A Knowledge-Light Approach to Regression using Case-Based Reasoning

Neil McDonnell

A thesis submitted to the University of Dublin, Trinity College
in fulfilment of the requirements for the degree of
Doctor of Philosophy

October 2006

Declaration

Acknowledgements

I would like to thank a number of people who have helped me in various ways over the past three years. Without their assistance, this thesis would certainly not have seen the light of day.

First, let me express my gratitude to my supervisor, Professor Pádraig Cunningham. His clear technical direction was invaluable in guiding me to a productive research path when I was struggling to find my bearings. He was unfailingly generous in his encouragement and support, and kind enough to administer a gentle push when it was most needed.

Second, I would like to thank my colleagues in the Machine Learning Group in Trinity College Dublin, both past and present. These include Lorcan Coyle, Marco Grimaldi, Sarah Jane Delany, John Loughrey, Conor Nugent, Dónal Doyle, Mike Carney, Ken Bryan, Derek Greene, Deirdre Hogan, Michael Davy and Alexey Tsymbal. It has been a pleasure to work with them, and I am constantly impressed by the dedication and expertise they bring to their research.

Third, thanks to my colleagues in the University of Ulster who have worked with me under the North-South Cooperation Programme. Our frequent meetings helped me crystallise many of the ideas that are presented in this thesis.

Finally, I owe a special thanks to my wife, Edel, for everything she has done for me over the years. This thesis is dedicated to her.

Neil McDonnell
University of Dublin, Trinity College
October 2006

Abstract

Case-based reasoning (CBR) is among the most influential paradigms in modern machine learning. It advocates a strategy of storing specific experiences in the form of cases, and solving new problems by re-using solutions from similar past cases. The most difficult aspect of CBR is deciding how to adapt past solutions to precisely match the circumstances of new problems. No generally applicable method of doing this has been found; different domains and tasks have their own individual characteristics, and successful adaptation has usually relied on the presence of explicit, hand-coded domain knowledge. Such knowledge is usually difficult both to acquire and maintain. For this reason, most CBR systems in operation today are 'retrieval only' in that they do not attempt to adapt the solutions of past cases to solve new problems.

For certain machine learning tasks, however, customisation of old solutions can be performed using only knowledge contained within the set of stored cases. One such task is regression (i.e. predicting the value of a numeric variable). Regression is among the oldest machine learning tasks, dating back to Francis Galton's work on predicting the heights of parents and their children in nineteenth century England. A modern example would be to predict tomorrow's stock market prices based on today's financial data. Many different approaches to solving regression problems have been developed over the years, for example, *k*-NN, locally weighted linear regression and artificial neural networks.

The aim of this thesis is to apply CBR to the problem of regression. It begins by analysing previous attempts to do this, paying particular attention to those aspects that might be improved. One CBR-based approach from the mid-1990's is examined in considerable detail. It works by finding the differences between a new problem and a similar past problem, then searching for a pair of stored cases with the same differences between them. These stored cases indicate the effect of the differences on the solution. This 'case differences' approach has much to recommend it. In particular, the knowledge needed to solve new problems is automatically generated from stored cases—no additional external knowledge must be added. Unfortunately, it also suffers from some theoretical limitations that greatly restrict its use.

This thesis presents two new CBR-based regression algorithms that build on the strengths of previous approaches while addressing their limitations. One is a minor variant of the traditional *k*-NN algorithm, while the other uses the case differences approach and is more sophisticated. The main contribution of the second algorithm is that it uses locally weighted linear regression as a guide to help choose past cases that are likely to be useful for solving new problems. It also takes steps to increase robustness when basing predictions on noisy datasets. An experimental evaluation of the new techniques shows that they perform well relative to standard regression algorithms on a range of datasets.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The field of Machine Learning looks at ways in which computer systems can learn from past experiences [Mitchell 1997, p. xv]. Most computer systems do not display this ability, but instead have their behaviour fully defined before they enter operation. All knowledge necessary for carrying out their tasks is pre-programmed; situations not catered for are viewed as specification and/or design failures, to be remedied by corrections in which the missing functionality is added. This approach is appropriate in circumstances where a system's inputs, outputs, and operating environment can be fully specified in advance. Imagine, for example, a system that accepts the votes from an election in a particular format, counts them according to a defined set of rules, and outputs a set of results. Assuming that everything is designed correctly, the system will display 100% competence in its task from the moment of deployment.

Unfortunately, a complete specification of this kind is not always available (or possible). Suppose, for example, that we are asked to construct an online customer support system that helps customers find solutions to problems they encounter while attempting to assemble an item of furniture. The range of problems that may arise cannot be fully specified in advance. What is needed is a system that initially offers assistance for what are anticipated to be the most common problems, and that expands its expertise based on the range of problems presented by users over time. This can be achieved by regular updates to the system by its human designers. Alternatively, the system can update itself by incorporating a machine learning element that learns from its experiences in some way. This element might accept discrete user sessions in a pre-defined format and use them to improve the system's response when faced with similar situations in the future.

In this example, a machine learning element embedded in a larger computer system monitors the system's performance with a view to improving it. This architecture is common to all machine learning systems [Russell and Norvig, pp. 51–54], but each system's task is likely to be unique. In order to study machine learning in isolation, therefore, the system task is often assumed to be one of a number of standard Artificial Intelligence tasks. Regression (i.e., predicting the value of a numeric variable) is among the most fundamental of these, both in its own right and as a sub-component of more complex tasks [Campbell at al. 2000]. Sample regression applications include predicting tomorrow's temperature based on today's meteorological data, or predicting a person's blood pressure based on lifestyle details.

This thesis focuses on the problem of regression in machine learning. It describes a generic framework for accepting prior experiences and using them to predict the value of some numeric variable. In principle, these prior experiences can take any form. In practice, however, they can usually be represented as a set of records in a database, each containing a number of well-defined fields. We will assume that prior experiences are available in this form, and that at least one data field has a numeric type. The two primary technologies used to construct this framework are case-based reasoning [Aamodt and Plaza 1994; Kolodner 1992] and locally weighted linear regression [Atkeson at al. 1996].

1.1 Case-Based Reasoning

In case-based reasoning (CBR) systems, prior experiences are stored as a set of *cases* in a *case base* (CB). New problems are solved by re-using solutions from similar, previously solved cases. Each case generally has a problem part that describes the problem to be solved, and a solution part that details the solution eventually applied. In this thesis, the problem part is assumed to consist of a set of problem attributes (at least one of which is numeric), with the solution part comprising a single numeric solution attribute.

A new case to be solved is often called a *query* case. Given values for its problem attributes, the system's task is to predict the value for its solution attribute (also referred to as the *target value*). The problem solving process involves retrieving prior cases that are similar to the query, and re-using their solutions in some way.

CBR systems have been described as *lazy* and *local*—lazy in that the process of generalizing from past experiences is deferred until a query is received, and local in that only those cases most similar to the query are used to provide a solution. This approach contrasts with that of *eager*, *global* learners such as neural networks, where all prior experiences are compiled into a global model during an initial training phase and this model is then used to predict solutions for new problems.

Case-based reasoning is described in greater detail in Chapter 3.

1.2 Locally Weighted Linear Regression

Locally weighted linear regression (LWLR) is another lazy, local approach to regression. Each problem is represented as a set of numeric attributes. When a new problem is received, similar past problems are retrieved and used to construct a linear model in multi-dimensional domain space. This model is than used to predict a solution.

LWLR is one of a number of standard algorithms that are commonly used to tackle regression problems. Although computationally expensive, it tends to perform well in domains where all relevant aspects of a problem can be represented by numeric attributes. Chapter 2 describes LWLR in greater detail.

1.3 Problem Statement

Regression is among the most important problems in machine learning. CBR is among the most important techniques in modern Artificial Intelligence [Aha 1998]. The goal of this research is to combine the two by using CBR to solve regression problems. In particular, it aims to develop a regression system that demonstrates the following capabilities:

- Prior experiences are stored as a set of cases, where each case is represented by a set of problem attributes—at least one of which is numeric—and a numeric solution attribute.
- New query cases are solved by retrieving similar past cases and re-using their solutions.
- Prediction accuracy matches or exceeds that of alternative regression techniques on a range of standard real-world datasets.

The aim of this research can be summed up as follows:

Design a regression system using CBR that provides effective, robust performance across different problem domains.

1.4 Contributions of this Thesis

The principal contributions of this thesis are as follows:

1. Previous research into the use of CBR for regression is critically analysed. Shortcomings in previous approaches are identified and discussed, and inform the requirements for an improved approach.
2. Two new CBR-based regression algorithms are presented. The first is a minor variant of *k*-NN in which diversity among nearest neighbours is preferred. The second aims to address the limitations of previous approaches and is more complex and robust.
3. A working CBR-based prototype is implemented that provides accurate, robust performance.

1.4.1 Analysis of previous research

The traditional approach to solving regression problems using CBR has been to apply a modified *k*-NN algorithm (for an introduction to *k*-NN, see [Mitchell 1997, Chapter 8]); predictions are made by taking the weighted average of solutions from past cases similar to the query. An alternative approach that uses the differences between stored cases to generate a set of adaptation rules was proposed in the late 1990's [Hanney and Keane 1996, 1997; McSherry 1998]. This approach contained much that was promising, but was hampered by theoretical limitations that made it unsuitable for use in real-world domains. Evaluations of the technique were therefore restricted to artificial datasets.

This thesis examines these previous approaches to performing regression using CBR. It looks in detail at the algorithm proposed by Hanney and Keane, at its limitations and the attempts made to solve them. These limitations lead directly to a set of requirements for an improved approach.

1.4.2 Presentation of new CBR-based regression algorithms

Two new regression algorithms based on CBR are presented. The first extends *k*-NN by aiming for *diversity* among the set of past cases used to make predictions. The second constitutes the main contribution of this thesis. It builds on previous research by Hanney and Keane, McSheery, and others, and is based on the idea that the gap between a query and a similar past case can be bridged by looking at similar pairs of cases stored in the CB. LWLR is used as a heuristic to determine which stored cases are most likely to be useful for solving each individual query.

1.4.3 Implementation and evaluation of a prototype system

The two new regression algorithms were fully implemented and evaluated on a number of standard datasets. Results show that they perform well relative to alternative regression techniques.

1.5 Publications Related to this Thesis

- McDonnell, N. and Cunningham, P.: 2006, A Knowledge-Light Approach to Regression using Case-Based Reasoning, *Proceedings of the 8th European Conference on Case-Based Reasoning (ECCBR 2006)*, pp. 91–105, Springer.

- McDonnell, N. and Cunningham, P.: 2005, Using Case Differences for Regression in CBR Systems, *Proceedings of the 25th Annual International Conference of the BCS SGAI (AI-2005)*, pp. 219–232, Springer.

1.6 Summary and Structure of this Thesis

In brief:

Chapter 2 examines the problem of regression and some traditional approaches to solving it. Chapter 3 introduces CBR and describes its characteristics and problem solving methodology. Chapter 4 shows how CBR can be applied to regression, and introduces a new algorithm that is a variant of *k*-NN. Chapter 5 introduces a second new algorithm for performing regression using CBR. Chapter 6 describes the implementation of the two new algorithms, and Chapter 7 evaluates their performance relative to alternative approaches. Chapter 8 finishes by presenting some conclusions and possibilities for future work.

In more detail:

Chapter 2 considers the problem of regression, the machine learning task that is the focus of this thesis. It examines some traditional and current approaches to regression, focusing on linear regression, *k*-NN, and locally weighted linear regression (LWLR). *k*-NN and LWLR lie at the heart of the new algorithms described in Chapters 4 and 5.

Chapter 3 begins with a presentation of the principles of CBR, the primary technology underlying this research. The process of constructing and operating a CBR system is examined in detail, and the different types of knowledge contained within a CBR system are identified.

Chapter 4 looks at three different ways in which CBR can be used to solve regression problems. The first makes predictions by taking the weighted average of solutions to past cases similar to the query. The second is a new variant that aims for diversity among the cases used for each prediction. The third is that taken in [Hanney and Keane 1996, 1997], and involves generating adaptation rules from the differences between past cases. The limitations of this approach motivate the new algorithm described in Chapter 5.

Chapter 5 constitutes the central part of this thesis. It takes a bottom-up approach to describing a new algorithm for performing regression using CBR. The basic ideas are presented first, then enhancements and modifications are made until the final algorithm is complete. The chapter concludes by examining some potential enhancements to the algorithm that failed to provide improved performance in practice.

Chapter 6 looks at the implementation of a prototype CBR system that includes the regression algorithms described in Chapters 4 and 5.

Chapter 7 evaluates the performance of the two new regression algorithms relative to alternative techniques.

Chapter 8 discusses some conclusions arising out of this research, and briefly outlines some possibilities for future work.

Chapter 2

Regression

All computer systems have a particular task and a set of technologies for achieving it. This research involves constructing a system whose task is *regression* (i.e., predicting the value of a real-valued numeric variable). The term ‘regression’ was first used by Francis Galton in the nineteenth century [Galton 1886]. He found that the heights of children with exceptionally tall or short parents tended towards the societal mean, and referred to this phenomenon as ‘regression towards mediocrity’. Graphing the mean heights of parents and children produced a straight line. This became known as a regression line, and the process of fitting data to such lines as regression [Bland and Altman 1994].

A straight line graph of two numeric variables can also be thought of as a linear model of the relationship between them. Given the value of one, the model can be used to predict a value for the other. Over time, the term ‘regression’ came to be applied more generally to the task of modelling the relationships between any number of numeric variables and using these models to make predictions. Where relationships among variables are linear (as in Galton’s example), this process is known as linear regression. Where relationships are non-linear, the task is called non-linear regression. Examples of both are shown in Figure 2.1. The graph on the left is a linear model fitted to Galton’s original data. The graph on the right is a non-linear model showing counts per minute (CPM) versus hormone concentration in a medical domain. This model also shows how regression models are used to make predictions: given a sample with CPM=1200, the model predicts a hormone concentration of 0.236 micromolar.

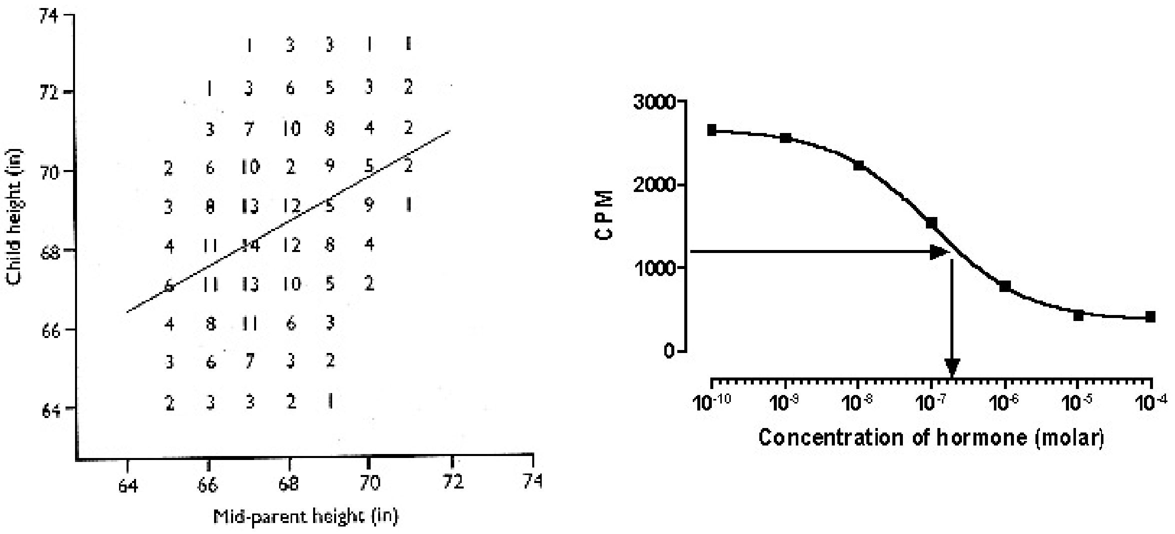


Figure 2.1: Examples of linear and non-linear regression models (linear model from [Bland and Altman 1994]; non-linear model from [Motulsky1999])

Many different approaches to the regression task have been used over the years (see [Uysal and Güvenir 1999] for an overview). The earliest (beginning with Galton) involved fitting data to a simple linear model; this will henceforth be referred to as linear regression. Other approaches include the possibility of modelling non-linear as well as linear relationships. These include non-linear regression [Seber and Wild 1989], locally weighted linear regression, *k*-NN, neural networks [Lippmann 1987], radial basis function networks [Orr 1996], regression trees [Breiman et al. 1984], and model trees. Those that have direct relevance for this research are linear regression, *k*-NN, and locally weighted linear regression.

