useful as predictors of the best maintenance algorithm to use. Some of the measures may compute very similar properties of the case base, and so all of the measures may not be necessary to be used as meta-attributes. Additionally, there may be new measures developed in the future that better capture properties of the case base that are predictive of a good maintenance algorithm for that case base. We therefore present an approach in this chapter that can be used with any set of measures chosen by the person wishing to maintain their case base.

If we want to pick the absolute best combination of measures to use as meta-attributes, we would need to perform experiments with every different possible combination of measures. However, with a large number of measures this is not a practical solution. If we tested all combinations of the seventeen measures that we have looked at, we would have over two million different combinations of meta-attributes to test to find the best combination. This brute-force way of testing is not very efficient. Instead, we look at feature selection as a way to choose a good set of measures to use.

There has been much research done in the area of feature selection [2, 31, 50]. There are three types of feature selection algorithms: wrapper, filter and embedded [31]. Wrappers use a model to evaluate the predictive power of subsets of the attributes and choose the best attribute subset according to this model. Filters instead rely on general characteristics of the data to choose and evaluate a subset. Examples of these characteristics include correlation between attributes and mutual information. Embedded feature selection methods are similar to wrappers in that they use a model to evaluate a set of attributes. However, with embedded methods, the feature selection is built into the learning process itself, while wrappers perform feature selection as a preprocessing step before learning takes place.

There are also three types of search strategies used by feature selection algorithms: complete, sequential and random. Complete search guarantees to find the optimal attribute subset. Sequential search gives up completeness and therefore may miss optimal subsets, but is faster than a complete search algorithm. Random search starts with a random subset and either continues with sequential search by adding or removing attributes, or generates another random subset and retains the better subset.

Since the predictive accuracy of a k-NN classifier is a suitable criterion to use to evaluate attribute subsets, we use a wrapper algorithm. We use a best-first search algorithm, which picks the most promising attribute at each point. It begins with an empty set and tests the accuracy of the classifier using each measure as a single attribute separately. It picks the best attribute to use, and adds it to the empty set of selected attributes. It then goes on to add each of the remaining attributes to the selected set and repeats the process, either choosing another attribute to add or stopping if the addition of any other attribute would not improve accuracy.

5.2.2 Meta-Solution

The best case base maintenance algorithm or algorithms for a case base are taken to be the meta-solution in the corresponding meta-case. In this section we look at how to choose which maintenance algorithm is considered to be the best for a case base.

We have already acknowledged that, because of our MACE approach, there is an infinite number of maintenance algorithms which can be used to maintain a case base. However, since it is not possible to run an infinite number of maintenance algorithms on a case base to see which one works best, we need to choose a subset of candidate algorithms on which we base our predictions.

The set of candidate algorithms depends on what the person maintaining the case base hopes to achieve. If the focus is on deleting noise, the candidates might only contain noise reduction algorithms such as RENN and BBNR; if the focus is on aggressive deletion, the candidates may be composites like RENN→RCR; for more careful and conservative deletion, candidates may be composites like BBNR→CRR.

Once a set of candidate algorithms is chosen, each of these algorithms is run on the case base. We then need to decide which of the algorithms is the winner so that this can become the meta-solution. We have already discussed the problem of deciding on the best algorithm since an algorithm that does well in maintaining or improving accuracy may not perform much deletion, and an algorithm that deletes a large percentage of the case base may not maintain good accuracy. We showed that the harmonic mean of the two, as shown in Equation 3.1, provides a good balance between the two. Therefore a good choice for the meta-solution would be a set that contains the names of the maintenance algorithms whose resulting case bases give the highest harmonic mean value. Alternatively, a weighted harmonic mean can be used to bias the choice towards accuracy or deletion if either one was more of a priority for the person maintaining the case base.

5.2.3 Performing Meta-Case-Based Classification

Assume that we have taken a sample of existing case bases. We have turned each one into a meta-case. Specifically, for each case base in the sample, we have computed the values of the dataset complexity measures, which become the values of candidate meta-attributes; we have run a set of candidate algorithms and the meta-solution is the set that contains the case base maintenance algorithm or algorithms with the highest harmonic mean. Then, on the meta-case base, we have run a feature selection algorithm to decide which of the candidate meta-attributes should be used by this classifier. Now we are ready to use this meta-case base for decision-support.

When we are presented with a case base that requires maintenance, we turn this case base into a query for our meta-case-based classifier. Firstly, we compute the values of the dataset complexity measures on the case base. Obviously, we only need to compute the values of those measures that were selected by feature selection. This gives us the values of the meta-attributes of the query. We then use our meta-case base to classify the query using k-NN. Since all of the meta-attributes are numeric-valued, we compute similarity using Equation 2.3 for each meta-attribute.

The result of the classification is the maintenance algorithm which the meta-case-based classifier considers to be the best CBM algorithm to use for that query case base. Since some of the solutions contain more than one winning algorithm, we are faced with the problem of multi-label classification. While there are a number of different approaches to deal with multi-label classification [90], we chose to use a very simple approach. After finding the k-NN, we counted the number of votes that each winning algorithm got over the k cases. We then chose the algorithm with the most votes to be the winner. If there was a tie for the most votes, we chose one from the tied algorithms at random.

5.3 Evaluation

5.3.1 Holdout Experiments

Experiment Objective

The objective of this experiment is to test our meta-case-based classification system to see if it is competitive with the classic composite algorithms.

Experiment Methodology

For our evaluation, we used the twenty-five datasets which we described earlier. We firstly divided the datasets up into their 60-20-20 splits that we previously created. We performed holdout validation, using the 60% split as our training set, the first 20% as a cross-validation set and the second 20% as a test set. We report all results as averages over the ten splits that we previously created.

Each of the twenty-five training sets becomes a meta-case in a meta-case base. On each training set, we compute the twelve complexity measures, and these are the values of the candidate meta-attributes. We run each of the candidate maintenance algorithms which, in these experiments, are RENN→ICFR, RENN→RCR and BBNR→CRR, on the training set. We record the percentage of cases deleted. We use the cross-validation set to compute the accuracy of the training set after maintenance by each of the algorithms. In other words, we take each case from the cross-validation set and treat it as a query. We present it to a k-NN classifier that uses the training set as its case base. We compute the percentage of the cross-validation set that the k-NN classifier correctly predicts. For each of the maintenance algorithms, we can now compute the harmonic mean of the percentage of cases deleted and the post-maintenance classifier accuracy on the cross-validation set. The algorithm or algorithms with highest harmonic mean are the best algorithms to use on this training set, and therefore a set containing these algorithms is the meta-solution for this meta-case. We do this for each of the twenty-five training sets to give us twenty-five meta-cases in our meta-case base.

At this stage, each meta-case has twelve candidate meta-attributes, one for each of the complexity measures. We run the feature selection algorithm on the meta-case base to decide which of the candidate meta-attributes to use in the rest of the experiment.

We apply a similar process to each of the twenty-five test sets. The goal here is to create queries that we can present to the meta-case-based classifier. It is important to keep in mind that the queries to this classifier are not individual cases; they are case bases that are to undergo maintenance (see Figure 5.1). For each test set, we compute the values of the complexity measures. The only measures we need are those that were selected by the feature selection algorithm previously. These become the values of the meta-attributes of the query.

We need to know the best maintenance algorithm to use on each test set, to act as the true class. So we run each of the candidate maintenance algorithms on the test set. We note the best algorithm or algorithms, i.e. those that have the highest harmonic mean of the percentage of cases deleted and post-maintenance accuracy on the cross-validation set.

At this point, we have a meta-case base of twenty-five meta-cases, and a meta-test set of twenty-five case bases that are to undergo maintenance. We take each case base in the meta-test set in turn. We present it to the meta-case-based classifier (henceforth referred

Algorithm	Error(%)	Harmonic Mean	Accuracy(%)	Deletion(%)
Best Possible	0	76.02	71.68	85.09
RENN→RCR	24.4	74.49	68.87	87.50
Meta-CBR	43.2	69.79	69.53	78.21
RENN→ICFR	79.2	62.29	69.63	64.54
BBNR→CRR	88.4	60.41	72.64	57.11

Table 5.1: Results of Classic Composites and Meta-CBR using Holdout Validation

to as the meta-CBR system), which uses 3-NN to predict an algorithm in the way described in Section 5.2.

In the experiment, we measure two things. We record the percentage of the meta-test set that the meta-CBR system incorrectly predicts–the error rate. A correct prediction is one in which the algorithm that the meta-CBR system predicts is the best algorithm, or, in general, it is one in which one of the algorithms that the meta-CBR system predicts is one of the best algorithms.

However, prediction error is not the whole story. Even if the meta-CBR system fails to choose the best algorithm (or one of the joint best algorithms), it may choose one that, when run, has a performance not too far below that of the best algorithm. So for each of the twenty-five case bases in the meta-test set, we run the meta-CBR system's predicted algorithm and record the percentage of cases deleted, the post-maintenance accuracy on the cross-validation set and their harmonic mean.

Results

The results are shown in Table 5.1. The Table compares the meta-CBR system to four other options. One option is to always run the RENN→ICFR algorithm, irrespective of the characteristics of the query case base. Similarly, we can instead always run RENN→RCR or BBNR→CRR. The other option (Best Possible) is a benchmark figure, which supposes we have an oracle available enabling us to always choose the best algorithm.

We see that RENN→RCR is one of the best algorithms in 75.6% of meta-cases and so always choosing RENN→RCR gives an error rate of 24.4%. RENN→ICFR and BBNR→CRR are only rarely among the best algorithms and so always choosing them gives quite high error. Unfortunately, in terms of prediction error, in this experiment meta-CBR is performing poorly: it gets 43.2% of predictions wrong. But, as we said, this is not the whole story.

The results in Table 5.1 show that the highest possible harmonic mean value if we chose the best algorithm every time is 76.02. The algorithm that comes closest to this is RENN→RCR, with a harmonic mean value of 74.49. Our meta-CBR classification system results in a harmonic mean value of 69.79, which is the next highest value after RENN→RCR. What the mean hides is that our meta-CBR system results in case bases which, on average, exhibit post-maintenance classification accuracy that is slightly higher than RENN→RCR's, 69.53% compared to 68.87%. But, RENN→RCR deletes over 9% more cases from the case bases on average, and this is why it ends up with the higher harmonic mean.

Always choosing BBNR→CRR results in the highest classification accuracy (72.64%), higher even than choosing the algorithm with the best possible harmonic mean, but this is unsurprising given how conservative BBNR→CRR is: it deletes by far the lowest proposition

of cases. Always choosing RENN→ICFR also results in a conservative maintenance strategy (although less conservative than BBNR→CRR), hence its accuracy is higher than meta-CBR's too. But, because they delete a lower proportion of the case base, RENN→ICFR and BBNR→CRR have lower harmonic means than meta-CBR.

Although our meta-CBR system does not have a higher harmonic mean than RENN→RCR, it has a higher harmonic mean than RENN→ICFR and BBNR→CRR. Additionally, it maintains a good level of accuracy at just 2% lower than the best possible on average. This is a promising result for our meta-CBR system.

There are reasons to believe that these results may not be wholly representative. Some of the case bases that we present as queries are very small. They comprise just 20% of their full datasets, which may themselves be small. Some of them may even end up containing cases all of the same class. A first problem is that they may not contain enough cases, or enough different-class cases, for the dataset complexity measures to be informative. For example, if all cases are of the same class, then all complexity measures should produce their lowest values. A second problem is that, if most or all of the cases are of the same class, then this will favour RENN→RCR's aggressive redundancy reduction, and we are seeing this in the results Table 5.1. Accordingly, we decided to complement the holdout experiments with leave-one-out experiments, in which the case bases that we present as queries are larger and much less likely to contain cases all of the same class.

5.3.2 Leave-One-Out Experiments

Experiment Objective

In this experiment we aim to improve the performance of our meta-case-based classification system from the previous experiment by using bigger test datasets which should be more representative of the domain.

Experiment Methodology

In our leave-one-out experiments, we use the same splits of the twenty-five datasets. We use exactly the same meta-case bases as before, constructed from the 60% training splits and having meta-solutions chosen with the assistance of the 20% cross-validation splits. We do not need a separate meta-test set, so we can discard the 20% test splits.

In a fashion akin to classic leave-one-out cross-validation, we remove one meta-case from the meta-case base.We present this meta-case as a query to the meta-CBR system, which uses the remaining twenty-four meta-cases. We then return it to the case base. We repeat, taking each meta-case in turn.

The advantage is that each query case base is now larger, since it was built from 60% of the original dataset, so it is more likely to give a good description of the domain. It is also less likely that it contains only cases of the same class.

Results

The results are shown in Table 5.2. In these experiments, RENN→RCR is less often one of the best algorithms and so always choosing it is not such a good strategy, resulting in 35.2% error. By contrast, the performance of the meta-CBR system is much improved. Its error, which was 43.2% in the holdout experiment, has fallen to 26.8%.

Algorithm	Error(%)	Harmonic Mean	Accuracy(%)	Deletion(%)
Best Possible	0	82.41	79.39	87.65
Meta-CBR	26.8	80.57	76.81	88.54
RENN→RCR	35.2	80.30	76.24	88.71
RENN→ICFR	70.0	74.41	74.69	78.38
BBNR→CRR	94.4	62.56	77.96	54.88

Table 5.2: Results of Classic Composites and Meta-CBR using Leave-One-Out Cross-Validation

In these experiments, the highest possible harmonic mean value is 82.41. Our meta-CBR system comes closest to this with a harmonic mean value of 80.57, followed closely by RENN→RCR with a value of 80.30. We also perform better than RENN→ICFR and BBNR→CRR, which have harmonic mean values of 74.41 and 62.56 respectively. Meta-CBR has post-maintenance accuracy that is just 2.6% off that achieved by using an oracle and second only to highly conservative BBNR→CRR. It deletes a slightly greater proportion of the case base than the oracle achieves, and second only to very aggressive RENN→RCR. This is a big improvement over our holdout experiments. This shows us that our complexity measures are more reliable and more predictive of the best maintenance algorithm to use when they are computed for case bases which have good coverage of the domain.

5.4 Previous Work

The idea of meta-reasoning?-reasoning about reasoning-?has been explored in a number of different areas. For example, there is an amount of research into meta-planning [36, 87, 95]. Within CBR, Fox and Leake [24, 23] build a system that reasons about CBR: it uses introspection to improve indexing in a case-based planning system. They use a model to evaluate the performance of a case-based planner in comparison to its expected ideal performance, and they use this model to suggest a repair when the planner does not perform as it should.

Within machine learning and data mining, there has been an amount of research into meta-learning, in particular training meta-classifiers that predict the best classifier to use on a dataset. For example, Lindner and Studer take an approach similar to our own: they use a number of dataset complexity measures and other statistical measures as their meta-attributes [15]. But their system selects classifiers, not maintenance algorithms. Peng et al. also use a k-NN meta-classifier to choose between a number of classification algorithms (rather than maintenance algorithms) [66]. For their meta-attributes, they build decision trees on datasets and compute characteristics of the decision trees, such as the length of the longest branch. Bensusan et al., by contrast, use a representation of the decision tree itself as the description of the dataset [4]. Pfahringer et al. take an approach that they call landmarking, whereby the meta-attributes come from measuring the performance of simple classifiers on the datasets [67]. Of course, some of our dataset complexity measures also have this character, e.g. N_3, which is the error rate of a 1-NN classifier.

Within CBR also, Recio-García et al. use CBR to choose a template from a case base of templates to aid the design and implementation of CBR systems [73]. In particular, they describe a case-based recommender that recommends recommender system templates from

a case base of twelve templates. Van Setten [91] compares decision-support systems built using CBR and built using decision trees that help the user to design a hybrid recommender system. Leake et al [44] use CBR to learn adaptation knowledge from a case base. This adaptation knowledge is stored as adaptation cases, which are then in turn used to help the CBR system in the future.

5.5 Conclusions

In this chapter, we have presented our meta-case-based classifier that can form the basis of a decision-support system that assists case base administrators. It can choose a case base maintenance algorithm that is suited to the characteristics of a given case base at a given point in its lifetime. A key design decision was how to characterize case bases: both those that are described by meta-cases in our meta-case base, and incoming query case bases that are to undergo maintenance. We have used a number of measures of dataset complexity, which attempt to characterize the geometry of the case bases. The actual meta-attributes that we use are chosen by a feature selection algorithm from a large set of complexity measures.

The results of our first experiment, which used a holdout method, were promising. The meta-CBR system picked maintenance algorithms that produced case bases that achieved slightly higher classification accuracy than always picking the RENN→RCR algorithm. However, always choosing RENN→RCR resulted in a much larger proportion of cases being deleted, and hence a higher harmonic mean. Similarly, compared with selecting the best possible maintenance algorithm using an oracle, the meta-CBR system picked algorithms that produced case bases with comparable classification accuracy, but again was too conservative in the proportion of cases deleted.

Our second set of experiments used a leave-one-out method, enabling us to make better use of the available data. In particular, query case bases were three times larger than in the first experiment, and were much less likely to contain cases most or all of which were of the same class. In this experiment, the meta-CBR system outperformed the approach of always picking the same algorithm. For example, the algorithms chosen by meta-CBR deleted a slightly larger proportion of cases, and resulted in case bases with slightly higher classification accuracy, than always using RENN→RCR. This makes the point that for meta-CBR to work, the case bases need to be large enough, and representative enough, that the dataset complexity measures are informative. In practice, of course, this is likely to be so, otherwise the case base administrator would not have deemed case base maintenance to be worthwhile.

Our experiments show that our meta-case-based classification system works well in selecting a good maintenance algorithm for a given case base. However, current maintenance algorithms delete whole subsets of the cases in a case base. But as soon as so much as one case is deleted, it no longer follows that the same algorithm is the best to decide on the next case to delete. In the next chapter, we will present incremental versions of the case base maintenance algorithms, which recommend the deletion of only one case at a time. We also present incremental versions of some of the dataset complexity measures, whose values can be rapidly re-calculated for a case base following deletion of a case. This allows us to build an incremental meta-CBR system, which repeatedly selects the best incremental case base maintenance algorithm, deletes one case, and then returns to the meta-CBR system to select an algorithm to delete the next case, and so on until the system predicts there to

be no advantage in deleting a further case.

Chapter 6

Incremental Meta-Case-Based Classification

6.1 Introduction

In the previous chapter, we discussed the need to be able to automatically choose which maintenance algorithm is best for a particular case base at a given time. In this chapter, we look at this problem in more fine-grained detail. While a maintenance algorithm may be the best one to use for a case base at a given time, it may no longer be the best choice once a number of the cases have been deleted. As the case base changes, different algorithms may be able to play a part in editing the case base.

An example of this already in use comes from the composite maintenance algorithms. These algorithms run a noise reduction algorithm and then change to a redundancy reduction algorithm once the noise has been removed. It is not unreasonable to imagine extending this to switch back and forth between the two algorithms as maintenance progresses. Noise would be removed as normal by the noise reduction algorithm. Then as redundancy reduction progresses it may reveal noise that was not detected among the redundant cases. Switching back and forth between the algorithms would allow this noise to be removed, giving a better edited case base than when the algorithms are run sequentially to completion.

In this chapter, we go on to extend this idea further. We modify each maintenance algorithm to delete cases incrementally so that, at any point in time, the algorithm can suggest the single best case for deletion. We then design a new incremental meta-case-based classification system to use these incremental maintenance algorithms. To choose which case is best for deletion, the incremental meta-case-based classifier computes the case base measures and chooses the best incremental algorithm to use for deletion. It then deletes one case and repeats the process until the algorithm chosen has no case for deletion. In doing this we also develop incremental versions of the case base measures because it would not be feasible to recompute every measure 'from scratch' each time that the case base changes due to the deletion of a case.

The incremental case base maintenance algorithms can also be useful to a knowledge engineer who may have already chosen which algorithm he wants to use. If he runs the original, 'batch' version of the maintenance algorithm, it will suggest all cases that it considers to be noisy and/or redundant. The knowledge engineer has no way of knowing which of these cases the algorithm considers to be the most harmful, so he can either delete all of

the cases that the algorithm suggests, or none of them.

By suggesting a single case at a time for deletion, an incremental algorithm gives the knowledge engineer more control over the deletion process. After each time he runs the algorithm, he has one decision to make: whether to delete the case or to keep it. This is much less overwhelming for the knowledge engineer than being faced with all of the cases suggested for deletion at once.