2.1 Linear Regression

Linear regression estimates the expected value of one numeric variable given the values of one or more others [Draper and Smith 1998]. Many different terms are used for these variables; we will refer to them as the solution and problem attributes respectively. Linear regression is 'linear' in that solution y is assumed to be a linear combination of problem attributes $a_1, a_2, \ldots, a_n$:

$$y = \alpha + \beta a_1 + \gamma a_2 + \ldots + \delta a_n + \varepsilon$$

where α is the intercept and ε is a random error[*]. Given problem attribute values (v_1, v_2, …, v_n) for a new problem, a linear model can be used to predict its target value $\hat{y}$:

$$\hat{y} = \alpha + \beta v_1 + \gamma v_2 + ... + \delta v_n \qquad \textbf{(2.1)}$$

The values of parameters α, β, γ, ..., δ are estimated by the method of least squares [Harter 1983]. Given a set of cases in the form (a_1, a_2, …, a_n, y), this method finds parameters that minimize the total squared difference between actual and predicted solutions. This can be expressed as an error function E:

$$E = \sum_{c \in CB} (y_c - \hat{y}_c)^2$$

One way to minimize the value of E would be to use an iterative gradient descent algorithm. Numerical optimization methods are not needed for linear regression, however—analytical techniques are available that estimate the parameters of a linear model much more efficiently (e.g., QR decomposition [Gentle 1998]). Even for databases of several thousand cases, each with dozens of problem attributes, a linear regression model can be constructed in a fraction of a second. (For non-linear regression models, on the other hand, error functions cannot generally be minimized using algebra and brute force optimization algorithms must be applied.)

Linear regression models are the most widely used of all regression models, and for good reason: they are simple to construct, easy to use, and easy to understand. The theory behind them is well understood, and a range of statistics known as ‘regression diagnostics’ have been developed to provide the user with information about a regression model. Linear models also perform well with small datasets. As a means of modelling data, they are to be preferred over alternative methods when the relationship between problem attributes and the solution is roughly linear.

On the downside, linear regression models are often sensitive to outliers in the data. More importantly, they assume that the target domain can be accurately represented by a linear model. (That is, they assume that the unknown *target function* that maps problem attributes to solutions is a linear function.) This assumption holds in a surprisingly high proportion of real-world domains, but where it does not, alternative regression techniques must be used.

Linear regression is an example of an *eager* learning algorithm: it accepts a set of historical cases stored as vectors of numerical values, and uses them to construct a global (linear) model that is used to predict solutions for new cases. The two regression algorithms examined below, *k*-NN and LWLR, are both *lazy*, *instance-based* approaches. Old cases are not used to construct a global model, but instead are simply stored together in a case base (CB). When a

[*] Throughout this thesis, scalars are represented in italics (*x, y, z*) and vectors in bold (**x, y, z**).

new query case is received, a local approximation to the global target function is constructed in the area of domain space surrounding the query. This is then used to predict the target value.

Eager approaches perform most processing during an initial training phase during which a global model is constructed. Predictions based on this global model can then be made very quickly. Lazy approaches, on the other hand, are computationally expensive each time a query is received (this is becoming less of a problem as the cost of computation declines). They have two compensating advantages, however: first, learning is extremely straightforward in that it simply involves storing new cases, and second, the query itself can be taken into consideration when the target function is being approximated. This allows the problem solving process to be tailored to the precise needs of each individual query.

2.2 *k*-Nearest-Neighbour (*k*-NN)

The *k*-nearest-neighbour (*k*-NN) algorithm is the simplest instance-based approach to regression (for an introduction to *k*-NN, see [Aha at al. 1991]). It is also one of the oldest and best understood algorithms in machine learning, dating back at least to the mid-1950s (early results are presented in [Duda and Hart 1973]). In common with linear regression, it assumes that all cases are represented as a set of numeric problem attributes A and a single numeric solution y. If all cases are plotted as points in multi-dimensional domain space $\mathfrak{R}^n$, the Euclidean distance between any two cases $\boldsymbol{\tau}$ and $\boldsymbol{\rho}$ can be calculated as

$$d(\boldsymbol{\tau},\boldsymbol{\rho}) = \sqrt{\sum_{a \in A} (\tau_a - \rho_a)^2} \qquad \textbf{(2.2)}$$

When presented with a new query case, the *k*-NN algorithm predicts its target value as follows:

1. The k nearest neighbours to the query are retrieved, where nearest neighbours are those cases whose Euclidean distance to the query is shortest;
2. The predicted target value for the query, $\hat{y}$, is calculated as the mean of the solutions among neighbouring cases:

$$\hat{y} = \frac{\sum_{c \in NNs} y_c}{k} \qquad \textbf{(2.3)}$$

where *NNs* is the set of k nearest neighbours to the query.

Different metrics have been used to calculate the distance between cases. The Manhattan distance is a popular alternative to the Euclidean distance:

$$d(\boldsymbol{\tau},\boldsymbol{\rho}) = \sum_{a \in A} abs(\tau_a - \rho_a)$$

The choice of distance function usually makes little difference, and the Euclidean distance is used in experiments involving *k*-NN in Chapter 7.

Obviously, the accuracy of predictions depends critically on the value assigned to *k*. Typical values used in practice are 1, 3 and 5. 1-NN is a very simple algorithm in which a query is assigned the solution of its nearest neighbour. It serves as a useful baseline when comparing the performance of different regression algorithms. Performance usually improves with higher values of *k*, reaching a peak at a particular value before declining again [Duda and Hart 1973]. Using values greater than 1 adds diversity to the problem solving process and helps to smooth the impact of noisy cases. In the experiments in Chapter 7, a suitable value of *k* for each dataset is found using cross-validation.

As presented above, the algorithm assigns equal importance to the contribution from each of the query's neighbours. Results are often improved, however, if the contribution from each case is weighted by its distance from the query so that cases closest to the query have greatest influence. This is achieved by assigning a weight to each neighbouring case using a *weighting function* (also called a *kernel function*). Many different weighting functions have been used—a number of them are shown graphically in [Atkeson et al. 1996]. The following are among the most common:

$$w = \frac{1}{d(\boldsymbol{\tau},\boldsymbol{\rho})}$$ Inverse distance weighting

$$w = \frac{1}{d(\boldsymbol{\tau},\boldsymbol{\rho})^2}$$ Inverse squared-distance weighting

$$w = e^{-d(\boldsymbol{\tau},\boldsymbol{\rho})^2}$$ Gaussian weighting

The optimal weighting function can be found for a particular problem domain using cross-validation [Howe and Cardie 1997]. (See [Wettschereck et al. 1997] for a survey of different attribute weighting methods.) Gaussian weighting generally performs well in all domains, particularly in the presence of noisy data; for this reason, it is used in all experiments involving *k*-NN in Chapter 7.

Equation 2.3 can be modified to take the weights of neighbouring cases into account when predicting the target value for a query:

$$\hat{y} = \frac{\sum_{c \in NNs} w_c \, y_c}{\sum_{c \in NNs} w_c} \qquad \textbf{(2.4)}$$

Note that the *k*-NN algorithm relies on the assumption that cases whose problem attributes are similar are likely to have similar solutions. This is known as the *similarity assumption* (also referred to as the *similarity hypothesis* in [Rendell 1986]), and is shared by case-based reasoning. In domain space $\Re^n$, the assumption holds if the target function is continuous and if domain space is reasonably smooth in the area surrounding any particular case. Assumptions such as these are referred to as *inductive bias*—they allow machine learning algorithms to generalize beyond the specific training examples they are given [Mitchell 1997, Section 2.7]. Linear regression, for example, has as its inductive bias the assumption that the target domain can be accurately represented as a linear model. Locally weighted linear regression (described below) has a similar inductive bias to *k*-NN—it assumes a continuous target function that is reasonably smooth in local areas of domain space.

Note also that the accuracy of a prediction depends on the retrieval of stored cases that are most similar to the query case. Similarity is calculated using the distance formula in Equation 2.2, which simply adds together the differences between pairs of problem attribute values. If each case contains problem attributes that are not correlated with the target attribute in some way, the distance function will be misled and will retrieve cases that are not very predictive. We might re-define 'nearest neighbours' to mean those cases most similar in the attributes that are most useful for predicting target values. One solution to the problem of irrelevant attributes is to remove them using an attribute selection algorithm [Kohavi and John 1997a]. An alternative solution is to weight each attribute by its predictive ability, so that less predictive attributes receive lower weights. The set of weights can be optimized for any particular dataset using, for example, a genetic algorithm. This approach can be problematic because it tends to overfit the training data. Choosing each attribute's weight from a reduced set of discrete values (e.g., {0, 0.5, 1}) may yield better performance on unseen data than allowing weights to take any real value [Kohavi et al. 1997b]. See [Aha 1992] for further discussion on the topic of dealing with problematic attributes.

A simple example of *k*-NN in action is shown in Figure 2.2. Here, solution *y* is a function of a single problem attribute *a*. Parameter *k* is set to 3, and a Gaussian weighting function is being used. When presented with query Q, the algorithm first retrieves its 3 nearest neighbours (i.e., those 3 cases whose values for *a* are closest to that of Q). It then predicts a solution for Q using Equation 2.4.

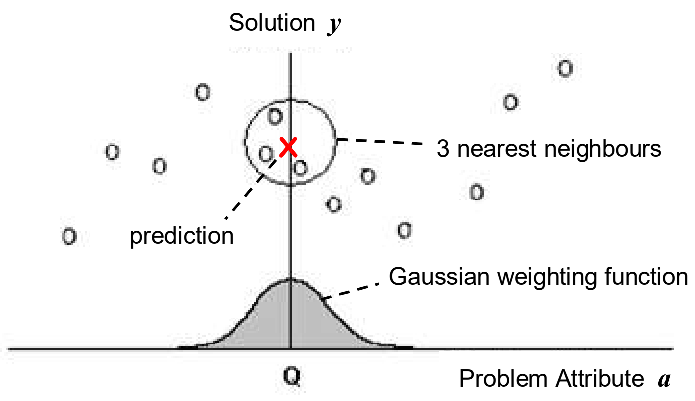


Figure 2.2: Predicting a numeric value using *k*-NN

As a prediction algorithm, *k*-NN has several advantages. Chief among them is that fact that it makes no assumptions about the form of the underlying target function, and so is suitable for use in domains where the target function is complex; linear regression and *k*-NN complement one another in this respect. (Algorithms utilizing models that make no assumptions about the statistical distribution underlying a dataset are referred to as *nonparametric* [Noether 1984].) *k*-NN is also simple to use and understand. Its main disadvantage is that it assumes that cases quite similar to the query can always be retrieved—this will not apply if data is sparse. Other potential disadvantages have already been discussed: it can be computationally expensive to solve each query, and the algorithm is sensitive to irrelevant problem attributes. Sufficient processing capacity and appropriate attribute selection may alleviate these, however.

2.3 Locally Weighted Linear Regression (LWLR)

Locally weighted linear regression (LWLR) offers an alternative instance-based approach to regression [Atkeson at al. 1996; Fan and Gijbels 1996]. LWLR can be seen as a combination of linear regression and *k*-NN. It is based on the idea that even highly non-linear target functions can be approximated *locally* by linear models, in the same way that any curve can be approximated by a series of short line segments joined together. Fitting polynomial functions to local subsets of a dataset dates back to the beginning of the 20th Century [Cleveland and Loader 1995]. More recently, the technique has been implemented in the LOESS (or LOWESS) algorithm [Cleveland 1979; Cleveland and Devlin 1988], which allows the user to specify the degree of polynomial to fit to a set of data. Here, we are only concerned with polynomials of degree one (i.e., linear models)—these are most widely used in practice.

When presented with a query case, LWLR predicts its target value as follows:

1. A set of nearest neighbours to the query is retrieved;
2. A local linear model is constructed from these cases, where the contribution from each neighbour is weighted by its distance from the query.
3. The linear model is used to predict a target value for the query.

The number of neighbours (k) to retrieve and the behaviour of the distance weighting function are closely related to one another, since there is no need to retrieve cases beyond a distance where they cease to have a significant influence on the linear model produced. The weighting function often incorporates a smoothing parameter (λ) that determines how rapidly a case's weight declines with distance from the query. This parameter is particularly important for weighting functions that reduce weights to zero (or close to zero) after a short distance, since incorrect values may yield too few (or even no) cases with which to construct an accurate local linear model. For example, the *triangular kernel* assigns zero weight to all cases whose distance from the query is greater than or equal to 1:

$$w = \begin{cases} 1-d & \text{if } d < 1 \\ 0 & \text{otherwise} \end{cases} \qquad \text{Triangular weighting}$$

A smoothing parameter can be introduced to change the distance at which weights reach zero:

$$w = \begin{cases} \lambda-d & \text{if } d < \lambda \\ 0 & \text{otherwise} \end{cases}$$

Parameter λ may be assigned any value that improves predictive performance. One approach is to set λ equal to the distance to the k^{th} nearest neighbour—this is called *nearest neighbour bandwidth*, and has the advantage that the radius of the weighting function decreases automatically as the number of cases in the CB increases. A simpler approach is to assign a fixed value to λ—this is referred to as a *fixed bandwidth* [Fan and Marron 1993]. When using triangular weighting, for example, λ may be set to the maximum theoretical distance between cases, so that weights decline linearly with distance from the query but never reach zero (this is called *similarity weighting* in the Weka machine learning program [Witten and Frank 2000]).

Several other factors also influence the choice of k. In more highly non-linear domains, a lower value may help avoid excessive smoothing and give better results. Noisy datasets, on the other hand, may benefit from greater smoothing and a higher value for k. The number of problem attributes is also relevant. For a dataset with 10 problem attributes, for example, each linear model will have 11 parameters and at least this many neighbours will be required to assign a unique value to each. In the experiments involving LWLR in Chapter 7, the value of k is set by cross-validation for each dataset, and the Gaussian weighting function (described in Section 2.2) is used with a fixed bandwidth.

Constructing a linear model involves finding parameters for the model given in Equation 2.1. The method of weighted least squares is used for this purpose [Carroll and Rupport 1988]. This is an analytical method that operates very efficiently; its goal is to minimize the total squared difference between actual and predicted solutions, where each case is weighted by distance so that the best fit is provided for cases closest to the query. As with linear regression, this amounts to minimizing the value of an error function *E*:

$$E = \sum_{c \in NNs} (y_c - \hat{y}_c)^2 \, w_c$$

Once the parameters for a local linear model have been estimated, the query's target value is predicted using Equation 2.1. The linear model is specific to the query in question, and so is discarded once it has yielded a prediction. A new model is built for each new query that is received.

The operation of LWLR is shown in Figure 2.3. As before, solution *y* is a function of a single problem attribute *a*. The number of nearest neighbours to retrieve (*k*) is set to 5, and a Gaussian distance weighting function is being used. Upon receiving query Q, the algorithm begins by retrieving its 5 nearest neighbours. It uses the method of weighted least squares to construct a local linear model, and uses this model to predict a solution for Q.

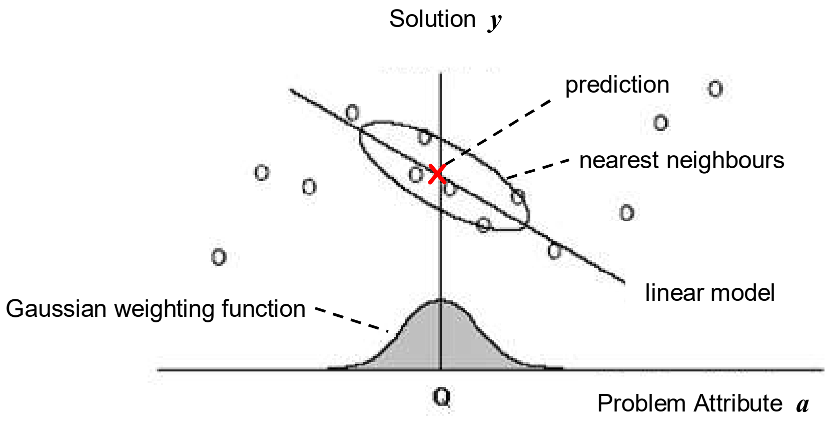


Figure 2.3: Predicting a numeric value using LWLR

LWLR shares the advantages of *k*-NN as a regression technique. It is suitable for use in domains where the target function is too complex to be represented by a simple model (e.g., a global linear regression model). It is also straightforward to use and understand, and excellent implementations such as LOCFIT [Loader 1999] are publicly available on the Internet. LWLR also shares some of *k*-NN's disadvantages: it is even more computationally expensive, and doesn't work well with sparse datasets.

2.4 Which Regression Algorithm to Use?

All three algorithms described above—linear regression, *k*-NN, and LWLR—are state-of-the-art regression algorithms that are widely used for data mining. The optimal choice for any domain will depend on a number of factors; the following are among the most important:

- Can the target function be modelled by a simple linear model? If so, linear regression is the preferred choice; otherwise, *k*-NN or LWLR may perform better.
- Are there plenty of historical cases to reason from? If not, linear regression is again the preferred choice, since it works well with limited data.
- What computational resources are available for each prediction? With linear regression, predictions can be made by hand once the model parameters have been estimated. LWLR and *k*-NN have become more popular in recent years as the considerable processing capability required for each prediction has become more widely available.