It would also be possible to prioritise the cases being suggested for deletion. Each time that it is run, the algorithm will choose the most harmful case for deletion. If the algorithm is then re-run, it will choose the next most harmful case, and so on. In doing this, even if the knowledge engineer is shown all of the cases to be deleted at once, he can see how harmful the algorithm considers each of the cases to be, and can take this into account when deciding which cases to delete and keep.

In the next section we describe the modifications that we have made to each of the case base maintenance algorithms to make them incremental. Following that we describe the incremental case base measures, i.e. the changes that we made to the way we compute the measures to allow them to adjust to changes in the case base. We use the incremental algorithms and measures to develop our incremental meta-case-based classifier. We then go on to describe our experiments and results and finally, we present our conclusions.

6.2 Incremental Case Base Maintenance Algorithms

We modified each of the five atomic case base maintenance algorithms to be able to suggest one case at a time for deletion. It would be possible to develop ways to also allow the composite algorithms to delete incrementally, but in our work we chose to only work with the algorithms that we use for our meta-case-based classification system. In this section we now present the changes to the original atomic algorithms to make them incremental. However, before we do this, we present an incremental version of the competence model, to ensure that coverage, reachability and liability sets are kept updated as cases are deleted.

6.2.1 Incremental Competence Model

In the competence model that we compute, we store the coverage, reachability and liability sets for each case in the case base (since the maintenance algorithms that we are using make no use of the dissimilarity sets, we do not consider them here). When we remove a case x from the case base, we firstly remove its sets from the competence model. We then need to update the other cases' sets because x will no longer contribute towards classifying or misclassifying any cases.

When we compute our original competence model, we save the information about which cases classify or misclassify each other case in the case base. We also store all of the nearest neighbours of each case in order of decreasing similarity, so that computing the new cases that classify or misclassify a case does not take long.

When we are removing x from the case base, we go through each case, x', in turn. We compute the cases that will classify or misclassify x' when x is not in the case base. We then compare these to the cases that classified or misclassified x' previously. If the cases are the same, then we do not need to make any changes for x'. However, if there are changes to the 'classifies' or 'misclassifies' sets, then we need to make a number of adjustments to our coverage, reachability and/or liability sets.

There are four possible changes that can occur:

1. A case x'' that did not previously classify x' now does.
2. A case x'' that previously classified x' no longer does (may include x).
3. A case x'' that did not previously misclassify x' now does.
4. A case x'' that previously misclassified x' no longer does (may include x).

Depending on the definition of 'classifies' that we are using, some of these situations may never arise. However, we deal with all possibilities so that our incremental competence model can be used with any definition of 'classifies'. We now show how the coverage, reachability and liability sets change in each of the four situations.

A case x'' that did not previously classify x' now does. We add x'' to x''s reachability set since it classifies x'. We also add x' to x'''s coverage set, which is the set of cases that x'' can classify.

A case x'' that previously classified x' no longer does. We do the opposite to what we did in the previous situation. We remove x'' from x''s reachability set, and we remove x' from x'''s coverage set.

A case x'' that did not previously misclassify x' now does. We add x' to x'''s liability set, since x'' contributes to the misclassification of x'.

A case x'' that previously misclassified x' no longer does. We remove x' from x'''s liability set.

We go through the case base and make these changes anywhere that the 'classifies' or 'misclassifies' sets have changed. When we are finished, our new competence model will no longer contain x or any of the contributions that it made. We can then use this competence model in our incremental maintenance algorithms.

6.2.2 RENN$_i$

RENN removes noise by deleting cases that have been misclassified by their nearest neighbours. To make this incremental, we go through the case base and flag all of the cases that would be deleted by the original algorithm. For each case, we compute how strongly it is misclassified. We do this by looking at its similarity to each of its k-NN. We add up its similarity to cases of a different class to the case itself, and subtract from this its similarity to cases of its own class. This gives us a measure of how badly each case is misclassified. We then order the flagged cases by how strongly they are misclassified, and choose the case that has the highest value as the case to be deleted by RENN$_i$. If there are a number of cases tied for the highest value, we choose one of these cases at random. We can see this algorithm in Algorithm 7.

6.2.3 BBNR$_i$

BBNR works by removing cases which contribute to the misclassification of other cases in the case base. It first builds a competence model to find the coverage and liability sets of each case in the case base. It then orders the cases by the size of their liability sets. It goes through the cases in this order, starting with the cases that do most harm. For each of these

Algorithm 7 Incremental Repeated Edited Nearest Neighbour Algorithm

$CB \leftarrow$ Case Base
$candidates \leftarrow \{\}$ (candidates for deletion)
$changes \leftarrow$ false
for each $x \in CB$ **do**
  if x does not agree with the majority of its NN **then**
    $candidates \leftarrow candidates \cup \{x\}$
    $classifySim(x) \leftarrow \sum_{x' \in k-NN(x)} Sim(x, x') if\, x_c = x'_c$
    $misclassifySim(x) \leftarrow \sum_{x' \in k-NN(x)} Sim(x, x') if\, x_c \neq x'_c$
    $misclassifyDiff\,(x) \leftarrow misclassifySim(x) - classifySim(x)$
    $changes \leftarrow$ true
  end if
end for
if *changes* **then**
  $candidatesSorted \leftarrow candidates$ sorted in descending order of $misclassifyDiff\,(x)$
  return first case in *candidatesSorted*
else
  return NULL
end if

cases in turn, it checks if all of the cases in its coverage set can be classified correctly without that case. If they can, the case is removed from the case base. It is relatively straightforward to make this algorithm incremental, because the cases are examined starting with the most harmful anyway. So the first case that would be deleted by the original BBNR algorithm becomes the case deleted by BBNR$_i$, as shown in Algorithm 8.

6.2.4 RCR$_i$

Rather than concentrating on which cases to choose for removal as RENN and BBNR do, RCR looks at which cases to keep for the edited case base. It then removes the cases covered by the cases that it is choosing to keep. In modifying the algorithm to make it incremental, we first order the cases by their relative coverage (Equation 2.8), as in the original algorithm. We then take each case in turn, starting with the case with the highest RC value. We remove the case from the original case base, and mark it to be kept as part of the new case base. If the case has cases in its coverage set that are not marked to be kept, we pick one of these at random for deletion. If not, we continue until we find a case to delete or until all cases are flagged to be kept, in which case RCR$_i$ has no case for deletion. The algorithm is presented in Algorithm 9.

6.2.5 ICFR$_i$

The original ICFR algorithm deletes a case that has a reachability set that is bigger than its coverage set, i.e. it has more cases classifying it than it itself classifies. To choose which case ICFR$_i$ deletes, we flag all cases whose reachability set is bigger than its coverage set. For each of these cases, we calculate the difference in size between its reachability and coverage sets. If there is a large difference, the case does not classify many cases in comparison to the number of cases that classify it. ICFR$_i$ deletes the case with the highest difference value, or

Algorithm 8 Incremental Blame-Based Noise Reduction Algorithm

```
CB ← Case Base
/* Build or adjust case base competence model */
for each x ∈ CB do
  CSet(x) ← Coverage Set of x
  LSet(x) ← Liability Set of x
end for
/* Remove noisy case */
CBSorted ← CB sorted in descending order of LSet(x) size
(for ties in LSet(x) size sort in ascending order of CSet(x) size)
x ← first case in CBSorted
while LSet(x) ≠ {} do
  CBSorted ← CBSorted − {x}
  misClassifiedFlag ← false
  for each x′ ∈ CSet(x) do
    if x′ cannot be correctly classified by CBSorted then
      misClassifiedFlag ← true
      break
    end if
  end for
  if misClassifiedFlag = false then
    return x
  else
    CBSorted ← CBSorted ∪ {x}
  end if
  x ← next case in CBSorted
end while
return NULL
```

if there are a number of cases tied with the highest value, it deletes one of these at random. This algorithm is shown in Algorithm 10.

6.2.6 CRR_i

The original CRR algorithm takes a similar approach to RCR by flagging cases to keep rather than to delete. It goes through the cases in increasing order of their coverage set size. For each case, it adds the case to the new case base and deletes the cases that it covers from the original case base. To make this algorithm incremental, we adopt a similar approach to that of RCR_i. Each time a case is flagged to be kept, if it has cases in its coverage set that are not themselves flagged to be kept, CRR_i chooses one of these cases at random to delete. Otherwise it continues flagging cases to be kept until it finds a case for deletion, or until it goes through all of the cases and there are none to be deleted, as shown in Algorithm 11.

Algorithm 9 Incremental Relative Coverage Algorithm

$CB \leftarrow$ Case Base
$CBNew \leftarrow \{\}$ (cases flagged to be kept)
while $CB \neq \{\}$ **do**
  /* *Build or adjust case base competence model* */
  for each $x \in CB$ **do**
    $CSet(x) \leftarrow$ Coverage Set of x
    $RSet(x) \leftarrow$ Reachability Set of x
    $RC(x) \leftarrow \sum_{x' \in CSet(x)} \frac{1}{|RSet(x')|}$
  end for
  /* *Remove redundant case* */
  $x \leftarrow$ case in CB with highest $RC(x)$ value
  $CBNew \leftarrow CBNew \cup \{x\}$
  $candidatesForDeletion \leftarrow CSet(x)$
  for each $x' \in candidatesForDeletion$ **do**
    if $x' \in CBNew$ **then**
      $candidatesForDeletion \leftarrow candidatesForDeletion - x'$
    end if
  end for
  if $candidatesForDeletion \neq \{\}$ **then**
    return random case from *candidatesForDeletion*
  end if
end while
return NULL

6.3 Incremental Computation of Case Base Measures

The case base measures that we use in our meta-case-based classification system are all designed to be computed once for a given case base. However, to make our meta-case-based classification system incremental, we need to adjust the measure values after each case deletion. Re-computing the measures 'from scratch' each time would be very expensive, especially for a large case base. Instead, we have developed incremental versions of the measures so that each time a case is deleted, the measure value is updated rather than re-computed. As in our earlier meta-case-based classification work, we only work with the twelve measures that can handle multi-class classification datasets and symbolic-valued attributes.

These incremental case base measures would also be useful in domains where case bases change regularly. Although our current work focuses on how to adjust the measures as cases are removed from the case base, it would also be possible to adjust the measures as new cases are added in.

We now present the adjustments that we have made to each of the measures to make their computation incremental.