Each learning algorithm has its own bias, and will perform well in domains that conform to that bias. This is expressed in the 'no free lunch' theorem [Wolpert and Macready 1997], which states that for any algorithm, an elevated performance over one class of problems is offset by reduced performance over another class. In other words, no single algorithm will perform best in all problem domains. When testing regression techniques in Chapter 7, therefore, a number of datasets with different characteristics are used to give a balanced view of each algorithm's performance in different conditions.

2.5 Use of Regression Algorithms in this Thesis

The algorithms described above re-appear in the following contexts in this thesis:

- ***k*-NN** provides the basis for case-based reasoning (CBR), which is described in Chapter 3. CBR systems traditionally perform regression on the same basis as *k*-NN, that is, by taking the (weighted) mean of solutions from the query's nearest neighbours. This approach will be referred to as CBR-Basic (abbreviated to **CBR-B**), and is described in Chapter 4 together with two alternative ways to use CBR for regression. The first of these is a new approach that aims to maximize diversity when choosing a set of nearest neighbours; this is referred to as CBR-Diverse (**CBR-D**). The second takes a multi-strategy approach by combining *k*-NN with a rule induction algorithm

[Hanney and Keane 1996, 1997]. This will be referred to as CBR-AdaptationRules (**CBR-AR**).

- **LWLR** lies at the heart of the new regression algorithm (described in Chapter 5) that is the primary contribution of this thesis. This algorithm is based on CBR and is referred to as CBR-CaseDifferences (**CBR-CD**). In addition, LWLR is one of a number of algorithms that are experimentally compared in Chapter 7.
- **Linear regression** provides the theoretical basis for LWLR, and also appears in experiments in Chapter 7.

One further regression algorithm appears in the experimental evaluation: **model trees** [Wang and Witten 1997; Quinlan 1992] extend decision trees so that they apply to regression tasks and are described below.

Figure 2.4 shows the regression algorithms listed above and the relationships between them. Those with a shaded background are relevant to this thesis, while those with a textured background (i.e., CBR-D and CBR-CD) are new.

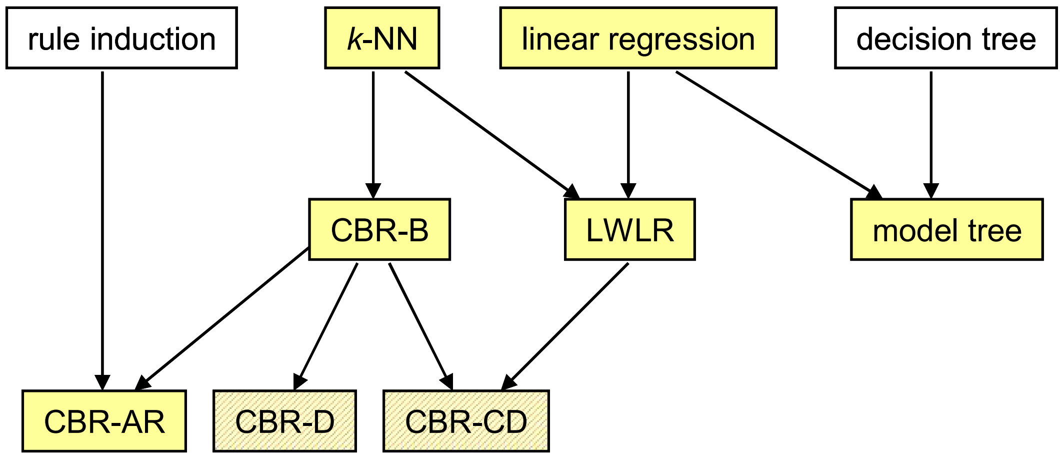
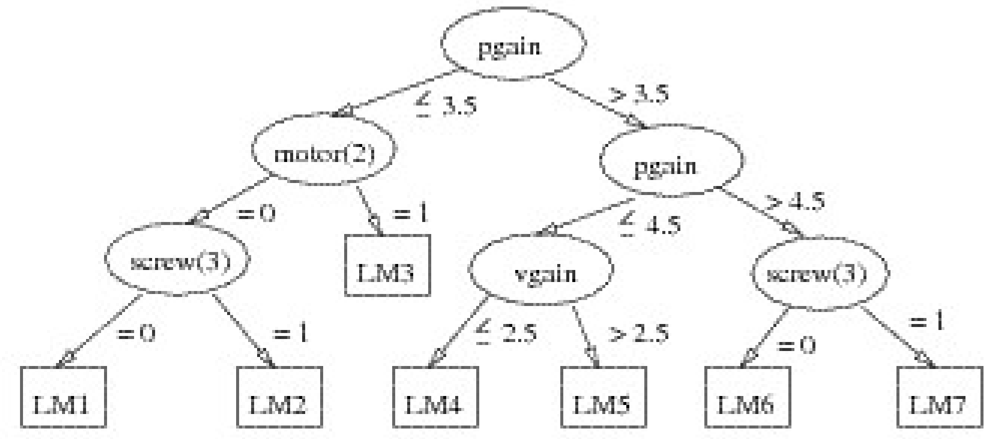


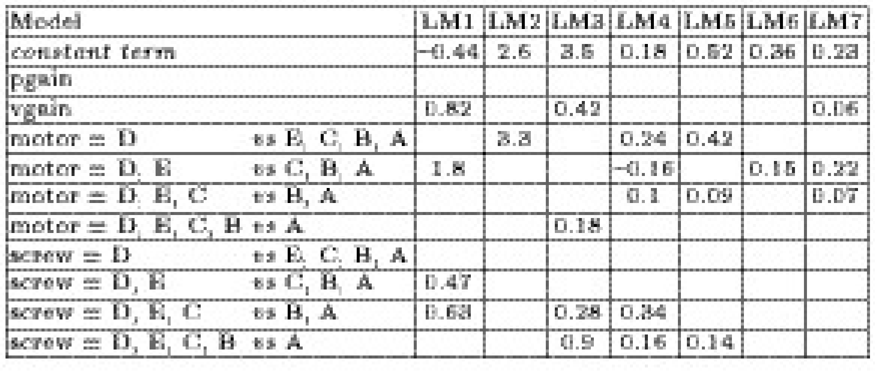
Figure 2.4: Regression algorithms used in this thesis

2.5.1 Model trees

Decision trees have been used for classification for many years [Breiman et al. 1984; Quinlan 1993]. Despite the sophistication of the mechanisms employed during decision tree construction for branching, pruning, etc., completed decision trees are extremely easy to use and understand. They also provide a useful insight into the global structure of a problem domain by decomposing it in a top-down manner. Decision trees cannot be used directly for regression tasks, however—with a real-valued solution attribute, each historical case will likely have a unique solution. Straightforward application of the decision tree algorithm therefore produces a tree with a leaf node for each case, resulting in an overlarge tree with no inductive bias.

Regression trees and model trees take a common approach to addressing this problem. Their basic tree construction algorithm is as follows: branch on problem attributes in the upper part of the tree, and end with a constant or linear regression formula at the leaves. Branching points in the upper tree are chosen to maximize the reduction in standard deviation among solutions on the two sides of the split. In other words, cases with similar solutions end up on the same branch. After a certain number of branching points have been created, the remaining cases' solutions on any branch will all be quite similar. This is the stage at which regression trees and model trees differ. Regression trees end with a leaf node that contains the mean solution for the remaining cases. Model trees end with a leaf node containing a linear model in the remaining attributes for the remaining cases' solutions. Figure 2.5 shows an example of a model tree for the Servo dataset, one of those used in experiments in Chapter 7.





Model		LM1	LM2	LM3	LM4	LM5	LM6	LM7
constant term		−0.44	2.6	3.6	0.18	0.52	0.36	0.23
pgain								
vgain		0.82		0.42				0.06
motor = D	vs E, C, B, A		3.3		0.24	0.42		
motor = D, E	vs C, B, A	1.8			−0.16		0.15	0.22
motor = D, E, C	vs B, A				0.1	0.09		0.07
motor = D, E, C, B	vs A			0.18				
screw = D	vs E, C, B, A							
screw = D, E	vs C, B, A	0.47						
screw = D, E, C	vs B, A	0.63		0.28	0.34			
screw = D, E, C, B	vs A			0.9	0.16	0.14		

Figure 2.5: Model tree and linear models for the Servo dataset (from [Wang and Witten 1997])

Model trees share many of the benefits of decision trees. A model tree represents a global model of the target domain that is simple enough to be applied by hand if necessary. The target domain is partitioned into different regions, each represented by a linear model—this allows the technique to be used in domains that are not globally linear. Model trees can be thought of as an eager version of LWLR, where linear models are created based on the characteristics of the training data rather than those of the query. Chapter 7 uses the M5′ implementation of model trees [Witten and Frank 2000, pp. 201–208].

2.6 Summary

This chapter looked at some standard approaches to predicting the value of a numeric variable (i.e., *regression*):

Linear Regression: A single linear model is fitted to a set of numeric data points and used to predict solutions for new instances.

***k*-NN:** For each new data instance, a number of similar stored instances are retrieved and their solutions averaged to make a prediction.

LWLR: A combination of linear regression and *k*-NN. For each new data instance, a local linear model is constructed from a set of similar instances and used to make a prediction.

Model Trees: A combination of decision trees and linear regression. A tree is produced for each dataset with local linear models at its leaves instead of classification labels.

Linear regression and model trees are *eager* approaches in that domain models are constructed during an initial training phase. The other two are described as *lazy* because they defer problem solving until a new problem is received. Each algorithm has its advantages and disadvantages and none is best in all circumstances; the optimal approach will depend on the characteristics of the domain and the dataset, as well as other requirements such as usability and efficiency. Of the algorithms examined, *k*-NN and LWLR are most relevant to this research as they provide the basis for the CBR-based regression algorithms described in Chapters 4 and 5.

Chapter 3

Case-Based Reasoning

The goal of this research is to construct an effective regression system using case-based reasoning (CBR). As a technology, CBR is a relatively recent invention. As a means of solving problems, however, it is among the oldest and most intuitive types of reasoning. The basic underlying idea is simple: given a new problem to solve, find a similar problem that was solved in the past and reuse its solution. Many researchers have explored this from a cognitive science perspective, and have argued that it constitutes a plausible model of the human reasoning process [Schank 1982; Kolodner 1993]. Most people solve new problems by *remembering* similar problems from the past and then re-applying an old solution that meets current needs (perhaps with some alteration). The importance of prior examples has been demonstrated in all domains of human reasoning from the simplest to the most sophisticated, including mathematics [Faries and Schlossberg 1994] and medicine [Schmidt et al. 1990].

Early research into CBR arose out of a desire to represent this reasoning process in a computer model. The cognitive roots of CBR were explored at Yale University by Roger Schank and his colleagues [Schank and Abelson 1977; Riesbeck and Schank 1989]. Researchers in the AI community built on this research to develop CBR as a practical problem solving methodology (e.g., [Aamodt and Plaza 1994]). CBR has been implemented in a multitude of different systems with an enormous variety of system tasks and technologies. Underlying all of them is the simple idea of directly reusing past experiences to solve new problems. The effectiveness of this approach is based on the idea that 'similar problems have similar solutions' [Leake 1996]. As discussed in Section 2.2, this has been called the *similarity assumption* and is equivalent to saying that the world is *regular*. In the context of numerical regression domains that are the target of this research, it means that domains are assumed to be continuous and not overly non-linear in local regions. This assumption constitutes the *inductive bias* of CBR: when solving a new problem, the solutions from similar past problems should provide a useful starting point.

Simplicity is only one of several advantages that have contributed to CBR's popularity over the past twenty-five years, and a number of others are presented in the following section. The new regression algorithms presented in Chapters 4 and 5 benefit from these advantages and are implemented as pure CBR systems.

3.1 Advantages of CBR

CBR's advantages over alternative AI problem solving techniques can be summed up as follows:

> *CBR systems are easier to construct and maintain, and offer improved performance during operation.*

These two aspects of CBR systems are discussed below.

3.1.1 Simpler system construction and maintenance

Traditional problem solving systems in AI have relied on the presence of explicit, domain dependent knowledge that is acted upon by a separate reasoning element. This broad architecture for intelligent systems is among those proposed in [Turing 1950], and systems built along these lines are known as *knowledge-based systems* [Aamodt 1993]. The importance of specific domain knowledge, often explicitly represented in symbolic form, is encapsulated in the 'Knowledge Principle':

> A system exhibits intelligent understanding and action at a high level of competence primarily because of the specific knowledge that it can bring to bear: the concepts, facts, representations, methods, models, metaphors, and heuristics about its domain of endeavour. [Lenat and Feigenbaum 1991]

This principle is in tune with one's intuition that in order to speak sensibly on a given topic, some knowledge of the topic is highly desirable. Acquiring knowledge is a difficult and time-consuming process for humans. Some computer systems operate in environments where knowledge may be easier to come by; a system controlling a physical process such as a production line, for example, can continuously monitor its status through sensors. However, most AI systems operate in domains that require an expert level of human-like knowledge; examples include decision support systems in the high-tech or medical domains. The explicit domain knowledge required by these systems has traditionally been acquired in two stages: first, knowledge engineers obtain it from domain experts; second, it is converted into a machine-readable format (e.g., a PROLOG rule-base). This process is fraught with difficulties, among them the following:

- Is the system's level of knowledge adequate to allow it to perform its task to the standard required? And how can this question be answered satisfactorily?
- Is the knowledge accurate and consistent? Domain experts may make erroneous and contradictory statements, and mistakes can occur during both stages of the knowledge transformation process. Again, ascertaining the answer to this question is problematic because domain experts are often not qualified to inspect or edit the knowledge in its machine-readable form.
- Can an expert's experiences be conveniently represented in an explicit form such as a set of rules? Doctors, for example, rely on their many years' experience when making decisions and often cannot list all the factors involved or the precise reasoning process followed. 'Instinct', 'intuition', 'gut feeling'—these may play a crucial part in distinguishing expert from non-expert levels of decision making, yet representing them explicitly may be difficult or impossible.
- Once knowledge has been acquired and the system constructed, how easy is it to maintain? All real-world computer systems must be continuously updated or become obsolete [Lehman 1985]. In a knowledge-based system, faulty or out-of-date knowledge must be removed and new knowledge added on an ongoing basis. Does this require regular iterations of the knowledge acquisition process, with knowledge engineers and domain experts performing a full audit of the system's knowledge store? Such an onerous maintenance process would not be viable for most decision-support systems.

Summing up, then, knowledge elicitation is a difficult and uncertain process for knowledge-based systems (see [Forsythe and Buchanan 1989] for further discussion on this topic). Perhaps the chief advantage of CBR is that it can ease both the acquisition and maintenance of domain knowledge.

Knowledge acquisition during system construction: CBR systems reason from complete cases, where each case encapsulates a single problem solving experience. In many domains, knowledge is already stored as a set of prior examples (i.e., cases) or can readily be converted to this format through a process of *case engineering*. In the medical domain, for example, prior cases may already be available as patient records. Instead of asking doctors exactly how they reach their decisions, it may be easier simply to work from a set of cases where correct decisions were made. Two caveats apply here: first, a case format may not be suitable for some domains (e.g., where knowledge is available only in natural language); second, CBR systems need additional retrieval and adaptation knowledge to operate successfully (see Section 3.3). Even taking these into account, however, CBR's approach to storing knowledge in cases often simplifies knowledge acquisition considerably.

Knowledge maintenance during system operation: Most knowledge in a CBR system is contained in the set of prior cases stored together in a case base (CB), where each case contains details of a single past problem and its solution. It is a straightforward matter for a domain expert to examine such a case and determine whether the problem description is inconsistent or the solution inappropriate. This accessibility allows users of a CBR system to remove faulty cases and add new ones without the aid of a knowledge engineer. This not only makes routine maintenance much easier, but also allows systems to be deployed with a minimal set of initial seed cases; additional cases can then be added by users as they become available. Note that routine removal of obsolete cases and addition of new ones allow CBR systems to adapt naturally in the presence of *concept drift*, a characteristic of some domains whereby the types of cases encountered changes over time [Widmer and Kubat 1996]. One domain subject to concept drift for which CBR is a natural fit is the filtering of spam email [Cunningham et al. 2003a].

3.1.2 Improved performance during operation

Ease of construction and maintenance are excellent characteristics for any AI system, but problem solving abilities are at least as important. After all, the simplest system to deploy and maintain will be useless if it offers poor performance in operation. Three criteria can be used to judge the *quality* of the solution offered by an AI system in response to a problem:

1. How close is the solution to the optimal solution that would be found by a committee of domain experts, or by a computer system that performed an exhaustive search?
2. How efficiently is the solution found? The level of efficiency required will vary with the application domain. For example, a helpdesk operator may need a response within a few seconds, whereas a computer chip designer may be prepared to wait overnight for a better response according to criterion 1.
3. Is the solution trusted and accepted by the user? Again, the importance of this factor will depend on the problem domain. Some users may simply accept an answer from the system and apply it without further thought. Such users/systems are the exception rather than the norm, however. AI systems are generally deployed in a supporting role

to human users, and if their conclusions are not trusted, they will (not unreasonably) be ignored. Trust may be established in two ways:

- The process by which a solution is found can be laid bare to allow the user to verify that the reasoning process is sound;
- Alternatively, a reasonable explanation or justification for a solution can be presented to the user.