Algorithm 10 Incremental Iterative Case Filtering Algorithm

```
CB ← Case Base
candidates ← {} (candidates for deletion)
/* Build or adjust case base competence model */
for each x ∈ CB do
   CSet(x) ← Coverage Set of x
   RSet(x) ← Reachability Set of x
end for
/* Remove redundant case */
changes ← false
for each x ∈ CB do
   if |RSet(x)| > |CSet(x)| then
      candidates ← candidates ∪ {x}
      sizeDifference(x) ← |RSet(x)| − |CSet(x)|
      changes ← true
   end if
end for
if changes then
   candidatesSorted ← candidates sorted in descending order of sizeDifference(x)
   return first case in candidatesSorted
else
   return NULL
end if
```

6.3.1 Measures of Overlap of Attribute Values

F'_{2i}: Incremental Volume of Overlap Region

Our F'_2 measure takes each attribute in turn and goes through all of the values that occur for that attribute. For each value, it looks at all of the cases that have that value for the attribute. If there are any two cases of different classes with the same attribute value, that attribute value is added into the overlap.

To be able to adjust the measure as a case is deleted, we need to keep track of the attribute values, and how many times each value occurs for each class. When we have a case to delete, we go through its attribute values and subtract one from the counts of these values for the class of the case. If removing the case causes an attribute value to now only occur in cases of one class, that attribute value can then be removed from the overlap.

The adjusted overlap value for the attribute is then normalised by the number of distinct attribute values (which may also have changed if the case being removed was the only case with a particular attribute value) and the product of the attribute measure values gives the measure value for the case base, as in F'_2.

F'_{3i} and F'_{4i}: Incremental Maximum and Collective Attribute Efficiency

Attribute efficiency is a count of the number of cases whose values for that attribute are outside of the overlap, normalised by the number of cases in the case base. Similarly to F'_{2i}, for our incremental version of attribute efficiency we need to store the different attribute values and how often they occur in cases of each class. When we remove a case from the

Algorithm 11 Incremental Conservative Redundancy Reduction Algorithm

```
CB ← Case Base
flaggedCases ← {} (cases flagged to be kept)
/* Build or adjust case base competence model */
for each x ∈ CB do
  CSet(x) ← Coverage Set of x
end for
/* Remove redundant case */
CBSorted ← CB sorted in ascending order of CSet(x) size
x ← first case in CBSorted
while CBSorted ≠ {} do
  flaggedCases ← flaggedCases ∪ {x}
  candidatesForDeletion ← CSet(x)
  for each x' ∈ candidatesForDeletion do
    if x' ∈ flaggedCases then
      candidatesForDeletion ← candidatesForDeletion − x'
    end if
  end for
  if candidatesForDeletion ≠ {} then
    return random case from candidatesForDeletion
  end if
end while
return NULL
```

case base, we adjust the counts according to the attribute values and the class of that case. If that case was the only one with a particular attribute value for its class, its removal may mean that the attribute value may now only occur in cases of one class. If this happens, this attribute value is no longer in the overlap, and cases with that attribute value are now added into the attribute efficiency count.

After the attribute efficiency values are updated, the maximum and collective attribute efficiency values are computed in the same way as for F_3' and F_4'.

6.3.2 Measures of Separability of Classes

N_{1i}': Incremental Fraction of Cases on a Boundary

To compute the fraction of cases on a class boundary, we build a minimum spanning tree and use this to count how many cases are connected to cases of a different class. The removal of any node from the MST will result in a tree that is no longer a MST itself. Therefore, every time we remove a case from the case base, we need to recompute the MST. In this sense, we do not have a true incremental version of this complexity measure.

In our earlier discussion of N_1', we pointed out that because MSTs are not unique, we would end up with different values of the measure depending on the ordering of the cases presented to the MST algorithm. To solve this, we shuffle the dataset ten times, compute the measure using each ordering and take the average of these as the final measure value.

However, we now need to recompute the MSTs each time that a case is deleted. It is not feasible to recompute the 10 MSTs every time. So we looked at what would happen if we ran it 5 times instead, since this would greatly improve the speed at which the measure

could be computed. We computed the measure values 10 times for each of the dataset splits. We then computed the average value for each dataset over the first 5 shuffles, and the average value over all 10 shuffles. We correlated the results using Pearson correlation between the two sets of figures. The correlation value between the two was 0.9993. This high correlation value shows us that the values obtained for the 5 shuffles are close to the values obtained for the 10 shuffles. This justifies the use of 5 shuffles rather than 10 to allow the measure to be able to be computed faster.

N_{2i}: Incremental Ratio of Average Intra/Inter Class Distance

To compute N_2, we calculate the average distance between each case and its nearest neighbour of the same class, and the average distance between each case and its nearest neighbour of a different class. We then compute the ratio as shown in Equation 4.12.

To modify this measure so that it can be updated incrementally, we store each case's nearest neighbour of the same and of a different class and the distance for each. We also store the sum of intra- and inter-class distances for the case base. When a case is then deleted, we first remove that case from the sums by subtracting its distance to its nearest neighbour of the same class from the intra-class distance total, and subtracting its distance to its nearest neighbour of a different class from the inter-class distance total. We then update those cases which have the deleted case as their nearest neighbour of the same or a different class. For each of those cases, we compute its new nearest neighbour to replace the deleted case. We then update the intra- or inter-class sum (depending on whether the neighbour being replaced is of the same or a different class) by subtracting the old distance and adding in the new distance instead.

We then use our updated totals to compute the ratio as shown in Equation 4.12.

N_{3i}: Incremental Error Rate of a 1-NN Classifier

N_3 computes the leave-one-out error rate of a 1-NN classifier on the case base. To update this measure as cases are deleted, we need to compute and store each case's nearest neighbour. We also count how many cases have been misclassified by their nearest neighbour and store this value. To get the error rate we divide this number by the number of cases in the case base.

When a case is removed, we make similar adjustments to those in N_{2i}. Firstly, we check if the case being removed had been misclassified by its nearest neighbour. If so, we subtract one from the misclassification count since this case will no longer be in the case base. We then find the cases which had the removed case as their nearest neighbour. For each of these, we calculate its new nearest neighbour and check if its classification has changed. If it had been correctly classified and is now being misclassified by its new nearest neighbour, we add one to the misclassification count. Conversely, if it had been misclassified and is now being classified correctly, we subtract one from the misclassification count. We then compute N_{3i} by dividing this count by the new case base size.

N'_{5i}: Incremental Separability Emphasis Measure

The separability emphasis measure is the product of N'_1 and N_2. To make it incremental, we compute it from the incremental versions of these two measures.

C_{1i}: Incremental Complexity Profile

To compute C_1, we need the K nearest neighbours of each case for some parameter K. We then compute the measure as shown in Equation 4.14. To modify this to allow it to update as cases are removed, we store these K-NN for each case. We also store each case's complexity profile value. When a case is deleted, we find any cases which have the removed case as one of their K-NN. For each of these cases, we update its K-NN, taking out the removed case and inserting the next nearest neighbour. We then recompute its complexity profile. We also remove the complexity profile of the removed case. We then compute C_{1i} using Equation 4.14 as before.

C_{2i}: Incremental Similarity-Weighted Complexity Profile

To create an incremental version of C_2, we do the same as we did for C_1 to update neighbours. The rest of the calculation then proceeds as it did before (Section 4.3.2).

6.3.3 Measures of Geometry, Topology and Density of Manifolds

T'_{1i}: Incremental Fraction of Maximum Covering Spheres

We compute T'_1 by computing the adherence subset of each case, x, which is a hypersphere of cases around x with all cases in the hypersphere of the same class as x, but with the next nearest neighbour outside the hypersphere being of a different class. Adherence subsets that are contained within the adherence subset of another case are then removed. The measure is given by the number of remaining adherence subsets required to represent the case base, normalised by the number of cases in the case base.

For an incremental version of this measure, we need to store each of the adherence subsets. In addition to this, we store the next nearest neighbour outside the adherence subset that is of a different class to the cases in the subset. Each time a case is removed from the case base, we first remove its adherence subset. We then find which adherence subsets contain this case, and we remove it from these subsets. Finally, we check if the removed case is the next nearest neighbour to any of the adherence subsets. If it is, we need to expand that adherence subset by adding the next nearest neighbours until we come to another neighbour of a different class. That neighbour will now be stored as the new next nearest neighbour of a different class.

After modifying the adherence subsets, we compute the measure as before.

T_{2i}: Incremental Number of Instances per Attribute

This measure is easily made incremental. As cases are removed, the number of instances goes down. To compute T_{2i}, we take this number and divide it by the number of attributes per case.

T_{3i}: Incremental Dataset Competence

To compute T_3, we use a competence model to compute the coverage of each case. We then find which cases share coverage, and divide these into competence groups. We then compute the coverage of each competence group, and we sum these values to get T_3, the competence of the case base.

To make this incremental, we work with the competence model. We create an incremental competence model that updates the coverage sets as each case is deleted. We then recompute the competence groups according to the new coverage sets, and we compute the competence of the dataset as before.

We have now presented the modifications that we have made to the case base maintenance algorithms and to the complexity measures to allow them to run incrementally. In the next section we will describe how we use these to put together our incremental meta-case-based classification system.

6.4 Incremental Meta-Case-Based Classification

Now that we have modified both the case base maintenance algorithms and the case base measures to work incrementally, we can build our incremental meta-case-based classification system. The approach is much the same as in the last chapter, except this time we make our maintenance decisions on a case-by-case basis. We build our meta-case base as before, containing meta-cases made up of a meta-problem description and meta-solution. Each meta-case corresponds to one training case base. The meta-problem description describes the case base using the case base measures we have described. The meta-solution consists of the best incremental case base maintenance algorithm or algorithms for that case base. Because the incremental CBM algorithms only delete one case, the best algorithm is chosen based on accuracy alone.

When we wish to perform maintenance on a case base, we turn this case base into a query as before. We run the incremental case base measures on the case base and store their values as the query meta-attributes. We then classify the query using our meta-case-base. For classification we use k-NN, with k set to 3. This will give us an incremental case base maintenance algorithm to run. We run this algorithm, delete the suggested case, and update the incremental measures. We keep doing this until the algorithm chosen by the incremental meta-case-based classification system has no case for deletion. This means that we may jump between different maintenance algorithms as we go through the deletion process. We hope that this will mean that the most suitable algorithm will be applied at every stage.

We now present our evaluation of the incremental meta-case-based classification system.

6.5 Evaluation

6.5.1 Experiment Objective

Our objective for this experiment is to test our incremental case base maintenance algorithms. In particular, we want to see if our incremental meta-CBR system outperforms the incremental atomic CBM algorithms.

6.5.2 Experimental Methodology

For our evaluation, we use the twenty-five datasets which we have used throughout this work. This time, we use 60-40 splits by combining the two 20% sets for each split to make up the 40%. We use the 60% for leave-one-out experiments, so we use the remaining 40% as a cross-validation set since we do not need a meta-test set.

Additionally, to give our system a bigger meta-case-base to work with, we took each of the 60% splits in turn and ran each of the five atomic case base maintenance algorithms on it. This gave us six case bases for each of the datasets, resulting in 150 case bases in total to work with for our meta-case-based classification system.

We prepare the meta-CBR system in a similar way to the previous chapter. We begin by computing the meta-solutions for the meta-cases. For each of the 150 case bases, we run the five incremental case base maintenance algorithms on the case base, remove the case suggested by the algorithm and compute the accuracy of each resulting case base in turn using the 40% cross-validation set. If more than one incremental maintenance algorithm ties with the highest accuracy, we include the tied algorithms as the best algorithms. One modification we make is that if all five algorithms tie with the highest accuracy, we leave that meta-case out of the meta-case base because it would not be helpful in making a decision as to the best algorithm to use. This modification takes away about half of the case bases for each split, leaving us with a meta-case base of between 70 and 80 meta-cases per split.

We then compute the twelve complexity measures for each of the remaining case bases. We run feature selection as before to find the best subset of the twelve measures to use as the meta-attributes.

To perform leave-one-out cross-validation, we remove each meta-case from the meta-case base in turn. We present it as a query to the meta-case base and perform 3-NN classification to choose an incremental case base maintenance algorithm to run. We run that algorithm to choose a case to delete from the original case base. We then remove that case and update the incremental case base measures to reflect the smaller case base. We repeat this until the algorithm chosen by the meta-case-based classification system chooses not to delete a case. We then record the accuracy of the resulting case base using our cross-validation set.

We also ran each of the incremental algorithms repeatedly until the algorithm chose to stop and recorded the accuracy of the resulting case bases. Finally, we ran a benchmark algorithm where at each point we ran all five incremental algorithms, removed the case chosen by each separately and tested each for accuracy so as to know which was the best to delete. We repeated this until the best accuracy was given by an algorithm that did not have any case to delete. This gave us a benchmark to aim for.

We did not record the percentage error for these experiments in the way we did for the experiments in the previous chapter. This is because there is no single winning algorithm for these experiments. The benchmark that we run can give us the best possible sequence of incremental algorithms to run, but once each of our other algorithms removes a different case at some point, this sequence is no longer relevant. As a result, it does not make sense to measure error here.

6.5.3 Results

Table 6.1 shows the results of the leave-one-out cross-validation described in the previous section. We can see that when we pick the best possible algorithm by accuracy at each stage of the incremental process, this produces much better results than any of the other incremental algorithms. It deletes almost 20% more than the next best algorithm, RCR_i, and is almost 20% more accurate also. This is a disappointing result for both the incremental versions of the original algorithms and for our incremental meta-case-based classification system. However, it does show that there is much scope for improvement.

We firstly compare the results of the incremental algorithms to those of the benchmark

Algorithm	Harmonic Mean	Accuracy	Deletion
Best Possible	78.22	81.56	75.13
RCR_i	59.28	62.66	56.24
Incremental Meta-CBR	59.14	65.99	53.57
CRR_i	58.42	63.25	54.27
$BBNR_i$	49.84	68.82	39.07
$RENN_i$	49.34	68.06	38.7
$ICFR_i$	46.25	70.98	34.3

Table 6.1: Results of Incremental Algorithms using Leave-One-Out Cross-Validation

algorithm. We can see that even the algorithm with the closest harmonic mean value, RCR_i, is still only at 59.28, compared with the benchmark value of 78.22. The story does not improve when we look at the accuracy and deletion percentages. $ICFR_i$ (70.98%) is the closest to the benchmark accuracy of 81.52%. RCR_i deletes 56.24%, in comparison to the benchmark value of 75.13%. This shows us that there is much room for further work in choosing which algorithm to use at any given point in the deletion process.

When we compare the results of our incremental meta-case-based classification system to the results of the other incremental maintenance algorithms, we see that it does quite well. RCR_i has a slightly higher harmonic mean value (59.28, compared with meta-CBR's 59.14). RCR_i deletes almost 3% more than meta-CBR on average, but RCR_i is just over 3% less accurate. Our system beats the other four incremental maintenance algorithms by their harmonic mean values. However, it deletes slightly less than CRR_i, and it is less accurate than the other three.

Finally, we look at the incremental versions of the atomic algorithms and how they perform in comparison to their original maintenance algorithms. RCR_i is the most aggressive, with lower accuracy than the others, which is not surprising. The two noise reduction algorithms produce quite similar results, with low deletion as expected and relatively high accuracy in comparison to the other algorithms. $ICFR_i$ gives the most surprising result. It has the lowest percentage of deletion of all of the algorithms, at 34.4%. This is unusual, given that the batch version of ICFR was one of the most aggressive algorithms. Its incremental version stops deleting much earlier than the original version. This also results in the incremental version being more accurate than the batch version. In fact, its accuracy value of 70.98% is the highest accuracy of all of the incremental algorithms.

6.6 Conclusions

In this chapter, we have looked at deleting cases incrementally from the case base instead of deleting a batch of cases at once. We have developed incremental versions of the five case base maintenance algorithms and incremental computations of the twelve complexity measures that can handle multi-class classification datasets and symbolic-valued attributes. We use these to produce an incremental version of our meta-case-based classification system.

We have presented experiments that test these incremental algorithms. Unfortunately, none of the algorithms produce results that come close to the benchmark result. However, this shows that there is much room for future work to improve the incremental results. One possible way to improve our results would be to look at different criteria for deciding

when to stop deleting cases. Currently, we stop deleting when the algorithm chosen by the meta-case-based classifier does not have any cases to suggest. However, we could instead use accuracy as a flag for when to stop deletion: if the deletion of a case would cause the accuracy to drop by a certain percentage or fall below a certain threshold then we would not remove that case.

When we compare our incremental meta-CBR results to the results obtained from the incremental atomic algorithms, we find that it performs well. It has higher accuracy on average than both RCR_i and CRR_i and it has higher deletion than the other three algorithms. Its harmonic mean value comes second to RCR_i but only by 0.14. So it is competitive with the incremental atomic algorithms.

We also look at how the results of the incremental atomic algorithms compare to the original batch algorithms. The pattern is very similar, with RCR_i being the most aggressive, and BBNR_i and RENN_i being conservative but maintaining higher accuracy. However, ICFR_i gives an unexpected result. With its batch version being one of the most aggressive algorithms, we would expect that the incremental version would be similar. However, it deletes the least of all of the incremental algorithms, and produces the highest accuracy result.

Chapter 7

Conclusions and Future Work

In this dissertation, we have shown that no single case base maintenance algorithm is the best to use for all case bases at all times. This motivates our research into combining and choosing maintenance algorithms. We combine algorithms to exploit the merits of different algorithms and compensate for any weaknesses that they may have. We then look at how to choose the best algorithm to use for a case base at any given point in time so that we produce the best resulting case base that we can. In this final chapter, we draw together our conclusions and we suggest areas for continuation of this work.

7.1 Conclusions

We began our research by looking at five commonly-used maintenance algorithms. We have shown both through analysis of the algorithms and through experiments with different case bases that no one maintenance algorithm is good for all case bases in all situations. Each of the algorithms has different biases, which causes the variation in their performance over different case bases. This finding motivated the remainder of our research into finding the best maintenance algorithm for a knowledge engineer to use for a case base.

Case base maintenance is a multi-objective problem and, in evaluating the maintenance algorithms, we needed to judge both the change in accuracy after maintenance and the proportion of the case base that was deleted. We presented two possible ways of dealing with this issue. Firstly, we computed the harmonic mean of accuracy and deletion so that we could compare the algorithms using one result rather than two. We used this when presenting our results for the rest of our experiments. Secondly, we computed the Pareto front of the algorithms and showed this on a graph. This gave us a visualisation of the accuracy and deletion results of the different algorithms and showed us which algorithms did well for both criteria.

In trying to address the issue that the different maintenance algorithms have different biases and therefore choose different cases for deletion, we looked at how we could combine maintenance algorithms to allow us to draw on the strengths of a number of different algorithms to find the best subset of cases for deletion. We introduced our approach that we used to combine algorithms, which we called Maintenance by a Committee of Experts, or MACE. We presented an EBNF grammar which explained the different ways in which algorithms can be combined in MACE. We ran experiments to compare some of the possible algorithms generated by MACE to the existing maintenance algorithms and found that the MACE algorithms were competitive. However, we again found that no one algorithm

worked best for all case bases, with different combinations of algorithms performing better on different case bases. The algorithms that worked well on noisy case bases were different to those that performed well on noise-free case bases. Additionally, we looked at the area of spam filtering and found that the special characteristics of this particular task required a different combination of maintenance algorithms to many other case bases. We realised that we needed to be able to identify properties of a case base that could help us to predict the best maintenance algorithm(s) to use for that case base.

We presented seventeen complexity measures in an aim to find case base properties that we could use to provide a numerical description of the case base. Thirteen of these measures were developed by Ho and Basu [37, 38], but we modified the definitions of five of of these measures to make them more usable. We developed the remaining four from measures proposed in the CBR literature. We presented an empirical comparison of the complexity measures, showing when the measures agreed and disagreed on the complexity of a case base. We also showed which measures were predictive of classification error and we examined the effect that case base maintenance had on the complexity measures. In particular, we found that RENN may be too aggressive. It often deleted a large percentage of cases, but sometimes this deletion was at the price of quite large decreases in classifier accuracy.

We then looked at the task of choosing which maintenance algorithm is best to use for a given case base at a given time. We used twelve of the seventeen complexity measures to describe a case base at a given point in time, and we employed feature selection to choose which of these complexity measures were most useful in predicting which maintenance algorithm to use. We then presented a meta-case-based classifier which uses this numerical description of a case base to choose which maintenance algorithm to use, based on previous experience of good maintenance decisions. We performed two sets of experiments and found that the complexity measures are more reliable and predictive for choosing a maintenance algorithm to use as we improve the previous experience.