The first approach is taken by expert systems that present users with the rules used to solve a problem. The chain of reasoning used is not always convincing (or even comprehensible) to the user, however [Moore 1994, p. 31; Barlizay et al. 1998]. The second approach may be taken if the problem-solver uses a reasoning process that is simply not accessible to humans (e.g., an artificial neural network). Once the solution has been arrived at, a second process is initiated to find an explanation that convinces the user that it is correct. This explanation may be presented as a set of rules, for example [Andrews et al. 1995].

The nature of CBR allows it to perform well according to all three of these performance criteria:

High quality solutions: In certain domains, reasoning from cases offers a distinct advantage over more abstract reasoning. It has already been pointed out that some domains may be extremely difficult to model using formal knowledge representations such as rule-sets. Domains in which relationships between important concepts are uncertain are known as *weak-theory* domains [Porter 1989]. Uncertainty may be due to the fact that some parts of the domain are not observable, or because aspects of the domain change over time or in response to certain events. Examples of weak-theory domains include medicine and law. Where rules are inadequate to model complex and uncertain domains, reasoning from cases may offer better solutions than logical inference from generalized models [Porter et al. 1990]. Presented with a new problem, a CBR system will retrieve stored cases that show precisely how similar problems were solved in the past. These past solutions may be presented directly to the user, or may provide the starting point for generating a solution that more closely fits the circumstances of the new problem.

Efficient problem solving: As discussed above, CBR systems begin problem solving by retrieving cases similar to the problem being tackled. Problem solving therefore does not proceed from first principles, but builds on instances of successful problem solving from the past. This allows CBR to be used in complex domains where finding a solution from first principles may be NP-hard, for example, solving planning problems [Spalazzi 2001] or synthesising new drugs [Craw 2001].

Convincing explanations: Any computer system not trusted by its users will almost certainly fail. CBR systems have a natural advantage in instilling trust in that they can accompany each solution with the actual past cases used to derive it. Users can verify for themselves the degree of similarity between these past cases and the current one. If they are convinced that past and current problems are similar, they are likely to accept a solution similar to and derived from the solutions to these past cases. If they do not accept that the retrieved and current problems are similar, they are free to reject the proposed solution or treat it with some caution. Either way, presenting past cases to users gives a high degree of insight into the problem solving process and allows people to decide their own level of confidence in each solution [Cunningham et al. 2003b]. This is particularly useful when the suggested solution is uncertain, perhaps because similar past cases contain contradictory solutions. In this case, the user can evaluate the available evidence and decide what to do.

Two additional aspects of presenting cases as explanations are interesting to note:

- Cases may be used to provide explanations for solutions found using black-box systems such as neural networks or support vector machines [Nugent and Cunningham 2004]. CBR is then used as an explanation system rather than a problem-solver.
- Cases that are most convincing to the user may not be those most similar to the new case. In classification tasks, for example, cases between the new case and the *decision surface* may prove most compelling [Doyle et al. 2004].

3.1.3 When to use CBR

It was mentioned in Section 2.4 that all problem-solvers have a bias that makes them more suitable for certain types of problems than others. Listing the advantages of CBR serves to highlight the characteristics of those domains where CBR can most usefully be applied:

Complex, weak-theory domains: CBR is most suited to complex domains in which the relationships between concepts are uncertain. Where a domain can be completely represented by a generalized model, alternative approaches may perform better.

Incomplete knowledge at time of construction, and ongoing learning required: CBR systems can be deployed with a minimal set of cases, and can learn incrementally by acquiring new cases during operation. Where a domain can be fully specified during construction, this advantage is nullified.

Users' trust in the system is important: The ability to provide concrete examples by way of explanation may be CBR's most significant advantage. It makes CBR particularly well-suited to tasks that require complete transparency in the reasoning process, for example, decision support systems that suggest medical diagnoses based on patients' symptoms [Schmidt and Gierl 2001].

CBR is a flexible approach that can accommodate many different problem tasks and domains; [Bartsch-Spörl et al. 1999] and [Aamodt and Plaza 1994] survey some of them. One application that has seen extensive growth in recent years is the use of CBR in recommender systems on the World Wide Web [Lorenzi and Ricci 2005]. Typical system tasks include assisting users in their choice of books, music, flights, restaurants, etc. Conversational recommender systems use user feedback to iteratively narrow down the search for a satisfactory product. The problem domain (i.e., human users choosing goods and services) matches all of the characteristics listed above, and provides an excellent example of where CBR is most useful. It is a complex, weak-theory domain in that users' preferences cannot be precisely modelled, and indeed may change over time. The case base can be initialized with a preliminary set of products, and updated continuously as new items are introduced and others become out of date. Finally, trust is important because users will not use a recommender system unless they believe that its suggestions are helpful and unbiased.

3.2 CBR Problem Solving Methodology

CBR solves new problems by re-using solutions from similar past cases. It takes a *lazy* approach to problem solving in that it does not generalize beyond the specific cases in the CB until asked to solve a new problem (this new problem is known as a *query case*). In particular, CBR demonstrates the following behaviour that is characteristic of all lazy problem-solvers [Aha 1997]:

Deferred problem solving: CBR systems delay processing of their inputs (i.e., new and past cases) until a query case is received.

Data-driven problem solving: Query cases are solved by combining solutions from prior cases.

Temporary results are discarded: Temporary results created while solving a query case are discarded—problem solving is tailored to the individual needs of each new query.

Eager systems take an alternative approach. They do not work directly with past examples to solve new problems, but instead compile these experiences into a global model during an initial training phase. Examples of eager problem-solvers include neural networks, decision trees, rule-sets, and linear regression models. Lazy and eager systems have complementary advantages and disadvantages, and many of the advantages of CBR presented in Section 3.1.2 are due to its lazy nature.

Having looked at the high-level motivations behind CBR, its advantages and appropriateness for different domains, we turn now to the mechanics of actually constructing and using a CBR system. This is fundamentally an engineering problem that can be tackled in many different ways. Based on researchers' experience of CBR over the years, however, the process has been distilled into a series of logical steps (see Figure 3.1). These can usefully be divided into two broad phases:

Construction phase: Comprises all of the actions needed to bring a CBR system into service.

Operation phase: The process that is followed to solve each new problem (this is referred to as the 'problem solving cycle' in [Aamodt and Plaza 1994]).

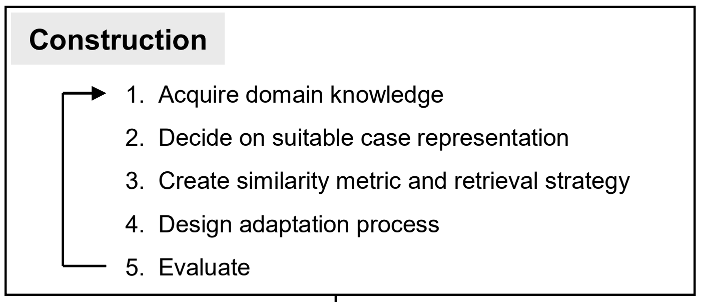


Figure 3.1: Construction and operation of a CBR system (problem solving cycle from [Aamodt and Plaza 1994])

3.2.1 Construction phase

CBR's problem solving process was summarized above as follows: "CBR solves new problems by re-using solutions from similar past cases". This sentence immediately suggests a number of mechanisms that must be in place before a CBR system enters operation:

- Case representation
- Initial set of cases

- Similarity metric and retrieval strategy
- Adaptation process

These aspects of CBR system construction are examined in the following sub-sections.

3.2.1.1 Acquisition of domain knowledge

Design of a CBR system begins with a detailed examination of the problem domain. The system task must be clearly specified and CBR's suitability for achieving it confirmed. Domain experts may be interviewed to ascertain how problem solving was carried out in the past. Historical data based on existing, perhaps manual, systems must be collected and analysed.

We will not examine this aspect of system construction in detail, since it is common to all knowledge-based systems. The output of this phase is a clear idea of the system's task and a rough idea of the knowledge that will be required to support it. Sufficient historical data should also have been collected to enable the assembly of an initial set of cases.

3.2.1.2 Case representation

Deciding exactly what to store in each case may involve considerable case engineering effort. Each case generally comprises two parts: a *problem part* containing a description of a single past problem, and a *solution part* containing the solution that was eventually applied. The problem part should contain all relevant information about the past situation. It should be *sufficient* to enable the solution to be arrived at by a process of reasoning, so that a human expert would arrive at the same or a very similar solution. Both problem and solution parts are often represented as a set of case *features* or *attributes*, where each attribute has a particular type (e.g., numeric, string, Boolean, nominal). More complex attribute types are also possible; a quadratic-equation attribute might be represented as a set of three parameters, for example, or a graph attribute as a set of nodes and edges. ([Aha and Wettschereck 1997] describes a number of different case representation approaches.) Problem and solution attributes together constitute a case structure that is represented in software as a data structure or data type. Each case is then stored as an instance of this type.

Potential attributes are often evaluated by a process of *attribute* (or *feature*) *selection* [John et al. 1994] to determine which are most useful in reasoning towards the solution. Eliminating irrelevant attributes serves the dual purpose of minimizing case size and maximizing the applicability of stored cases to new problems. Note that the optimal set of attributes must be *sufficient* but not *necessary* for a solution to be found. It should have minimal size while also containing attributes that make sense to users (in order to maximize confidence in the system's operation). The relationship between problem attributes and the solution is one of correlation rather than causation—changes in problem attributes should correspond to changes in the solution without necessarily causing them. The hypothetical target function f that maps problem attributes a_i to a solution y (i.e., $y = f(a_1, a_2, ..., a_n)$) is *surjective* because each valid set of problem attribute values maps to a single solution and different sets of attribute values can share the same solution.

Once a suitable case representation has been arrived at, the case base can be populated with an initial set of cases. These can be used as a test-set to help tune the retrieval and adaptation strategies and to confirm that they meet their performance requirements.

3.2.1.3 Similarity metric and retrieval strategy

Solving a new query begins with the retrieval of similar past cases. The retrieval process has two components: first, past cases must be assessed to determine their similarity to the query; second, similar cases must be located in a timely and efficient manner. The first component involves defining a *similarity metric*, that is, a function that determines the distance between any two cases. Armed with a suitable metric, a CBR system can retrieve past cases most similar to a query—these are referred to as the query's *nearest neighbours*. The second component entails deciding on a suitable storage structure for the CB, and a suitable search strategy for locating cases similar to the query.

Designing a good similarity metric involves answering the following question: What does it mean to say that two cases are *similar*? For our purposes, the answer is that a stored case is similar to the query if its solution can easily be adapted to solve the query [Leake 1995; Bergmann et al. 2001]. Retrieving past cases based on their *adaptability* has been called 'adaptation-guided retrieval' [Smyth and Keane 1998]. The difficulty with this approach is that since the query's solution is unknown, how can a case's adaptability be determined without actually trying to adapt its solution and seeing how much effort is required to fit it to the query? This problem is generally resolved by recourse to the similarity assumption: similar problems are assumed to have similar solutions, and similar solutions are assumed to be easily adaptable from one to another. So cases most similar to the query are taken to be those with problem parts most similar to the query. Determining the similarity between two cases' problem parts can be accomplished using a *similarity function* [Althoff and Richter 2001]. One common approach is to take the global similarity between two cases $\boldsymbol{\tau}$ and $\boldsymbol{\rho}$ as the sum of the local similarities between each pair of problem attribute values:

$$Sim(\boldsymbol{\tau}, \boldsymbol{\rho}) = \sum_{a \in A} w_a \ sim(\tau_a, \rho_a) \qquad \textbf{(3.1)}$$

where A is the set of all problem attributes and w_a is the *weight* of attribute a.

The local similarity function $sim(\tau_a, \rho_a)$ (also known as a *comparator*) can be tailored to the requirements of each individual attribute. For commonly occurring nominal and numeric attributes, the following function is often used:

$$sim(\tau_a, \rho_a) = \begin{cases} 1 & , a \text{ nominal and } \tau_a = \rho_a \\ 0 & , a \text{ nominal and } \tau_a \neq \rho_a \\ 1 - \dfrac{|\tau_a - \rho_a|}{a_{\max} - a_{\min}} & , a \text{ continuous} \end{cases} \quad \textbf{(3.2)}$$

The first part of this similarity function is used for nominal attributes in the experimental evaluation in Chapter 7. For continuous (numeric) attributes, an alternative local similarity measure is used in which Gaussian distance weighting is applied. To calculate the similarity between two attribute values, the distance between them is first normalized by converting it to a z-score (i.e., a multiple of the attribute's standard deviation s_a [Runyon and Haber 1991, p. 167]). The similarity score is then the (two-tailed) proportion of the standard normal distribution beyond this z-score:

$$sim(\tau_a, \rho_a) = \text{proportion of standard normal distribution beyond } \frac{|\tau_a - \rho_a|}{s_a}, \quad \textbf{(3.3)}$$

$$a \text{ continuous.}$$

This value is most conveniently read from a lookup table rather than computed from scratch for each similarity calculation.

Note that Equation 3.1 also includes a weight w_a for each problem attribute. An attribute's weight can be set to reflect its *relevance*; those attributes most useful/least useful in arriving at a solution can be assigned the highest and lowest weights respectively (weight values typically lie in the range 0–1). Weights may be set by a human expert with knowledge of the domain, or may be set automatically using an optimization algorithm that minimizes the mean solution error [Wettschereck et al. 1997; Gabel and Stahl 2004]. The setting of global attribute weights is a difficult problem because in complex domains, the relevance of attributes often varies throughout domain space. Optimization algorithms also run the risk of overfitting weight values for each particular dataset, thereby increasing the system's bias and reducing its applicability to new problems [Loughrey and Cunningham 2005]. For this reason, attribute selection is often used instead of attribute weighting to eliminate irrelevant attributes. Indeed, attribute selection can be thought of as a weighting process in which only weight values 0 and 1 are considered. Practical approaches to attribute selection and weighting include filter methods that only consider the statistical characteristics of each dataset (e.g., RELIEF [Kira and Rendell 1992]), and wrapper methods that also take the bias of the problem-solver into account when choosing attributes/weights [Kohavi et al. 1997a].

The second component of successful case retrieval involves designing the system in such a way that the most useful cases can be found in a timely manner. Cases may be organized with some indexing structure that enables rapid retrieval; early systems such as CYRUS [Kolodner 1983] used a structure based on Schank's cognitively inspired dynamic memory model [Schank 1982]. More recent analysis has identified three conditions that should be met by a case retrieval mechanism [Lenz and Burkhard 1996]:

Efficiency: cases should be retrieved in a timely manner, ideally without examining every case in the CB;

Completeness: the same cases should be retrieved as would be found using an exhaustive search;

Flexibility: retrieval should be possible even when the query case is incomplete.

The *case retrieval net* indexing strategy was designed to meet these criteria [Lenz 1999]. It is a highly efficient search algorithm for large CBs, but for smaller CBs, a simple sequential search is often sufficient.

3.2.1.4 Adaptation strategy

Having decided on a suitable case representation and a retrieval strategy, the next step is to decide how retrieved cases should be used to solve new query cases. The process of reusing old cases to solve new problems is called *adaptation*. On the surface, it might appear a minor matter to make small modifications to a retrieved solution to match it to a query. In practice, the adaptation process is often the most difficult part of a CBR system to design, and its complexity generally increases with that of the system's problem solving task. The adaptation process can be viewed as a separate problem-solver embedded within the overall CBR system. As such, it processes a set of inputs (query and retrieved cases) to produce an output (proposed solution for a query).

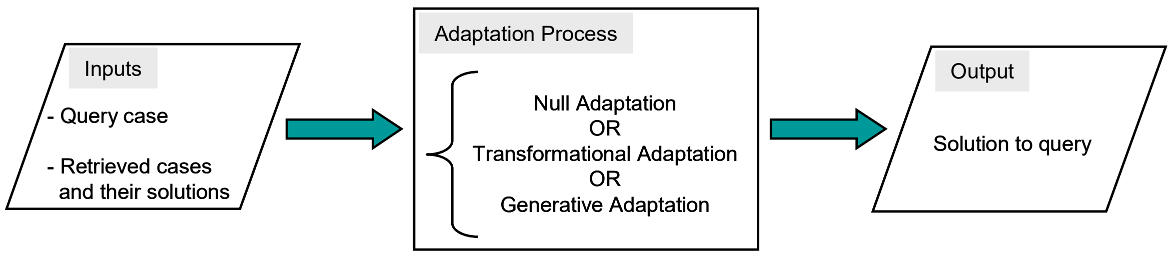


Figure 3.2: The CBR adaptation process

As Figure 3.2 shows, three different approaches can be taken to the adaptation task [Wilke and Bergmann 1998]:

Null Adaptation: Solutions to one or more past cases are reused without modification. This may be because no alterations are necessary; in a classification task, for example, the most commonly occurring solution among a query's neighbours can be proposed as the solution. Alternatively, null adaptation may be used because the process of modifying past solutions is too difficult to attempt. In a medical diagnosis task, for example, the doctor may be presented with past cases and left to evaluate and modify their solutions manually [Schmidt and Gierl 2001]. Null adaptation is the simplest approach to adaptation, and is also the most common among commercial systems [Fuchs and Mille 1999].

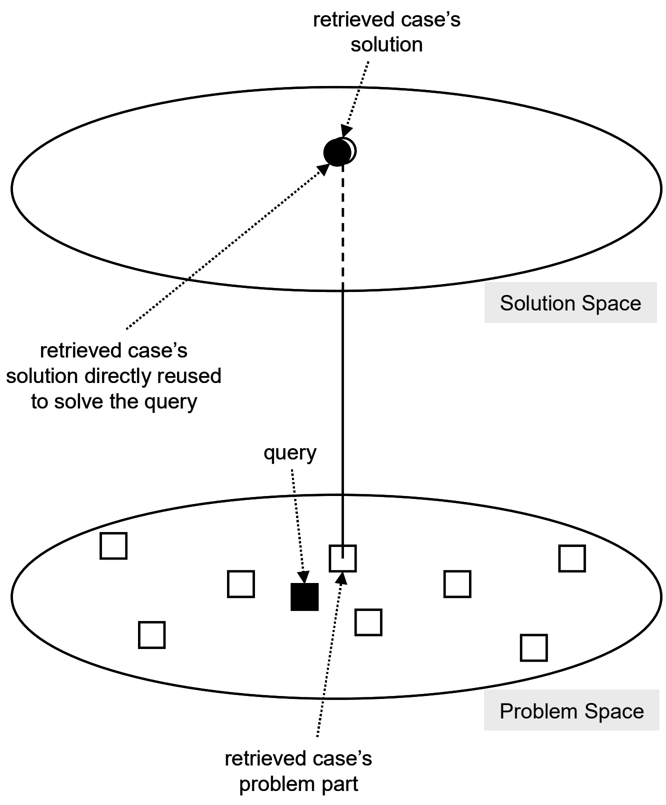

retrieved case's solution
Solution Space
retrieved case's solution directly reused to solve the query
query
Problem Space
retrieved case's problem part

Figure 3.3: Null adaptation

Transformational Adaptation: A second approach to adaptation involves modifying the solution of a past case (or cases) to fit the circumstances of the query more precisely. This modification is normally carried out using a domain-dependent rule-based system [Leake et al. 1995]. The nature of the transformation applied to retrieved solutions will vary with the structure of the solution. In regression tasks, for example, transformational adaptation simply involves altering the value of a real-valued variable. With structured solutions such as plans or artefact designs and configurations, adaptation may involve adding, deleting, or modifying aspects of the retrieved solutions [Smyth and Keane 1996]. The challenge when using transformational adaptation is to acquire the set of rules needed to alter retrieved solutions appropriately. As noted in Section 3.3, CBR is often used in circumstances where a strong domain model is not available. Fortunately, transformational adaptation does not require a complete domain model since it only involves reasoning within the solution space and not within the entire domain space [Bergmann and Wilke 1998]. For this reason, transformational adaptation is generally preferred to generative adaptation (described below).

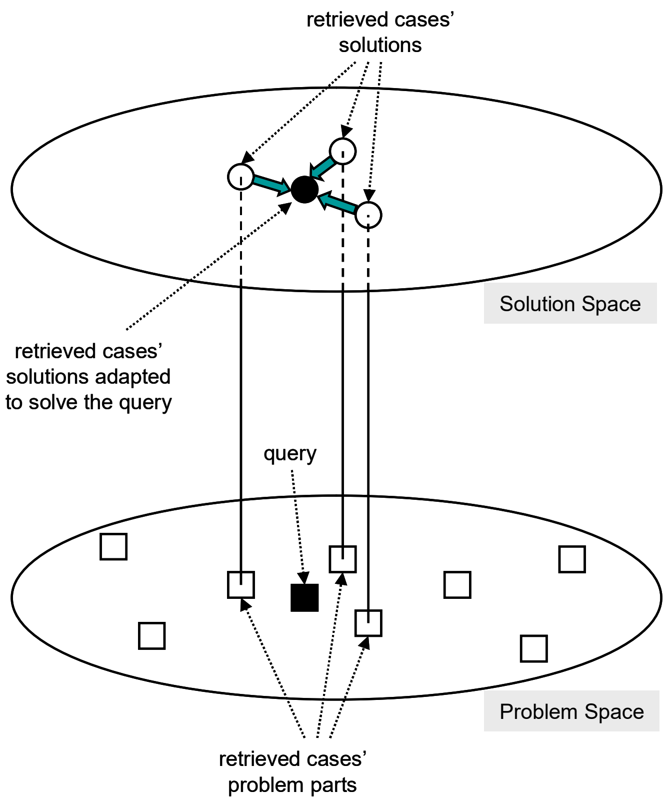

retrieved cases' solutions
Solution Space
retrieved cases' solutions adapted to solve the query
query
Problem Space
retrieved cases' problem parts

Figure 3.4: Transformational adaptation

Generative Adaptation: The final approach to adaptation is very different from those preceding it. Generative adaptation requires the adaptation part of a CBR system to incorporate a full problem-solver capable of solving queries *from scratch*. Given the existence of such a problem-solver, one might ask why CBR is required at all. The answer is that solving queries from first principles may yield poor quality solutions and may be computationally intractable. A trace of the reasoning process used to solve similar problems in the past can provide a skeleton on which to build a new solution. Differences between old and new problems will mean that certain parts of the problem solving process will have to be repeated. Nevertheless, the general framework of past reasoning traces may be directly applicable to the query, and solutions to some specific sub-problems may be immediately reusable. Because it involves applying a similar problem solving process to similar problems, generative adaptation has also been called *derivational analogy* [Carbonell 1983]. It is the least widely used approach to adaptation because from-scratch problem-solvers are most difficult to construct in those domains most amenable to CBR (see Section 3.3).

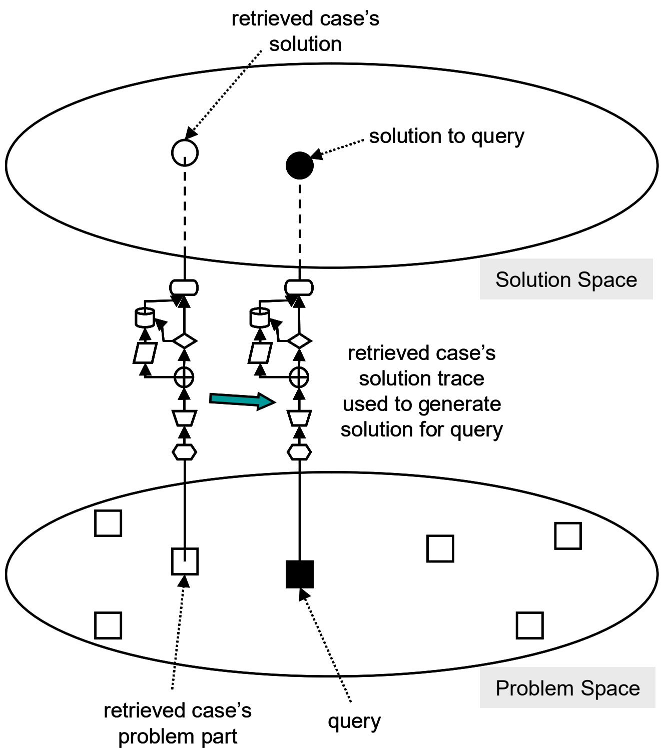

retrieved case's solution
solution to query
Solution Space
retrieved case's solution trace used to generate solution for query
Problem Space
retrieved case's problem part
query

Figure 3.5: Generative adaptation

3.2.1.5 System evaluation

Before a CBR system enters service, it must be evaluated to confirm that it meets its various requirements. These will include 'soft' requirements (e.g., is it easy to use? Is it visually appealing?) as well as 'hard' requirements (e.g., does it perform its task to the specified level of accuracy? Are answers returned within a certain time? Does it behave reasonably when presented with an unrealistic query, or one that it does not have sufficient knowledge to answer?). Based on the outcome of this evaluation, some re-engineering of the system is usually necessary. There may be an insufficient number of initial cases in the CB. Case structure may contain irrelevant attributes, may store data inefficiently, or may not contain enough information for effective problem solving. The retrieval and adaptation mechanisms may not provide an acceptable level of efficiency or accuracy. A return to the knowledge acquisition stage is generally required to gather new data and improve understanding of the domain and system task. Several iterations of the system construction cycle (shown in Figure 3.1) are typically required before a CBR system is ready for live operation.

3.2.2 Operation phase

By the end of the construction phase, a suitable data structure has been found to represent past problems and their solutions, an initial set of cases has been collected, and appropriate retrieval and adaptation mechanisms have been put in place. The system now enters its *operation phase* during which new problems are received and solved. As shown in Figure 3.1, the procedure followed each time a query is received has been formalized as a problem solving cycle with four distinct steps:

Retrieve: The case or cases most similar to the query are retrieved. As discussed in Section 3.2.1.3, these are cases with problem parts similar to the query.

Reuse: The solutions to retrieved cases are reused through a process of adaptation (see Section 3.2.1.4). The output from this step is a proposed solution that is presented to the user.

Revise: The quality of the proposed solution is assessed. This step may be performed automatically or by a human user, and involves subjecting the solution to a set of external criteria (e.g., Does it work? Is it aesthetically pleasing?). The solution may be accepted without modification, accepted with some modifications, or rejected entirely.

Retain: Once a query has been solved it becomes a candidate for being added to the CB. As discussed in Section 3.1.1, CBR systems *learn* by accumulating new cases. The decision about whether to add a case to the CB hinges on whether the case is likely to improve the system's problem solving ability in the future. One approach is to add cases whose solutions were modified or rejected during the Revise step—this has been called *failure-driven learning* [Leake 1996]. Adding such cases may enable the system to correctly solve similar problems in the future. Another approach is to add cases that increase the *coverage* of the CB, that is, that extend the range of possible problems that the system can solve [Smyth and Keane 1995; Smyth and McKenna 2001]. A third approach is simply to add all cases that are solved successfully, on the basis that this increases CB density in those areas of domain space where queries most typically occur and consequently where problem solving needs to be most accurate. Addition and deletion of cases are issues of *CB maintenance* that have grown in importance as more CBR systems enter commercial operation [Leake and Wilson 1998]. Where elements of the retrieval and adaptation processes were optimized using the initial CB, adding new cases may also trigger a re-tuning of these mechanisms [Roth-Berghofer and Iglezakis 2001].

3.3 Knowledge Contained Within a CBR System

Section 3.1.1 described CBR as a knowledge-based approach because it relies on domain-specific knowledge to solve new problems. It is useful to look at exactly where this domain knowledge is stored. [Richter 1995, 1998] proposed dividing a typical CBR system's knowledge into four *knowledge containers*, each closely linked to one of the steps in CBR system construction (see Figure 3.1):

Vocabulary: This is equivalent to *case representation*. As discussed in Section 3.2.1.2, past problems must be represented in a data structure that includes all relevant information while omitting irrelevant and unreliable information. A great deal of effort and expertise may be required to decide on a suitable case structure and to select the optimal set of problem attributes. This expertise is embodied in the vocabulary knowledge container.

Similarity knowledge: This container stores the knowledge necessary for retrieving past cases similar to the query. As discussed in Section 3.2.1.3, this includes the ability of the system to determine the degree of similarity between two cases, as well as knowledge about how to retrieve these cases in an efficient manner.

Adaptation knowledge: The knowledge used to adapt past cases to solve new queries is stored in this container. Section 3.2.1.4 pointed out that adaptation is often carried out using an embedded rule-based system. This embedded system will have its own knowledge containers—a traditional rule-based system has three containers, for example: facts, rules, and an inference engine. The complexity of the problem domain (and the ambition of the adaptation process) will determine the level of knowledge in the adaptation container—a great deal of external data may be needed for effective problem solving in weak theory domains [Aamodt 1994].

Case base: The set of prior cases stored in the CB completes the knowledge of a CBR system. Each case contains one successful problem solving experience from the past. The addition of new cases is the principal means by which CBR systems learn.

Note that the first three containers are usually filled during construction of the system, while the fourth is partially filled prior to deployment and updated during operation. As discussed previously, one of the key advantages of CBR is that knowledge can easily be added by users in the form of new cases.

The 'knowledge container' view of CBR offers the following useful, complementary insights:

- CBR knowledge containers are largely independent of one another in that the knowledge in one can be altered with little effect on the others. For example, cases can be added or removed from the CB or the similarity metric tweaked at any time. This decoupling of components eases maintenance of the overall system.

- The knowledge in each container is not static. Knowledge can be moved between containers, and indeed, any one container can store almost all the knowledge of the entire system. For example, if there is a stored case for all possible scenarios that might arise in the future, then retrieval is sufficient to solve new problems and no adaptation is necessary. Alternatively, if the adaptation process is capable of adapting the solution of *any* past case to solve a query, then the CB need only contain a single case and the retrieval strategy is trivial.

A useful distinction can also been drawn between *knowledge-intensive* and *knowledge-light* systems [Wilke et al. 1997]. This distinction largely rests on the distribution of knowledge between the CB and adaptation containers. Where most domain-specific knowledge lies in the CB, systems are referred to as knowledge-light. Complex problem domains may necessitate extensive rule-based adaptation processes. Systems in which the adaptation container contains large amounts of explicitly coded domain knowledge are referred to as knowledge-intensive. It is important to note that both are *knowledge-based systems* as defined in Section 3.1.1; the distinction lies in the form that the domain knowledge takes. Most commercial CBR systems in operation today are knowledge-light. Two reasons can be given for this. First, it can be argued that storing domain knowledge as a set of cases is most natural for CBR systems, and indeed constitutes the primary advantage of CBR as a technology. Second, complex domain-specific rule-sets are difficult to construct and maintain from a practical point of view, and are therefore not suitable (or necessary) for most ordinary CBR applications.

3.4 Case Study: CBR-CD

This chapter opened by stating that the goal of this research is to construct an effective regression system using CBR. Details of a new CBR-based system (CBR-CD) are contained in Chapter 5, but it is useful at this stage to examine its outline within the general CBR framework developed above:

System task and domain: CBR-CD has the task of predicting the value of a numeric variable (i.e., regression). It is immediately applicable to any domain where case attributes have numeric and nominal types, and can be extended to work in domains with more complex attribute types.

System construction: Following the steps described in Section 3.2.1:

Initial cases: The experimental evaluation described in Chapter 7 is performed over a number of standard datasets that provide the initial cases for the CB.

Case representation: Each case is represented as a vector of attribute values.

Similarity metric and retrieval strategy: Similarity between cases is calculated using Equations 3.1, 3.2 and 3.3. In Equation 3.1, the attribute weighting parameter w_a is not used (i.e., it is set to 1 for all attributes). Additional similarity functions can be added for attribute types other that numeric and nominal. Retrieval is performed using a simple sequential search—this is adequate for the experimental domains used, and is guaranteed to find those cases most similar to any query.

Adaptation strategy: Adaptation is the primary focus of this thesis. Figure 3.2 shows that the adaptation process can be modelled as a separate problem-solver in its own right. CBR-CD uses a second, embedded CBR system as its adaptation problem-solver whose case representation and similarity metrics are borrowed from the main system and whose CB is constructed from the differences between cases. Transformational adaptation in the main system is then performed using a null adaptation (i.e., retrieval-only) strategy within this embedded system.

System operation: The system only operates within an experimental environment, and so does not undergo the full range of system operation tasks:

Retrieve: Retrieval is performed as described above. The appropriate number of cases to retrieve varies with the dataset used, and is set using cross validation.

Reuse: Adaptation is performed using the embedded CBR system described above. Within this, a number of adaptation strategies are possible; the best strategy for any particular dataset is again found using cross validation.

Revise: Numeric predictions are presented to the user as output from the system. The experimental evaluation proceeds by comparing actual with predicted values to measure the accuracy of the system on test datasets. During live operation, the 'revise' step would involve an assessment of the probable accuracy of each prediction. To assist the user with this task, the system also outputs case data from the adaptation process to show how the predicted solution was arrived at.

Retain: This step is not performed by the system; no additional cases are added to the CB beyond the initial test cases supplied.

System knowledge: Knowledge in CBR-CD is distributed between Richter's four knowledge containers as follows:

Vocabulary: The domain knowledge needed to decide on the most appropriate case attributes and attribute types is embedded within the experimental datasets used.