Finally, we looked at how we could make the deletion process easier for a knowledge engineer. We developed incremental versions of the CBM algorithms to give the knowledge engineer the power to decide what to delete on a case-by-case basis. We also developed incremental versions of the complexity measures to adjust them as the case base changes, and we used these to develop an incremental version of our meta-CBR system. This allowed us to make more fine-grained maintenance decisions because each time a case is deleted, the system re-evaluates its choice of maintenance algorithm to use to delete the next case. In our empirical evaluation, our incremental meta-CBR system did better than all but one of the classic atomic algorithms.

In summary, we achieved what we set out to do in this research: to improve case base maintenance algorithms by finding ways to combine maintenance algorithms to exploit the merits of different algorithms and by being able to choose the best maintenance algorithm to use for a given case base. While our work uncovers many areas for future research, we believe that the novel approaches to maintenance that we have presented will be of much use to case-based reasoning systems.

7.2 Future Work

In our study of case base maintenance algorithms, we have discovered a number of ways to improve their performance and useability for knowledge engineers. However, in doing so,

we have uncovered a numbers of areas that could be addressed in future work. This section presents these areas.

- We have chosen to work with five CBM algorithms which are commonly-used and which we believe to be representative of all CBM algorithms in the literature. It would be interesting to analyse other CBM algorithms and to see how they perform if they were included in MACE and in our meta-case-based classifiers. In particular, we would like to see how Delany's RDCL algorithm [18] could contribute to our MACE and meta-CBR systems since it shows improved accuracy results over the BBNR and RENN noise reduction algorithms. It would also be interesting to see what effect deletion of cases with different RDCL profiles has on dataset complexity.

- On a similar note, we chose seventeen case base measures to work with to describe the complexity of a case base. It would be interesting to look at other measures that could be used, or even to devise new measures. For example, new measures of the overlap of attribute values could be based on entropy or the Gini coefficient.

- In addition, we would like to extend our experiments to see how our approaches work in different situations. Firstly, we could extend the range of datasets that we use for experimentation. On the one hand, we could use real datasets that exhibit different characteristics from the ones we have used, such as image processing and bioinformatics datasets. On the other hand, there is also room for more work using artificial datasets as in [37, 38, 51].

- Further work for our MACE approach would involve investigating different combinations of algorithms. Our current experimentation places a number of restrictions on the algorithms generated by the grammar. We could remove some or all of these restrictions to produce more MACE algorithms to work with and test.

- Our approach for choosing which maintenance algorithm to use has a number of areas for future work. Firstly, the meta-case-based classifier uses multi-label classification. At present we use a very basic system for choosing a winner: we simply count up the votes that each algorithm gets among the k-NN. However, there are a number of other ways to deal with multi-label classification [90]. It would be interesting to see how using these would impact our meta-case-based classifier system. Secondly, we chose to use a CBR approach to choosing the best algorithm to use since it was in line with the rest of our work. However, we also pointed out the fact that we could have used any classification or regression method such as a neural net, a rule-based system or a decision tree. We would like to see if we could improve our results by finding which system works best for our task.

- In our work on incremental measures and algorithms, we also found a number of areas for future work. Firstly, it would be interesting to investigate optimum stopping points for the incremental algorithms. At present, when they run to completion, there is an accuracy drop for all of the algorithms. We would like to see if we can use some trigger for when to stop the incremental algorithm so that accuracy is maintained.

  Secondly, we would like to use our incremental algorithms to help the knowledge engineer through the process of maintaining their case base. When he is presented with a large group of cases suggested for deletion, a knowledge engineer may be afraid

of damaging his system by deleting them all at once. Using the incremental algorithms, we can present him with each case suggested for deletion in turn and ask him to make the choice of whether to delete or keep that case. This makes it easier for him because he only needs to consider one case at a time instead of the group of cases suggested by a batch algorithm. It also allows him more control over the percentage of the case base to delete because he can stop the deletion process at any stage when he feels that enough cases have been removed.

Additionally, the deletion decisions made by the knowledge engineer can be remembered by the system. Maintenance algorithms could then use these as part of their deletion process. For example, if the engineer marked a particular case as being very important for the system, it can be ignored by the CBM algorithm for a certain length of time (after which it may be worth re-checking in case things have changed).

Appendix A

Correlation Tables

In Chapter 4, we looked at how complexity measure values correlate with the error rates of a number of classifiers. This appendix provides the full tables of these correlation values. We have four classifiers: NN, SVM, J48 and IBk. We show the correlation of each of the measures with each of these classifiers, and also with the mean error over the four classifiers.

We show two tables of correlation coefficients. The first table, Table A.1, is based on all twenty-five datasets (although a number of measures are only calculated for twenty-one of these because they cannot handle symbolic values). The second table, Table A.2, is based on just the fourteen Boolean classification datasets (although, again, a number of measures can only be computed on thirteen of these.

Table A.1: Correlation coefficients of measures and classifiers over all twenty-five datasets

	NN	SVM	J48	IBk	Mean Error	Number of datasets
F_1	0.01	0.1	0.05	0.01	0.05	21
F'_2	0.1	0.12	0.23	0.2	0.16	25
F'_3	-0.12	-0.07	-0.21	-0.18	-0.14	25
F'_4	-0.46	-0.44	-0.57	-0.5	-0.5	25
N'_1	0.94	0.93	0.94	0.95	0.96	25
N_2	0.84	0.77	0.85	0.85	0.84	25
N_3	0.96	0.9	0.95	0.97	0.96	25
N_5	0.95	0.9	0.95	0.95	0.95	25
L_1	0	0.01	0.14	0.07	0.05	21
L_2	0.62	0.56	0.7	0.64	0.64	21
C_1	0.98	0.95	0.96	0.98	0.98	25
C_2	0.98	0.93	0.92	0.96	0.97	25
L_3	0.54	0.52	0.62	0.55	0.57	21
N_4	0.75	0.66	0.81	0.81	0.77	21
T'_1	0.55	0.49	0.48	0.55	0.53	25
T_2	0.08	0.12	0.23	0.16	0.15	25
T_3	-0.22	-0.2	-0.08	-0.16	-0.17	25

Table A.2: Correlation coefficients of measures and classifiers over fourteen Boolean classification datasets

	NN	SVM	J48	IBk	Mean Error	Number of datasets
F_1	-0.29	-0.33	-0.25	-0.31	-0.3	13
F_2'	0.44	0.37	0.39	0.48	0.42	14
F_3'	-0.27	-0.25	-0.3	-0.25	-0.27	14
F_4'	-0.72	-0.68	-0.69	-0.64	-0.68	14
N_1'	0.98	0.95	0.97	0.97	0.98	14
N_2	0.82	0.75	0.85	0.8	0.81	14
N_3	0.98	0.93	0.97	0.96	0.97	14
N_5	0.95	0.92	0.96	0.94	0.95	14
L_1	0.27	0.4	0.39	0.29	0.34	13
L_2	0.95	0.96	0.95	0.95	0.96	13
C_1	0.98	0.96	0.98	0.97	0.98	14
C_2	0.99	0.96	0.97	0.98	0.98	14
L_3	0.97	0.91	0.92	0.96	0.95	13
N_4	0.86	0.82	0.86	0.85	0.85	13
T_1'	0.18	0.14	0.21	0.13	0.17	14
T_2	0.69	0.62	0.67	0.64	0.66	14
T_3	-0.22	-0.25	-0.19	-0.3	-0.25	14

Bibliography

[1] A. Aamodt and E. Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7(1):39–59, March 1994.

[2] D. W. Aha and R. L. Bankert. A comparative evaluation of sequential feature selection algorithms. In *Proceedings of the 5th International Workshop on Artificial Intelligence and Statistics*, pages 1–7, 1994.

[3] D.W. Aha and L.A. Breslow. Refining conversational case libraries. In *Proceedings of the 2nd International Conference on Case-Based Reasoning*, pages 291–302, Heidelberg, Germany, 1997. ICCBR, Springer-Verlag.

[4] H. Bensusan, C. Giraud-Carrier, and C. Kennedy. A higher-order approach to meta-learning. In *Proceedings of the Workshop on Meta-Learning at the European Conference on Machine Learning*, pages 109–117. ECML, June 2000.

[5] E. Bernadó-Mansilla and T. K. Ho. On classifier domains of competence. In *Proceedings of the 17th International Conference on Pattern Recognition*, pages 136–139, 2004.

[6] E. Bernadó-Mansilla, T. K. Ho, and A. Orriols. Data complexity and evolutionary learning. In M Basu and T. K. Ho, editors, *Data Complexity in Pattern Recognition*, pages 115–134. Springer, 2006.

[7] A. Bonzano, P. Cunningham, and B. Smyth. Using introspective learning to improve retrieval in CBR: A case study in air traffic control. In *Proceedings of the 2nd International Conference on Case-Based Reasoning*, pages 291–302, Heidelberg, Germany, 1997. ICCBR, Springer-Verlag.

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

[9] H. Brighton and C. Mellish. On the consistency of information filters for lazy learning algorithms. In J. Rauch and J.M. Zytkow, editors, *Proceedings of the 3rd European Conference on Principles of Data Mining and Knowledge Discovery*, pages 283–288. Springer-Verlag, 1999.

[10] C. E. Brodley and M. A. Friedl. Identifying and eliminating mislabeled training instances. In *In AAAI/IAAI*, pages 799–805. AAAI Press, 1996.

[11] M.T. Cox and A. Ram. Introspective multistrategy learning: On the construction of learning strategies. *Artificial Intelligence*, 112:1–55, 1999.

[12] K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292, 2001.

[13] S. Craw, J. Jarmulak, and R. Rowe. Maintaining retrieval knowledge in a case-base reasoning system. *Computational Intelligence*, 17(2):346–363, May 2001.

[14] L. Cummins and D. Bridge. Maintenance by a committee of experts: The MACE approach to case-base maintenance. In L. McGinty and D. C. Wilson, editors, *Case-Based Reasoning Research and Development (Proceedings of the 8th International Conference on Case-Based Reasoning)*, LNAI 5650, pages 120–134. Springer, 2009.

[15] L. Cummins and D. Bridge. Choosing a case base maintenance algorithm using a meta-case base. In M. Bramer, M. Petridis, and L. Nolle, editors, *Research and Development in Intelligent Systems XXVIII (Proceedings of AI-2011, The 31st SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence)*, pages 167–180. Springer, 2011.