Similarity knowledge: Similarity knowledge for numeric and nominal attributes is built into system itself. If other attribute types were used, custom similarity metrics would be needed to accommodate them.

Adaptation knowledge: The knowledge required for successful adaptation is automatically generated from the vocabulary, similarity and CB containers.

Case base: All case knowledge is supplied by the experimental test-sets used.

Knowledge-intensive vs. knowledge-light: As indicated in the title to this thesis, a knowledge-light approach is taken to the regression task. This is because the adaptation process does not require additional domain knowledge in the form of rule-sets or other explicit structures. Section 3.3 discussed the idea of moving system knowledge between the four containers. CBR-CD provides an example of this: the adaptation container is automatically filled with knowledge from the other three. This makes it easier to construct and maintain regression systems for new problem domains.

3.5 Summary

This chapter introduced case-based reasoning (CBR), a form of problem solving in which new problems are addressed by re-using the solutions to similar past problems. CBR was initially designed to model problem solving in humans, but has grown from its cognitive science roots to become one of the major paradigms of modern machine learning. Its chief advantage is its simplicity—CBR systems are generally easier to construct, maintain, and use than alternative knowledge-based systems. CBR is especially useful in weak-theory domains that cannot be accurately modelled by static rule-sets; for example, recommending products to users on the Internet.

Construction of new CBR systems is an iterative process that begins with a study of the application domain and the collection of relevant historical data. From this knowledge, the system designer fashions a case structure to hold relevant details of past problems and their solutions, a retrieval mechanism capable of recalling relevant past cases to memory, and an adaptation mechanism for modifying the solutions of past cases to suit the particular circumstances of each new case. A CBR system can enter operation with a small initial set of past cases, and then incrementally improve its performance (i.e., *learn*) by adding new cases over time. The different parts of a CBR system (i.e., case representation, set of past cases, similarity and adaptation mechanisms) have been described as *knowledge containers* that embody the system's knowledge store; like other knowledge-based approaches, CBR's usefulness depends on the effective utilization of relevant, high quality, domain-specific knowledge. As a case study, the chapter concluded with an examination of the CBR-CD regression system that is described in detail in Chapter 5.

Chapter 4

Using CBR for Regression

CBR is among the most widely used problem solving approaches in modern AI, and regression is among the most important and fundamental tasks in machine learning. This chapter combines the two by looking at how CBR can be used for regression. Several different approaches are possible, but all utilize the basic CBR methodology described in Chapter 3. That is, a query case is always solved by retrieving its nearest neighbours (NNs) and re-using their solutions. The various approaches differ from one another in their adaptation processes—they re-use the solutions from past cases in different ways.

Four different approaches to performing regression using CBR will be examined; they are summarized in Table 4.1 below. The first three, CBR-B, CBR-D and CBR-AR, are described in the remainder of this chapter. CBR-CD is the main contribution of this thesis and is described in detail in Chapter 5.

Table 4.1: CBR-based regression algorithms

Regression Algorithm	*Abbrev.*	*Summary of problem solving methodology*
CBR-Basic	CBR-B	Take the weighted average of solutions for *k*-NNs
CBR-Diverse	CBR-D	Take the weighted average of solutions for *k* diverse neighbours
CBR-AdaptationRules	CBR-AR	Bridge the gap between the query and an NN using adaptation rules derived from the CB
CBR-CaseDifferences	CBR-CD	Bridge the gap between the query and its NN using difference-cases generated from the CB

The following sections will refer to a simple example domain where the task is to predict the value of a house based on its number of bedrooms and location:

$housePrice = f(numBedrooms, location)$.

Problem attributes *numBedrooms* and *location* have range 1–6, with location 1 generally considered least desirable. Each stored case has structure

(*numBedrooms*, *location*, *housePrice*)

and the existence of a CB containing multiple (e.g., 20) such cases is assumed.

4.1 CBR-B Algorithm

The simplest approach to performing regression using CBR is to retrieve a query's nearest neighbours and take the weighted average of their solutions.

4.1.1 Problem solving methodology

CBR-B is identical to *k*-NN except that any attribute type is allowed (recall from Section 2.2 that *k*-NN is restricted to numeric attributes). The distance between two cases **C** and **Q,** $d(\mathbf{C},\mathbf{Q})$, cannot be calculated with the Euclidean distance function used by *k*-NN (Equation 2.2). Instead, it is based on the CBR similarity function given by Equation 3.1:

$$d(\mathbf{C},\mathbf{Q}) \;=\; Sim_{\max} - Sim(\mathbf{C},\mathbf{Q}) \;=\; |A| - \sum_{a \in A} sim(Q_a, C_a) \qquad \textbf{(4.1)}$$

where A is the set of all problem attributes, and local similarity function $sim(Q_a, C_a)$ has range 0–1. The nearest neighbours to a query are those closest to it (i.e., those that minimize this distance function).

For cases containing only nominal and numeric attributes, $sim(Q_a, C_a)$ is given by Equations 3.2 and 3.3 (see Section 3.2.1.3). The contribution from each neighbour to the predicted target value is weighted by distance from the query using a Gaussian weighting function:

$$w_{\mathbf{C}} = e^{-d(\mathbf{Q},\mathbf{C})^2} \qquad \textbf{(4.2)}$$

Note that from a maximum value of 1, the rate at which weights decline is related to the number of problem attributes. An alternative approach would be to divide the distance function $d(\mathbf{C}, \mathbf{Q})$ by a smoothing parameter λ that is optimized for each individual problem domain [Atkeson and Schaal 1996]. This is omitted to preserve the generality of the algorithm described here, but it is useful to be aware that performance in any particular domain can be improved by adding a smoothing parameter and tuning the weighting function w_C. (Another view of the approach taken here is that the same *fixed bandwidth* with $\lambda = 1$ is used for all datasets. See Section 2.3 for more about smoothing parameter λ.)

The final step in predicting the query's target value is to take the weighted average of its neighbours' solutions:

$$\hat{y} = \frac{\sum_{\mathbf{C} \in NNs} w_{\mathbf{C}}\, y_{\mathbf{C}}}{\sum_{\mathbf{C} \in NNs} w_{\mathbf{C}}} \tag{4.3}$$

The accuracy of predictions made using CBR-B will depend on the value chosen for *k* (i.e., the number of neighbours whose solutions are averaged). Larger values of *k* can offset the ill-effects of noisy cases in real-world datasets. On the other hand, smaller values mean that each prediction only uses cases most similar to the query with solutions likely to be most relevant. The optimal choice of *k* varies with the problem domain, and is generally found using cross-validation during an initial training phase.

The problem solving methodology for CBR-B is summarized in Figure 4.1.

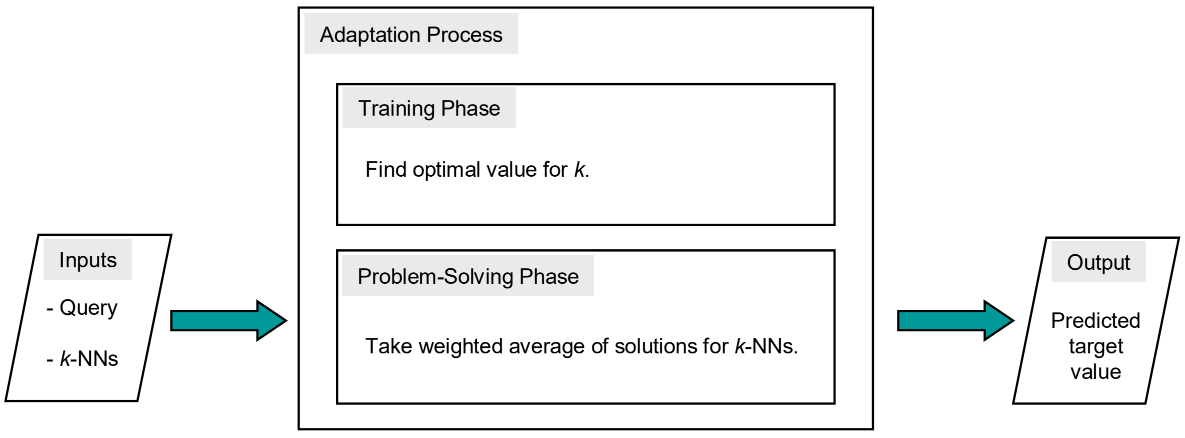


Figure 4.1: Adaptation process for CBR-B

4.1.2 Advantages and limitations

CBR-B shares many of the advantages and disadvantages of the *k*-NN algorithm. Its advantages can be summed up as follows:

No assumptions about target function: CBR-B is *nonparametric* in that it makes no assumptions about the underlying target function. It is therefore suitable for use in domains where the target function is complex.

Simplicity: CBR-B is simple to use and understand. It is also easy to apply in new problem domains—if a set of cases is available with numeric and nominal attributes, the only step required to construct a working CBR-B system is to choose a suitable value for *k*.

The first three disadvantages of CBR-B are shared by all CBR systems: CBR-B is computationally expensive, sensitive to irrelevant problem attributes, and relies on adequate case coverage. These aspects of lazy, instance-based problem solving have already been examined. The first, high processing requirements, is largely redundant thanks to today's availability of cheap computing power. The second, sensitivity to irrelevant attributes, can be avoided by careful selection of attributes by human experts or by applying a range of automatic, widely available attribute selection algorithms. Finally, the need for adequate case knowledge is simply a restatement of the principle underlying all knowledge-based systems: problem solving in a particular domain relies on the presence of domain-specific knowledge. Only in the simplest domains can an accurate model be inductively generated using minimal domain-specific knowledge.

The remaining two disadvantages apply specifically to CBR-B. It is these disadvantages that alternative CBR-based approaches (i.e., CBR-D, CBR-AR and CBR-CD) seek to alleviate in their different ways.

Relative positions of the query and its neighbours in domain space are not considered: Each predicted target value will always be bounded by the highest and lowest solution values among the query's NNs. The relative positions of the query and its NNs in domain space are ignored, and so the prediction may not always be accurate. For example, suppose a query in the housing domain has *numBedrooms*=5 and its two NNs have *numBedrooms*=4 and 3. Using CBR-B, the predicted *housePrice* will be between those of the NNs when it should in fact be higher. This is illustrated in Figure 4.2. Ideally, we would like to take the relative positions of the query and its NNs into account when making predictions. In domains where all problem attributes are numeric, this can be achieved by constructing a local model (e.g., using locally-weighted linear regression). Unfortunately, this approach isn't possible where important aspects of the problem domain are represented by nominal or more structured attribute types.

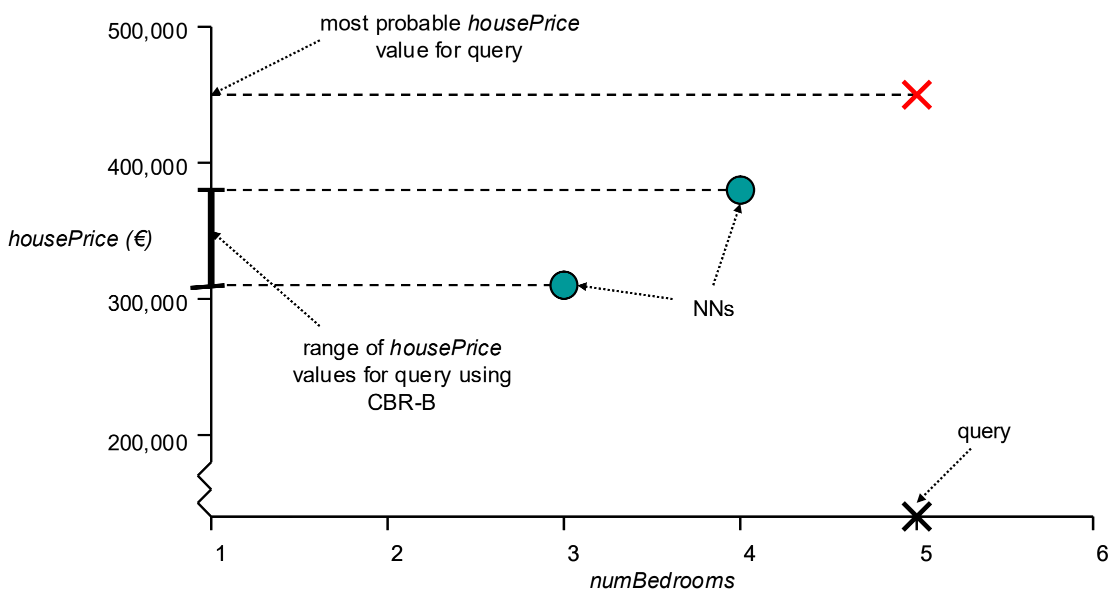

most probable *housePrice* value for query
500,000
400,000
300,000
200,000
housePrice (€)
range of *housePrice* values for query using CBR-B
NNs
query
1
2
3
4
5
6
numBedrooms

Figure 4.2: Inability of CBR-B to take case positions into account

Only the query's neighbours influence each prediction: Only those cases most similar to the query have some input into the predicted target value. However, a great deal of implicit domain knowledge is contained in the remainder of the CB that is potentially useful for adaptation. Ideally, we would like to extract and apply this knowledge to supplement local problem solving.

4.2 CBR-D Algorithm

CBR-D is a new algorithm that is a variation of CBR-B. It seeks to address the problem that the relative positions of the query and its neighbours in domain space are not taken into account. It differs from CBR-B in that each prediction is based not on a query's most immediate neighbours but on a diverse set of neighbouring cases. Taking the average of solutions from a set of NNs with the query at its centre should yield a better prediction than using a set of NNs uniformly offset from the query in one particular direction. In Figure 4.2, for example, a prediction based on NNs with *numBedrooms*=4 and 6 would likely be more accurate than one using the NNs shown.

The idea of introducing diversity to a set of nearest neighbours has been considered in previous research. [Jain et al. 2003] looked at adding diversity to a simple database application. When presented with a query, a set of instances was returned with the property that instances were similar to the query but different from one another. A restaurant domain was used to demonstrate the value of this: recommending different types of restaurants in response to a query made the system more useful for the user. In a related area, [McGinty and Smyth 2003] examined the role of diversity in CBR-based conversational recommender systems.

Nearest neighbour diversity for regression tasks has also been explored by [Zhang et al. 1997] and [Hüllermeier 2005]. Zhang et al. introduced a variation of the *k*-NN algorithm called *k*-SN (*k* surrounding neighbour). The idea behind *k*-SN is the same as that of Jain et al.: cases used for a prediction should be similar to the query but different from each other. The retrieval algorithm iteratively chooses pairs of cases to add to the set of NNs such that

- The first case is close to the query, and
- The second is further from the first than from the query, and closer to the query than any case not already in the set.

That is, cases are added in diverse pairs with one offsetting the other. The diversity between each pair of cases and those pairs added before and after it is not considered.

Hüllermeier took a different approach, and assessed the quality of a set of cases by using a Choquet integral to aggregate the contributions of the individual cases. A non-additive measure is used so that high similarity between member cases reduces the overall value of the set. The algorithm (referred to as Cho-k-NN) is therefore capable of assessing the overall diversity of a set of cases. Both *k*-SN and Cho-k-NN share the underlying principle that diversity is a measure of the distance (i.e., pairwise dissimilarity) between entire cases. CBR-D takes a different approach and measures diversity at the level of individual attributes.

4.2.1 Problem solving methodology

The problem solving process adopted by CBR-D is shown in Figure 4.3. It differs from that of CBR-B only in the selection of neighbouring cases: the adaptation process receives 3*k* NNs as input from the retrieval process and selects a diverse set of *k* cases from among them. (Note that this could also be modelled as a modification of the retrieval algorithm.)

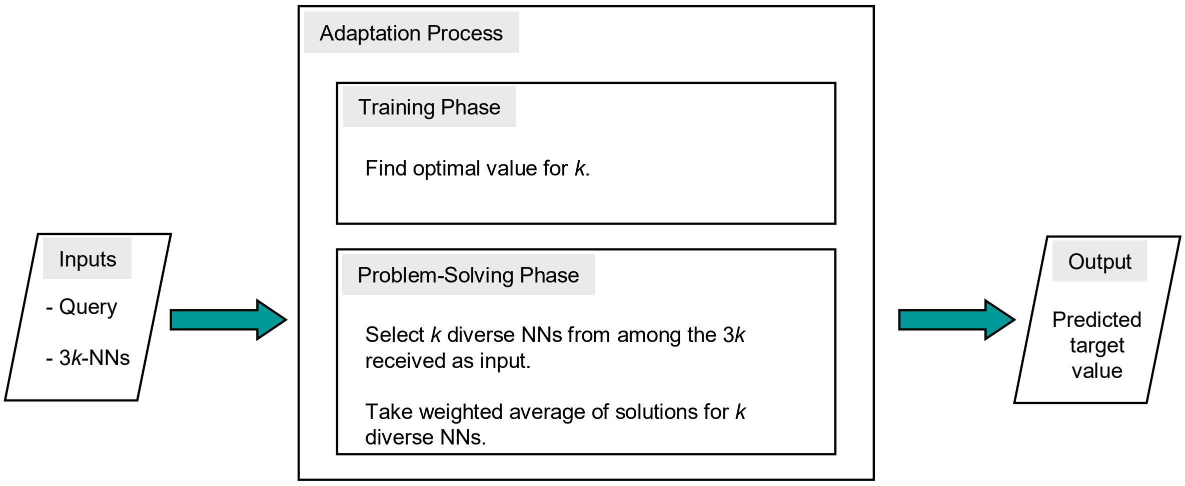

Adaptation Process
Training Phase
Find optimal value for *k*.
Inputs
- Query
- 3*k*-NNs
Problem-Solving Phase
Select *k* diverse NNs from among the 3*k* received as input.
Take weighted average of solutions for *k* diverse NNs.
Output
Predicted target value

Figure 4.3: Adaptation process for CBR-D

Selecting *k* diverse neighbours from a larger set is performed using a greedy search algorithm:

1. Begin by seeding the set of diverse neighbours with the case most similar to the query.

2. Add additional neighbours one by one so that the diversity of the overall set is maximized at each stage. The diversity of the overall set is calculated as the *sum of diversities for each individual problem attribute*. Each attribute type has its own diversity function that accepts the following input: the query, a set of existing attribute values, and a new potential value. A diversity score in the range 0–1 is returned, with scores closer to 1 indicating that the new attribute value makes a positive contribution to the set. CBR-D has inbuilt support for nominal and numeric attribute types, and their diversity functions operate as follows:

   Nominal attribute: NNs with the same attribute value as the query are preferred. Otherwise, NNs are chosen to maximize *entropy* among the attribute's values—diversity is calculated as the values' entropy divided by the maximum possible entropy for the set. (Entropy is a concept in information theory first introduced in [Shannon 1948]. Basically, entropy is maximized by having the highest number of different nominal values and roughly similar numbers of each.)

   Numeric attribute: NNs are chosen so that the mean value for the attribute among the set of diverse NNs is close to that of the query; the aim is to balance values about the query's value.

   Other attribute types can be supported by the definition of custom diversity functions.

An NN may score well on some attributes and badly on others. If attributes are assigned weights that reflect their relevance, these weights can be multiplied by diversity scores so that diversity is prioritized for more important attributes (attribute weights are not used for CBR-D in the experimental evaluation in Chapter 7).

CBR-D differs from *k*-SN and Cho-k-NN in that diversity is defined on the level of individual attributes rather then entire cases. The motivation for this is to avoid the loss of information that comes from measuring diversity among cases by the simple pairwise distances between them. This problem can be demonstrated with a simple example (depicted in Figure 4.4). Suppose the following query is received in the housing domain:

$\mathbf{Q}$ = (4, 3, ?),   i.e., *numBedrooms*=4 and *location*=3.

Retrieving a sample nearest neighbour and calculating the distance between the two (using the usual CBR approach of summing individual attribute distances):

NN = (3, 3, €250,000), $d(\mathbf{Q}, \mathbf{NN}) = 0.2$ (from Equation 4.1).

Now suppose the following two cases are also stored in the CB:

$\mathbf{C_1}$ = (5, 3, €350,000), $d(\mathbf{Q}, \mathbf{C_1}) = 0.2$, $d(\mathbf{NN}, \mathbf{C_1}) = 0.4$;

$\mathbf{C_2}$ = (4, 2, €260,000), $d(\mathbf{Q}, \mathbf{C_2}) = 0.2$, $d(\mathbf{NN}, \mathbf{C_2}) = 0.4$.

Cases $\mathbf{C_1}$ and $\mathbf{C_2}$ are equidistant from **Q**, and also equidistant from **NN**. (Note that in this simple example, **NN**, $\mathbf{C_1}$ and $\mathbf{C_2}$ are all equidistant from **Q** and so the choice of **NN** as the nearest neighbour is somewhat arbitrary. This doesn't affect the conclusion drawn.) If a set of diverse neighbours is assembled starting with **NN**, then $\mathbf{C_1}$ or $\mathbf{C_2}$ would contribute equally to the diversity of the set when considered as entire cases. Yet looking at the attribute values of $\mathbf{C_1}$ and $\mathbf{C_2}$, we can see that $\mathbf{C_1}$ would actually be more useful—it has the same value for *location* as **Q** and **NN**, and its value for *numBedrooms* neatly counterbalances that of **NN**. That is, the set of diverse cases {**NN**, $\mathbf{C_1}$} has **Q** at its centre whereas {**NN**, $\mathbf{C_2}$} does not. This example shows that considering diversity at attribute rather than case level can lead to a choice of cases more likely to give an accurate prediction.

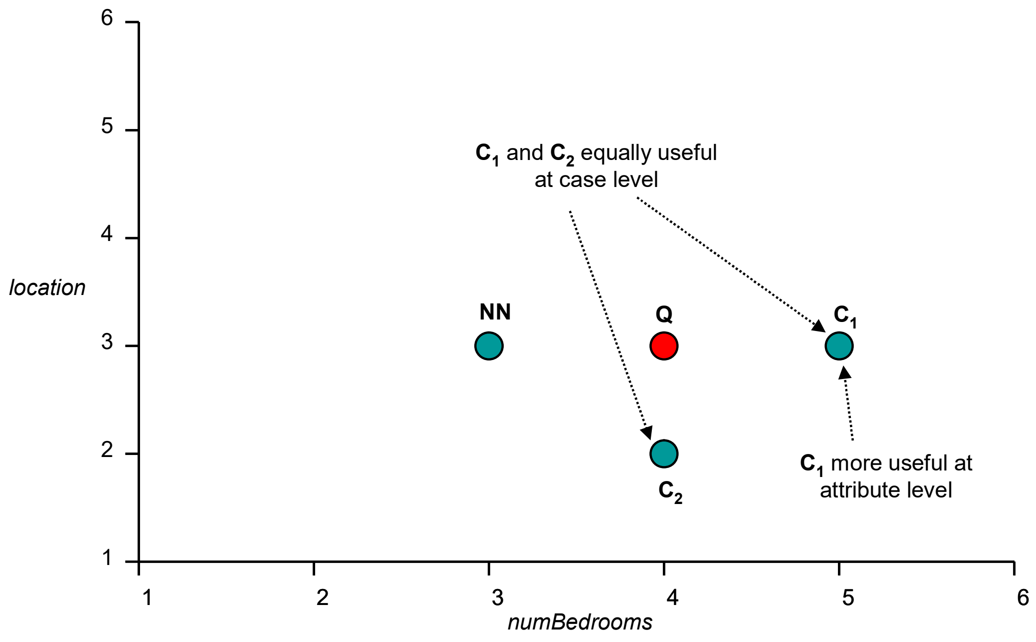

C$_1$ and C$_2$ equally useful at case level
NN
Q
C$_1$
C$_2$
C$_1$ more useful at attribute level
location
numBedrooms
1
2
3
4
5
6

Figure 4.4: Advantage of considering diversity at attribute level

Once a set of k diverse neighbours has been chosen from the initial $3k$, problem solving proceeds as for CBR-B. The predicted target value for the query is calculated as the distance-weighted average of solutions for the set of diverse neighbours using Equation 4.3. Note that CBR-D is another demonstration of adaptation-guided retrieval: a set of diverse cases is chosen with the aim of maximizing their combined usefulness during adaptation rather than their individual similarities to the query.

4.2.2 Advantages and limitations

CBR-D inherits the advantages of CBR-B in that it is quite straightforward to use and does not make any assumptions about the form of the underlying target function. In addition, it has the following advantage of its own:

Relative positions of the query and its neighbours in domain space are considered: Diversity is preferred among the NNs used for each prediction. The goal is to have the query at the centre of its NNs in problem space, so that the weighted average of their solutions is likely to give a more accurate prediction. Diversity is considered at the level of individual attributes, avoiding the loss of information that results from considering only the pairwise distance between entire cases.

As might be expected, CBR-D also comes with its own limitations:

More complex and computationally expensive than CBR-B: Calculating diversity for a set of NNs requires a diversity function for each attribute type. Predicting a target value also involves extra computation because the diversity among NNs must be assessed repeatedly. Neither problem is especially serious, however; defining diversity is quite simple for common attribute types and the additional computation has proved to have a negligible impact on performance in practice.

Performance may be poor in non-linear domains: Choosing neighbours on the basis of diversity means choosing cases more distant from the query than its most immediate neighbours. In highly non-linear domains or in scenarios where case coverage is sparse, the benefits of a diverse set of NNs may be more than offset by the fact that more distant cases are a lot less useful. This problem is lessened by the fact that the contribution from each NN is distance-weighted in Equation 4.2.

4.3 CBR-AR Algorithm

The third CBR-based regression algorithm, CBR-AR, takes a different approach to adaptation than those described above. It does not predict a query's target value by taking the weighted average of solutions from a (diverse) set of neighbouring cases. Instead, it generates a set of adaptation rules during an initial training phase, and when a query is received, assembles a sequence of rules to bridge the gap between it and a nearest neighbour.

The CBR-AR algorithm was originally proposed in [Hanney and Keane 1996, 1997]. Embedded rule-based systems constitute the standard approach to adaptation, and CBR-AR offers the possibility of generating an adaptation rule-set automatically from differences between cases in the CB. In doing so it addresses both of the limitations of CBR-B listed in Section 4.1. When solving a query case, it takes the relative positions of the query and its nearest neighbour into account by constructing a custom path from one to the other. If a different neighbour equally distant from the query was used, a different path between the two cases would result. CBR-AR also supplements local knowledge from close to the query (i.e., the query's NN) with global adaptation knowledge mined from all parts of domain space (i.e., adaptation rules). Tapping into adaptation knowledge implicitly stored in the CB increases the amount of relevant domain knowledge that is brought to bear during problem solving, and as in any knowledge-based system, this is likely to improve performance.

Section 4.3.1 describes how an adaptation rule-set can be generated and used for a regression task in the example housing domain (this domain was also used in the original research).

4.3.1 Problem solving methodology

The problem solving methodology adopted by CBR-AR has two parts: generation of adaptation rules during an initial training phase, and use of these rules during operation. This section also examines ways in which adaptation rules can be altered to extend the range of possible problems they can be applied to.

4.3.1.1 Generating a set of adaptation rules

CBR-AR automatically generates adaptation rules by comparing each case in the CB with all other cases. Suppose the first case in the CB is called **Case_1**. Since we know the solution to this case, comparison with other cases can be used to construct a set of rules as follows:

```
IF (problem changes from Case_2_problem → Case_1_problem) THEN
(solution changes from Case_2_solution → Case_1_solution)

IF (problem changes from Case_3_problem → Case_1_problem) THEN
(solution changes from Case_3_solution → Case_1_solution)

…
```

For example,

Rule 1

```
IF (numBedrooms changes from 1 → 2 and location from 3 → 4) THEN
(housePrice changes from €200,000 → €400,000)
```

Rule 2

```
IF (numBedrooms changes from 3 → 4 and location from 4 → 4) THEN
      (housePrice changes from €550,000 → €680,000)
```

Each rule contains the difference between two cases and the consequent change in solution. Note that both problem attributes change value in *Rule 1*, while only one changes in *Rule 2*. In general, changing attribute values constitute the *differences* covered by the rule and unchanged values provide the rule's *context*.

Generating a full set of rules for a CB of size *n* would result in *n* x (*n–1*) rules. To reduce storage requirements, each case may be compared with only a limited number of other cases. For example, comparing each case with its 10 NNs produces a rule-set with size *n* x *10*.

4.3.1.2 Using adaptation rules to make a prediction

Most rule-based systems take the following approach to problem solving: starting from an initial state, search for one or more rules that create a path to the specified goal state [Thagard 2005]. This is precisely how adaptation rules are applied to solve new problems: the query constitutes the goal state, its NN the initial state, and adaptation rules create a path between them. When presented with a new query, CBR-AR predicts its target value with the following sequence of actions:

1. Retrieve the query's NN.
2. Calculate the differences between the two cases' problem parts.
3. Search for one or more rules matching the set of differences—these rules predict the change in solution resulting from the problem differences.
4. Make a prediction by adding together the solution from the NN and the solution changes in the matching rules.

This process can be explained more easily with an example. Suppose the following query is received: **Q** = (*numBedrooms*=5, *location*=1). Then *housePrice* is predicted as follows:

1. Retrieve the query's **NN**: (*numBedrooms*=3, *location*=2, *housePrice*=€350,000)
2. Calculate the differences between **NN** and **Q**:
   Δ(**NN**, **Q**) = (*numBedrooms*: 3 → 5, *location*: 2 → 1)
3. Find rules to account for these differences:

```
IF (numBedrooms changes from 3 → 5) THEN
      (housePrice changes from €300,000 → €450,000)

IF (location changes from 2 → 1) THEN
      (housePrice changes from €300,000 → €270,000)
```

4. Predict *housePrice* for **Q**: €350,000 + €150,000 – €30,000 = €470,000

Here, two rules bridge the gap between the query and its NN. They indicate what effect the differences in *numBedrooms* and *location* have on *housePrice*. Note that the context of each rule (i.e., the values of unchanging attributes) is not shown—the reason that context is sometimes omitted is explained below.

The problem solving process followed by CBR-AR is summarized in Figure 4.5.

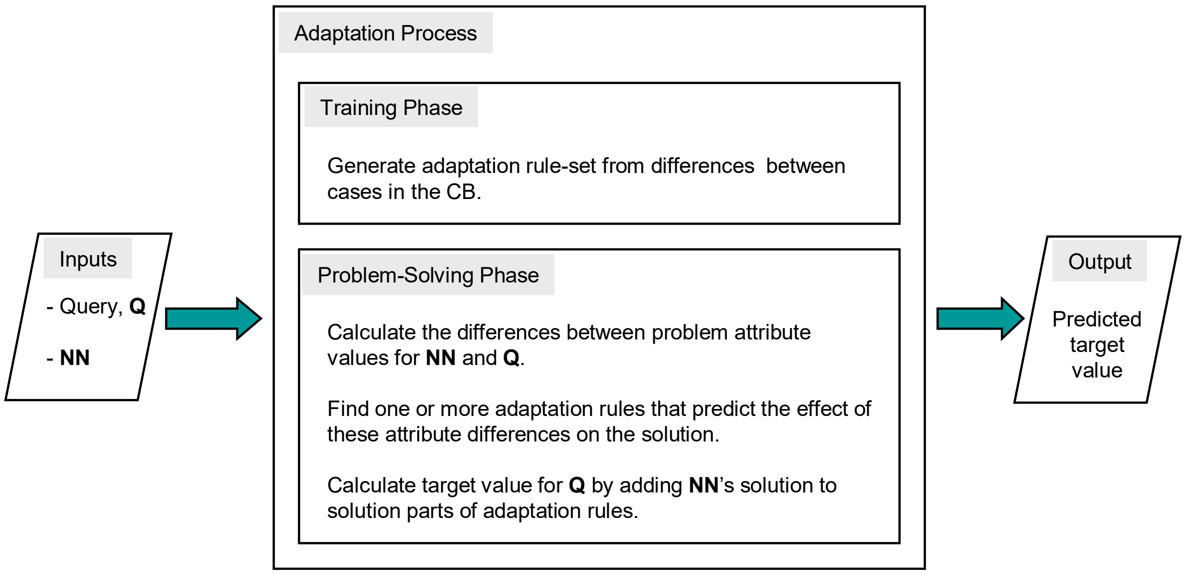

Adaptation Process
Training Phase
Generate adaptation rule-set from differences between cases in the CB.
Problem-Solving Phase
Calculate the differences between problem attribute values for NN and Q.
Find one or more adaptation rules that predict the effect of these attribute differences on the solution.
Calculate target value for Q by adding NN's solution to solution parts of adaptation rules.
Inputs
- Query, Q
- NN
Output
Predicted target value

Figure 4.5: Adaptation process for CBR-AR

4.3.1.3 Extending the applicability of adaptation rules

Adaptation rules specify the impact that particular differences between cases have on the solution, thereby codifying the adaptation knowledge implicitly contained in the CB. This knowledge is highly specific, however, depending as it does on the precise set of cases available.

The limited scope of the knowledge contained in adaptation rules can be seen by considering a domain with 5 numeric problem attributes, each with 100 possible values. This domain has 100^5 (= 10^{10}) possible combinations of attribute values, so that a relatively well-stocked CB of 1000 cases will still contain only 0.000001 of all possible cases. Problem solving remains possible in domains such as this because cases tend to be clustered in those areas of domain space where query cases typically occur. Even allowing for this, however, it is highly unlikely that a rule-base will contain rules to cover the differences between a particular query case and its NN while also respecting the query's context.