[16] L. Cummins and D. Bridge. On dataset complexity for case base maintenance. In A. Ram and N. Wiratunga, editors, *Case-Based Reasoning Research and Development (Proceedings of the 9th International Conference on Case-Based Reasoning)*, LNAI 6880, pages 47–61. Springer, 2011.

[17] P. Cunningham and G. Zenobi. Case representation issues for case-based reasoning from ensemble research. In D.W. Aha and I. Watson, editors, *Proceedings of the 4th International Conference on Case-Based Reasoning*, volume 2080 of *Lecture Notes in Computer Science*, pages 146–157. ICCBR, Springer-Verlag, 2001.

[18] S. J. Delany. The good, the bad and the incorrectly classified: Profiling cases for case-base editing. In L. McGinty and D. C. Wilson, editors, *Proceedings of the 8th International Conference on Case-Based Reasoning*, volume 5650 of *Lecture Notes in Computer Science*, pages 135–149. Springer, 2009.

[19] S.J. Delany and P. Cunningham. An analysis of case-based editing in a spam filtering system. In P. Funk and P. González-Calero, editors, *Proceedings of the 7th European Conference on Case-Based Reasoning*, volume 3155 of *Lecture Notes in Artificial Intelligence*, pages 128–141. ECCBR, Springer, 2004.

[20] T. G. Dietterich. Ensemble methods in machine learning. In J. Kittler and F. Roli, editors, *Multiple Classifier Systems*, volume 1857 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2000.

[21] D. Doyle, P. Cunningham, D. Bridge, and Y. Rahman. Explanation oriented retrieval. In P. Funk and P. González-Calero, editors, *Proceedings of the 7th European Conference on Case-Based Reasoning*, volume 3155 of *Lecture Notes in Artificial Intelligence*, pages 157–168. ECCBR, Springer, 2004.

[22] A. Fornells, J.A. Recio-García, B. Díaz-Agudo, E. Golobardes, and E. Fornells. Integration of a methodology for cluster-based retrieval in jcolibri. In L. McGinty and D. C. Wilson, editors, *Proceedings of the 8th International Conference on Case-Based Reasoning*, volume 5650 of *Lecture Notes in Computer Science*, pages 418–433. Springer, 2009.

[23] S. Fox and D. Leake. Learning to refine indexing by introspective reasoning. In M. M. Veloso and A. Aamodt, editors, *Proceedings of the 1st International Conference on Case-Based Reasoning*, volume 1010 of *Lecture Notes in Computer Science*, pages 430–440, Heidelberg, Germany, October 1995. ICCBR, Springer.

[24] S. Fox and D. Leake. Using introspective reasoning to refine indexing. In C.S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 391–397, San Francisco, California, August 1995. IJCAI, Morgan Kaufmann.

[25] S. Fox and D. Leake. Introspective reasoning for index refinement in case-based reasoning. *Journal of Experimental and Theoretical Artificial Intelligence*, 13(1):63–88, 2001.

[26] A.G. Francis and A. Ram. Computational models of the utility problem and their application to utility analysis of cased-based reasoning. In *Proceedings of the Workshop on Knowledge Compilation and Speed-Up Learning*, 1993.

[27] A. Frank and A. Asuncion. UCI machine learning repository. http://archive.ics.uci.edu/ml, 2010.

[28] Y. Freund and R.E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In Paul M. B. Vitányi, editor, *EuroCOLT '95: Proceedings of the 2nd European Conference on Computational Learning Theory*, volume 904 of *Lecture Notes in Computer Science*, pages 23–37. Springer, 1995.

[29] G.W. Gates. The reduced nearest neighbor rule. *IEEE Transactions on Information Theory*, 18(3):431–433, May 1972.

[30] M.H. Göker and T. Roth-Berghofer. Development and utilization of a case-based help-desk support system in a corporate environment. In K.-D. Althoff, R. Bergmann, and L.K. Branting, editors, *Proceedings of the 3rd International Conference on Case-Based Reasoning*, volume 1650 of *Lecture Notes in Computer Science*, pages 132–146, Heidelberg, Germany, 1999. ICCBR, Springer-Verlag.

[31] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[32] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The WEKA data mining software: An update. *SIGKDD Explorations*, 11(1), 2009.

[33] K.J. Hammond. Explaining and repairing plans that fail. *Artificial Intelligence*, 45:173–228, 1990.

[34] A. Hanft and M. Minor. A low-effort, collaborative maintenance model for textual CBR. In S. Brüninghaus, editor, *Proceedings of the Workshop Programme at the 6th International Conference on Case-Based Reasoning*, pages 138–149, 2005.

[35] P.E. Hart. The condensed nearest neighbor rule. *IEEE Transactions on Information Theory*, 14(3):515–516, May 1968.

[36] B. Hayes-Roth. An architecture for adaptive intelligent systems. *Artificial Intelligence*, 72:329–365, 1995.

[37] T. K. Ho and M. Basu. Measuring the complexity of classification problems. In *Proceedings of the 15th International Conference on Pattern Recognition*, pages 43–47, 2000.

[38] T. K. Ho and M. Basu. Complexity measures of supervised classification problems. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(3):289–300, 2002.

[39] T. K. Ho, M. Basu, and M. Law. Measures of geometrical complexity in classification problems. In *Data Complexity in Pattern Recognition*, pages 1–23. Springer-Verlag, London, 2006.

[40] A. Hoekstra and R. P. W. Duin. On the nonlinearity of pattern classifiers. In *Proceedings of the 13th International Conference on Pattern Recognition*, pages 271–275, 1996.

[41] J. Jarmulak, S. Craw, and R. Rowe. Using case-base data to learn adaptation knowledge for design. In *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, pages 1011–1016, San Francisco, California, 2001. IJCAI, Morgan Kaufmann.

[42] J. Kolodner. *Case-Based Reasoning*. Morgan Kaufmann, San Mateo, California, 1993.

[43] Luc Lamontagne. Textual CBR authoring using case cohesion. In M. Minor, editor, *Proceedings of the Workshop on Textual Case-Based Reasoning, Workshop Programme, 8th European Conference on Case-Based Reasoning*, pages 33–43, 2006.

[44] D. B. Leake, A. Kinley, and D. C. Wilson. Learning to improve case adaption by introspective reasoning and CBR. In M. M. Veloso and A. Aamodt, editors, *Proceedings of the 1st International Conference on Case-Based Reasoning*, volume 1010 of *Lecture Notes in Computer Science*, Heidelberg, Germany, October 1995. ICCBR, Springer.

[45] D.B. Leake and R. Sooriamurthi. When two case bases are better than one: Exploiting multiple case bases. In D.W. Aha and I. Watson, editors, *Proceedings of the 4th International Conference on Case-Based Reasoning*, volume 2080 of *Lecture Notes in Computer Science*, pages 321–335. ICCBR, Springer-Verlag, 2001.

[46] D.B. Leake and D.C. Wilson. Categorizing case-base maintenance: Dimensions and directions. In B. Smyth and P. Cunningham, editors, *Proceedings of the 4th European Conference on Case-Based Reasoning*, pages 196–207, Heidelberg, Germany, 1998. ECCBR, Springer-Verlag.

[47] D.B. Leake and D.C. Wilson. When experience is wrong: Examining CBR for changing tasks and environments. In K.-D. Althoff, R. Bergmann, and L.K. Branting, editors, *Case-Based Reasoning Research and Development, Proceedings of the 3rd International Conference on Case-Based Reasoning*, volume 1650 of *Lecture Notes in Computer Science*, pages 218–232, Heidelberg, Germany, 1999. ICCBR, Springer-Verlag.

[48] D.B. Leake and D.C. Wilson. Remembering why to remember: Performance guided case-base maintenance. In E. Blanzieri and L. Portinale, editors, *Proceedings of the 5th European Workshop on Advances in Case-Based Reasoning*, volume 1898 of *Lecture*

Notes in Computer Science, pages 161–172, Heidelberg, Germany, September 2000. EWCBR, Springer-Verlag.

[49] F. LeBourgeois and H. Emptoz. Pretopological approach for supervised learning. In *Proceedings of the 13th International Conference on Pattern Recognition*, pages 256–260, Washington, DC, USA, 1996. IEEE Computer Society.

[50] H. Liu and L. Yu. Toward integrating feature selection algorithms for classification and clustering. *IEEE Transactions on Knowledge and Data Engineering*, 17:491–502, 2005.

[51] N. Macià, E. Bernadó-Mansilla, and A. Orriols-Puig. On the dimensions of data complexity through synthetic data sets. In *Proceedings of the 11th International Conference of the Catalan Association for Artificial Intelligence*, pages 244–252, 2008.

[52] W. Malina. Two-parameter Fisher criterion. *IEEE Transactions on Systems, Man, and Cybernetics*, 31(4):629–636, 2001.

[53] S. Markovitch and P.D. Scott. Information filtering: Selection mechanisms in learning systems. *Machine Learning*, 10(2):113–151, February 1993.

[54] S. Massie, S. Craw, and N. Wiratunga. Complexity-guided case discovery for case-based reasoning. In T. Roth-Berghofer, M.H. Göker, and H. Altay Güvenir, editors, *Proceedings of the 20th National Conference on Artificial Intelligence*, pages 216–221. AAAI, AAAI Press, 2005.

[55] S. Massie, S. Craw, and N. Wiratunga. Complexity profiling for informed case-base editing. In T. Roth-Berghofer, M. H. Göker, and H. Altay Güvenir, editors, *Proceedings of the 8th European Conference on Case-Based Reasoning*, pages 325–339. ECCBR, Springer-Verlag, 2006.

[56] S. Massie, S. Craw, and N. Wiratunga. When similar problems don't have similar solutions. In R. Weber and M. Richter, editors, *Proceedings of the 7th International Conference on Case-Based Reasoning*, pages 92–106, Berlin, Heidelberg, 2007. ICCBR, Springer-Verlag.

[57] L. McGinty and B. Smyth. Collaborative case-based reasoning: Applications in personalized route planning. In D. W. Aha and I. Watson, editors, *Proceedings of the 4th International Conference on Case-Based Reasoning*, pages 362–376, Heidelberg, Germany, 2001. ICCBR, Springer-Verlag.

[58] E. McKenna and B. Smyth. Competence-guided case-base editing techniques. In E. Blanzieri and L. Portinale, editors, *Proceedings of the 5th European Workshop on Advances in Case-Based Reasoning*, volume 1898 of *Lecture Notes in Computer Science*, pages 186–197, Heidelberg, Germany, September 2000. EWCBR, Springer-Verlag.

[59] S. Minton. Quantitative results concerning the utility of explanation-based learning. *Artificial Intelligence*, 42(2-3):363–391, March 1990.

[60] T. M. Mitchell. The need for biases in learning generalizations. In J. Shavlik and T. Dietterich, editors, *Readings in Machine Learning*, pages 184–191. Morgan Kaufmann, 1990.

[61] H. Muñoz-Avila. Case-base maintenance by integrating case-index revision and case-retention policies in a derivational replay framework. *Computational Intelligence*, 17(2):280–294, May 2001.

[62] H. Muñoz-Avila and J. Hüllen. Feature weighting by explaining case-based planning episodes. In I. Smith and B. Faltings, editors, *Proceedings of the 3rd European Workshop on Advances in Case-Based Reasoning*, volume 1168 of *Lecture Notes In Computer Science*, pages 280–294. EWCBR, Springer-Verlag, 1996.

[63] A. Orecchioni, N. Wiratunga, S. Massie, and S. Craw. k-NN aggregation with a stacked email representation. In K.-D. Althoff, R. Bergmann, M. Minor, and A. Hanft, editors, *Proceedings of the 9th Eurpoean Conference on Case-Based Reasoning*, volume 5239 of *Lecture Notes in Computer Science*, pages 415–429. ECCBR, Springer-Verlag, 2008.

[64] A. Orriols-Puig, N. Macià, E. Bernadó-Mansilla, and T. K. Ho. Documentation for the data complexity library in C++. Technical Report GRSI Report Number 2009001, Universitat Ramon Llull, 2009.

[65] H. Osborne and D. Bridge. A case base similarity framework. In I. Smith and B. Faltings, editors, *Proceedings of the 3rd European Workshop on Case-Based Reasoning*, pages 309–323, 1996.

[66] Y. Peng, P. A. Flach, C. Soares, and P. Brazdil. Improved data set characterisation for meta-learning. In *Proceedings of the 5th International Conference on Discovery Science*, pages 141–152. Springer-Verlag, January 2002.

[67] B. Pfahringer, H. Bensusan, and C. Giraud-Carrier. Meta-learning by landmarking various learning algorithms. In *In Proceedings of the 17th International Conference on Machine Learning*, pages 743–750. Morgan Kaufmann, 2000.

[68] E. Plaza and L. McGinty. Distributed case-based reasoning. *The Knowledge Engineering Review*, 20(3):261–265, 2006.

[69] E. Plaza and S. Ontañón. Ensemble case-based reasoning: Collaboration policies for multi-agent cooperative CBR. In D. W. Aha and I. Watson, editors, *Proceedings of the 4th International Conference on Case-Based Reasoning*, pages 437–451, Heidelberg, Germany, 2001. ICCBR, Springer-Verlag.

[70] J.R. Quinlan. Bagging, boosting, and c4.5. In *AAAI/IAAI, Vol. 1*, pages 725–730. AAAI Press, 1996.

[71] K. Racine and Q. Yang. Maintaining unstructured case bases. In *Proceedings of the 2nd International Conference on Case-Based Reasoning*, pages 553–564, Heidelberg, Germany, 1997. ICCBR, Springer-Verlag.

[72] K. Racine and Q. Yang. Redundancy detection in semi-structured case bases. *IEEE Transactions on Knowledge and Data Engineering*, 13(3):513–518, May-June 2001.

[73] J. A. Recio-García, D. Bridge, B. Díaz-Agudo, and P. A. González-Calero. CBR for CBR: A case-based template recommender system for building case-based systems. In K.-D. Althoff, R. Bergmann, M. Minor, and A. Hanft, editors, *Proceedings of the*

9th Eurpoean Conference on Case-Based Reasoning, volume 5239 of *Lecture Notes in Computer Science*, pages 459–473. ECCBR, Springer-Verlag, 2008.

[74] T. Reinartz, I. Iglezakis, and T. Roth-Berghofer. On quality measures for case base maintenance. In E. Blanzieri and L. Portinale, editors, *Proceedings of the 5th European Workshop on Advances in Case-Based Reasoning*, volume 1898 of *Lecture Notes in Computer Science*, pages 247–259, Heidelberg, Germany, September 2000. EWCBR, Springer-Verlag.

[75] T. Reinartz, I. Iglezakis, and T. Roth-Berghofer. Review and restore for case-based maintenance. *Computational Intelligence*, 17(2):214–234, May 2001.

[76] M. Richter. Introduction. In M. Lenz, B. Bartsch-Spörl, H.-D. Burkhard, and S. Wess, editors, *Case-Based Reasoning Technology, From Foundations to Applications*, volume 1400 of *Lecture Notes in Computer Science*, pages 1–16. Springer-Verlag, Heidelberg, Germany, 1998.

[77] C.K. Riesbeck and R.C. Shank. *Inside Case-Based Reasoning*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1989.

[78] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *Journal of Machine Learning Research*, 5:101–141, 2004.

[79] G.L. Ritter, H.B. Woodruff, S.R. Lowry, and T.L. Isehhour. An algorithm for a selective nearest neighbor decision rule. *IEEE Transactions on Information Theory*, 21(6):665–669, November 1975.

[80] T. Roth-Berghofer and I. Iglezakis. Six steps in case-based reasoning: Towards a maintenance methodology for case-based reasoning systems. In *Proceedings of the 9th German Workshop on Case-Based Reasoning*, pages 198–208, Aachen, 2001. GWCBR, Shaker–Verlag.

[81] C. Schaffer. A conservation law for generalization performance. In W. W. Cohen and H. Hirsh, editors, *Proceedings of the 11th International Conference on Machine Learning*, pages 259–265, 1994.

[82] S.C.K. Shiu, D.S. Yeung, C.H. Sun, and X.Z. Wang. Transferring case knowledge to adaptation knowledge: An approach for case-base maintenance. *Computational Intelligence*, 17(2):295–314, May 2001.

[83] B. Smyth and P. Cunningham. The utility problem analysed: A case-based reasoning perspective. In I. Smith and B. Faltings, editors, *Proceedings of the 3rd European Workshop on Advances in Case-Based Reasoning*, volume 1168 of *Lecture Notes In Computer Science*, pages 392–399, Heidelberg, Germany, November 1996. EWCBR, Springer-Verlag.

[84] B. Smyth and M.T. Keane. Remembering to forget: A competence-preserving case-deletion policy for case-based reasoning systems. In C.S. Mellish, editor, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pages 377–383, San Francisco, California, August 1995. IJCAI, Morgan Kaufmann.

[85] B. Smyth and E. McKenna. Modelling the competence of case-bases. In B. Smyth and P. Cunningham, editors, *Proceedings of the 4th European Workshop on Advances in Case-Based Reasoning*, volume 1488 of *Lecture Notes in Computer Science*, pages 208–220, Heidelberg, Germany, 1998. EWCBR, Springer.

[86] B. Smyth and E. McKenna. Building competent compact case-bases. In K.-D. Althoff, R. Bergmann, and L.K. Branting, editors, *Case-Based Reasoning Research and Development, Proceedings of the 3rd International Conference on Case-Based Reasoning*, volume 1650 of *Lecture Notes in Computer Science*, pages 329–342, Heidelberg, Germany, 1999. ICCBR, Springer-Verlag.

[87] M. Stefik. Planning and meta-planning (MOLGEN: Part 2). *Artificial Intelligence*, 16(2):141–170, 1981.

[88] I. Tomek. An experiment with the edited nearest-neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 6(6):448–452, June 1976.

[89] I. Tomek. Two modifications of CNN. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(2):679–772, 1976.

[90] G. Tsoumakas and I. Katakis. Multi-label classification: An overview. *International Journal of Data Warehousing and Mining*, 2007(3):1–13, 2007.

[91] M. van Setten. *Supporting People in Finding Information: Hybrid Recommender Systems and Goal-Based Structuring*. PhD thesis, University of Twente, 2005.

[92] I. Watson. Report of the workshop on automating the construction of case-based reasoners at the 16th international joint conference on artificial intelligence (IJCAI 1999). http://www.ai-cbr.org/ijcai99/workshop.html, 1999.

[93] B. Weber, M. Reichert, and W. Wild. Case-base maintenance for CCBR-based process evolution. In T. Roth-Berghofer, M. H. Göker, and H. A. Güvenir, editors, *Proceedings of the 8th European Conference on Case-Based Reasoning*, volume 4106 of *Lecture Notes in Computer Science*, pages 106–120, Heidelberg, Germany, 2006. ECCBR, Springer.

[94] D. Wettschereck and D.W. Aha. Weighting features. In *Proceedings of the 1st International Conference on Case-Based Reasoning*, pages 347–358. ICCBR, Springer, 1995.

[95] R. Wilensky. Meta-planning: Representing and using knowledge about planning in problem solving and natural language understanding. *Cognitive Science*, 5(3):197–233, 1981.

[96] D. R. Wilson and T. R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 6:1–34, 1997.

[97] D.L. Wilson. Asymptotic properties of nearest neighbor rules using edited data. *IEEE Transactions on Systems, Man, and Cybernetics*, 2(3):408–421, May/June 1972.

[98] D.R. Wilson and T.R. Martinez. Instance pruning techniques. In D. Fisher, editor, *Proceedings of the 14th International Conference on Machine Learning*, pages 403–411, San Francisco, California, July 1997. ICML, Morgan Kaufmann.

[99] D.R. Wilson and T.R. Martinez. Reduction techniques for instance-based learning algorithms. *Machine Learning*, 38(3):257–286, March 2000.

[100] N. Wiratunga, S. Craw, and Ray. Rowe. Learning to adapt for case-based design. In S. Craw and A. D. Preece, editors, *Proceedings of the 6th European Conference on Case-Based Reasoning*, volume 2416 of *Lecture Notes in Computer Science*, pages 421–435. ECCBR, Springer-Verlag, 2002.

[101] D. H. Wolpert. On the connection between in-sample testing and generalisation. *Complex Systems*, 6:47–94, 1992.

[102] Z. Zhang and Q. Yang. Dynamic refinement of feature weights in CBR using quantitative introspective learning. In T. Dean, editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 228–233, San Francisco, California, August 1999. IJCAI, Morgan Kaufmann.