The technique of using adaptation rules to solve new cases needs a *problem solving bias* that allows it to broaden the range of problems it can tackle. CBR-AR introduces this bias in two ways. The first extends the set of attribute differences handled by a rule, while the second extends the context in which a rule may be applied. Both increase the applicability of individual rules to give them greater problem solving potential.

Strategy 1: Combine adaptation rules to create generalised rules that cover a range of values for a problem attribute.

For example, combine rules

```
IF (numBedrooms changes from 2 → 3 and location from 4 → 4) THEN
(housePrice changes from €400,000 → €500,000)

IF (numBedrooms changes from 5 → 6 and location from 4 → 4) THEN
(housePrice changes from €800,000 → €900,000)
```

to create a generalised rule:

```
IF (numBedrooms changes by 1 in range 2–6 and location from 4 → 4)
THEN
(housePrice increases by €100,000)
```

This strategy is based on Michalski's closing interval rule [Michalski 1983], which states that if two rules differ only in the value of one *linear* descriptor, they can be replaced by a single rule covering the interval between the values.

Strategy 2: Relax the context in which adaptation rules may be applied.

For example, replace rule

```
IF (numBedrooms changes from 1 → 2 and location from 4 → 4) THEN
(housePrice changes from €250,000 → €380,000)
```

with a context-free rule:

```
IF (numBedrooms changes from 1 → 2) THEN
(housePrice changes from €250,000 → €380,000)
```

In the example in Section 4.3.1.2 above, Strategy 2 was applied to create rules that depend only on attribute differences and not on context.

4.3.2 Advantages and limitations

The CBR-AR algorithm is now complete. We have seen how adaptation rules can be created from the differences between stored cases and used to solve regression problems by forming a path from an NN to the query. Adaptation rules based on raw differences between cases have been shown to have insufficient problem solving bias to be really useful, and two strategies designed to address this issue have been examined.

All in all, the idea of mining adaptation knowledge from the CB is a good one and CBR-AR has several significant advantages over alternative approaches:

Knowledge-light approach to adaptation: As previously stated, embedded rule-based systems are the standard approach to adaptation for most problem domains and system tasks. CBR-AR extracts an adaptation rule-set from knowledge implicitly stored in the CB, thereby eliminating the knowledge elicitation bottleneck associated with manually constructing a set of rules. In terms of Richter's four knowledge containers, this amounts to populating the adaptation container with knowledge from the other three: the structure of adaptation rules is derived from the vocabulary container, the rules themselves are generated from the CB container, and the process of comparing each case with its *k*-NNs relies on knowledge from the similarity container.

The original research ([Hanney and Keane 1996]) demonstrated the operation of CBR-AR in a synthetic regression domain (i.e., the housing domain). It is interesting to note, however, that the underlying methodology was envisaged as a generic adaptation framework that could be applied to any system task. In principle, adaptation knowledge can be mined from a CB in any domain for any task; in practice, the amount of implicit adaptation knowledge in a CB is generally insufficient for more complex tasks such as design and planning. The reason is that the search space of potential paths from an NN to a query increases exponentially with task complexity, and it becomes increasingly unlikely that the differences between two prior cases can be directly used to bridge the gap between them. We return to this theme in Chapter 8.

Local and global domain knowledge is combined during problem solving: Instance-based algorithms such as CBR and *k*-NN base their problem solving abilities on the similarity assumption that similar problems have similar solutions. CBR-AR maintains this bias by taking an NN and its solution as the starting point for solving new queries. However, the problem solving process supplements this local knowledge with global knowledge from other parts of domain space (in the form of adaptation rules). As already noted, maximizing the amount of *relevant* domain knowledge that is brought to bear during problem solving is likely to maximize the quality of solutions in a knowledge-based system.

Predictions can be accompanied by good explanations: Among the advantages of CBR listed in Section 3.1.2 is the fact that each prediction can be accompanied by the concrete cases used to derive it; these can be used by the user to assess the likely quality of the solution. CBR-AR can not only present the user with a neighbouring case similar to the query, but can also show how the gap between the two was bridged. The user can verify that the NN is indeed similar to the query, and that its solution is reasonable and likely to be close to the query's. The rules used to account for the difference between the two cases can also be examined and assessed. Presenting a fully transparent explanation of how the target value was obtained allows the user to make an informed decision about how much confidence to place in it.

Balancing these advantages are a number of limitations that impair CBR-AR's performance in real-world domains. One fundamental problem with adaptation rules is that when global domain knowledge is used to supplement local problem solving, how can we be sure that this global knowledge is actually relevant? After all, instance-based algorithms rely on local knowledge precisely because cases from more distant parts of domain space are dissimilar to the query and so unlikely to be useful in solving it. The first three limitations listed below describe how inappropriate use of global knowledge will make predictions worse, not better. The final two are of a more practical nature, and relate to the manner in which adaptation rules are chained together to bridge the gap between a query and its NN.

Failure to account for interacting attributes: Section 4.3.1.2 showed a simple example of adaptation rules in action. Two rules bridged the gap between a query and its NN, each dealing with one of two attribute differences. Where two or more attributes interact with one another, however, this approach will not work. For example, the effect of increasing *numBedrooms* by 2 and decreasing *location* by 1 may be less than the impact of these changes individually. One can also imagine domains in which combining attribute differences increases their influence. In short, using a combination of simple rules to handle multiple attribute differences is only valid when there is no interaction between the attributes. Hanney and Keane proposed using expert knowledge to edit the rule-base to take account of these interactions. But there may be interactions between several attributes, and their strength may vary in different parts of the attributes' ranges. In these circumstances, editing the rule-base manually becomes an impossible task.

Failure to account for variable attribute relevance: Generalised rules created using Strategy 1 (see Section 4.3.1.3) are not always valid because the strong assumption is made that the solution's response to changes in the problem attribute is *linear*. The example used to illustrate Strategy 1 makes the assumption that changes in *numBedrooms* from 2→3, 3→4, 4→5 and 5→6 will result in the same change to *housePrice*. Clearly this may not be the case; changing *numBedrooms* from 1→2, for example, may have a greater impact on *housePrice* than changing from 3→4 (or vice versa). Strategy 1 cannot be applied where attributes have variable relevance through their range. Again, Hanney and Keane suggested using expert knowledge to take attribute relevance into account during rule-base construction.

Failure to allow for contextual dependencies: Strategy 2 (see Section 4.3.1.3) relaxes the context in which rules may be applied. Although the resulting rules have wider applicability than their predecessors, it is often the case that a rule's correctness is inextricably linked to its context; ignoring the context invalidates the rule. For example, the effect of changing *numBedrooms* from $1 \rightarrow 2$ will very probably be different in *location* 1 than in *location* 4. Where there are contextual dependencies of this type, Strategy 2 cannot be applied. Hanney and Keane suggest using expert knowledge to manually identify and account for these dependencies.

Additive solution errors when more than one adaptation rule is used: In noisy domains, solutions to cases in the CB will not always be accurate. Let us assume that solutions are distributed normally about their true values with variance v^2. Let us also assume that errors are independent of problem attribute values (i.e., the error function is *homoschedastic*). When two cases are used to construct an adaptation rule, the rule's solution difference will have error variance equal to $2 \times v^2$ (subtracting one normal variable from another results in a distribution whose variance is the sum of the individual variances [Upton and Cook 2002]). If r adaptation rules are used to cover the differences between a query and its NN, the predicted solution is calculated by adding the solution difference in each rule to the NN's solution. This prediction has error variance of $v^2 + r \times (2 \times v^2)$ (the initial v^2 is the error in the NN's solution). Where predictions are made using 'adaptation paths' made up of several adaptation rules, results will exhibit a high error variance. This will result in poor performance in noisy real-world domains where v^2 is high.

Problem solving is too brittle: To succeed in solving a query, CBR-AR must chain together rules that account for all of the attribute differences between the query and an NN. With linear, noise-free datasets and sufficient numbers of adaptation rules to cover every possible set of attribute differences, this strategy will return a perfect solution every time. With real-world datasets subject to uncertainty and noise, perfect solutions are simply impossible to find. Making predictions is a process of educated guesswork, and the best that can be hoped for in any domain is that estimated solutions approximate the true values with an acceptable statistical error. CBR-AR is therefore overly ambitious in attempting to find perfect solutions in imperfect domains. Failing to account for all attribute differences should not cause problem solving to fail when a less exacting approach would probably yield equally good results.

These limitations can be amalgamated into two fundamental difficulties:

1. *CBR-AR will only work correctly in linear domains* (i.e., domains that can be accurately represented by a linear model). Many real-world domains do not exhibit global linearity, including the real-world housing domain. Sources of non-linearity include attributes interacting with one another and attributes whose relevance varies throughout their range.
2. *Problem solving with CBR-AR is not robust*. Adaptation rules derived from real cases are noisy and error-prone. Chaining rules together to cover all differences between a query and its NN compounds the error in the resulting solution and aims for a level of accuracy that is not achievable.

4.3.3 Prior research on using case differences for adaptation

Mining adaptation knowledge from the CB is clearly a good idea despite the fact that using it to supplement local problem solving is not straightforward in practice. Given that the underlying idea is sound, it is not surprising that a number of researchers have explored different variants of CBR-AR.

The research described in [McSherry 1998] is most closely analogous to Hanney and Keane's in that it also considers the problem of automatically learning adaptation knowledge for regression (and also uses an artificial housing domain for demonstration purposes). Where a query and its NN differ in a *single attribute*, the impact of this difference on the solution is found by examining its effect on two stored cases; this technique is referred to as the *difference heuristic*. The problem of applying case differences in non-linear problem domains is explicitly recognized, and partially avoided by insisting that all cases involved in problem solving (including the query) have exactly the same values for all but one of the set of interacting attributes. Application of the difference heuristic also requires that all cases in the CB strictly dominate one another (i.e., cases are ordered in such a way that each case's attribute values are greater than or equal to those of its predecessors)—this condition does not hold in real-world datasets.

[Jarmulak et al. 2001] and [Wiratunga et al. 2002] looked at the introspective learning of adaptation knowledge for a design task (tablet formulation). Differences between stored cases are stored together with contextual information in *adaptation cases* that contain similar information to Hanney and Keane's adaptation rules. Attribute interactions and constraints are also mined automatically from the case base. This research addressed the problem of non-linearity in problem domains by proposing that adaptation cases generated from cases close to the query should be preferred to those generated from more distant cases. This strategy makes the implicit assumption that the area of domain space around the query is locally linear; limiting the search for difference-cases to this region therefore avoids the problem.

The difficulty with these attempts to avoid non-linearity is that they greatly reduce the search space for cases to use for problem solving. Furthermore, the restrictions imposed are inflexible in that they do not take into account the individual characteristics of different domains and queries. With the approach of Jarmulak et al., there may be many cases outside the immediate vicinity of the query whose differences are applicable to the query and its NN. For example, the difference in *housePrice* resulting from an increase in *numBedrooms* from 1→2 may very well be the same (on average) as that resulting from an increase from 5→6. A similar narrowing of the search space occurs with the McSherry approach; attempting to hold interacting attributes' values constant will most likely result in a failure to find any applicable case differences in domains where there are several such attributes.

A more recent variation on the Hanney and Keane approach was proposed in [d'Aquin at al. 2006]. This suggested replacing each problem attribute by a set of Boolean attributes, and then searching the differences between cases for *frequent itemsets*. Adaptation rules are not derived from pairs of concrete cases, but instead contain sets of differences that enjoy a certain level of *support* in the CB. These are higher level, more generalized rules than those in CBR-AR, where rules consist of the raw differences between actual cases or minor generalizations of them. The assumption is that since a rule holds true for a certain proportion of the CB, it will probably hold for new query cases too. This approach appears to ignore the specific adaptation needs of individual queries, but no experimental evaluation was carried out and so its effectiveness is unknown.

4.4 Discussion

This chapter has described three different ways of performing regression using CBR. The first, CBR-B, predicts the target value for a query by simply taking the weighted average of solution values among its NNs. As shown in Table 4.2, this algorithm has two shortcomings that the other algorithms, CBR-D and CBR-AR, seek to address.

Table 4.2: Problem solving capabilities of CBR-based regression algorithms

Algorithm	*Takes relative positions of cases in domain space into account*	*Supplements local problem solving with global adaptation knowledge*
CBR-B	✗	✗
CBR-D	✓	✗
CBR-AR	✓	✓

CBR-D takes the relative positions of cases into account by attempting to find a set of NNs with the query at its centre. These cases should give better quality predictions than the query's immediate neighbours (this is borne out in the experimental results in Chapter 7). CBR-AR considers the precise differences between the query and its NN and attempts to account for these differences using one or more adaptation rules. This approach takes the exact positions of the two cases into account, but unfortunately it is not robust for the reasons given at the end of Section 4.3.2.

CBR-D solves each new query using only knowledge from its local neighbourhood. In theory, bringing additional relevant domain knowledge to bear should improve the quality of predictions. CBR-AR attempts to do this by applying adaptation rules derived from cases in all parts of domain space. As pointed out in Section 4.3.2, however, inappropriate use of global domain knowledge is likely to make predictions worse, not better. In particular, CBR-AR's use of adaptation rules is only directly applicable in linear problem domains—the original research called for expert knowledge to be manually added if the technique was to be used in non-linear domains.

This analysis leads to a clarification of the requirements for a more effective regression system based on CBR—these are shown in Table 4.3.

Table 4.3: Requirements for a new CBR-based regression system

Requirement	*Description*
Accuracy	Prediction accuracy should exceed that of alternative CBR-based approaches on a range of datasets.
Robustness	System should perform well in noisy, real-world domains.
Simplicity	System should minimize the need for explicit expert domain knowledge. Predictions should also be easily explainable to users.

The CBR-CD algorithm was developed to meet these requirements. It is based on CBR-AR in the important respect that it bridges the gap between a query and its neighbours using adaptation knowledge mined from the CB. It shares the advantages of this approach in taking the relative positions of cases into account and supplementing local problem solving with global knowledge. However, it also systematically addresses the limitations of CBR-AR to give greater accuracy and robustness. In particular, it takes a greater degree of care to ensure the quality and relevance of the global knowledge used to bridge the gap between a query and its neighbour. This is achieved by using LWLR as a heuristic to guide the search for cases used in problem solving. Robustness is also improved by averaging several intermediate predictions to make a final prediction, thereby reducing the overall error variance.

4.5 Summary

This chapter looked at how CBR can be used for regression. Three different approaches were described in detail:

CBR-Basic (CBR-B): This is the simplest approach. A query's solution is predicted by taking the weighted average of solutions from a number of its NNs.

CBR-Diverse (CBR-D): This is a modified version of CBR-B. It tries to address the problem that a query's NNs may be unevenly distributed around it, resulting in a prediction that is biased in a particular direction. The solution is to aim for a diverse set of NNs with the query at their centre.

CBR-AdaptationRules (CBR-AR): This algorithm takes a traditional CBR approach to adaptation: it predicts a query's solution by using a set of rules to adapt the solution from a neighbouring case. This rule-set is automatically generated from the differences between pairs of stored cases during an initial training phase. Despite being an intuitive and appealing approach to regression, CBR-AR suffers from two fundamental problems: it will only work correctly in linear domains, and it is not robust in the presence of noisy data.

Prior research into the generation of adaptation knowledge from case differences has failed to address the problems inherent in CBR-AR. This thesis therefore proposes a new algorithm, CBR-CD, that builds on the strengths of previous approaches while avoiding their shortcomings.

Chapter 5

Case-Differences Regression Algorithm (CBR-CD)

CBR-CD is a new CBR-based regression algorithm. Its goal, as set out in Section 4.4, is simply to provide effective and robust performance across different problem domains. The primary idea underlying CBR-CD is to use adaptation knowledge mined from the differences between stored cases to bridge the gap between queries and their neighbours. In this respect it builds on earlier research, particularly that proposed by Hanney and Keane [Hanney and Keane 1996, 1997] and embodied in CBR-AR. However, it differs significantly from CBR-AR in its treatment of case differences and in its approach to adaptation. In particular, its adaptation process uses a second, embedded CBR system instead of a more traditional rule-based system. This mechanism allows CBR-CD to avoid the limitations of CBR-AR (listed in Section 4.3.2) and to operate effectively and robustly in noisy, non-linear problem domains.

5.1 Introduction to Case Differences

This section looks at how adaptation knowledge can be mined from the differences between stored cases. It also shows how case differences can be applied to solve regression problems in a simple domain. This 'first attempt' at using case differences is subject to a number of limitations that must be overcome before the technique can be applied in real-world domains. These limitations are discussed, and solutions proposed in Section 5.2.

5.1.1 Constructing an embedded CBR system to use for adaptation

Let us assume for the moment that each case is stored as a vector of numeric attributes. The difference between any two cases can then be calculated simply by subtracting one from the other.

Returning to the simple artificial housing domain introduced in Chapter 4, recall that the value of a house is a function of its number of bedrooms and location:

housePrice = *f*(*numBedrooms*, *location*)

where problem attributes *numBedrooms* and *location* have range 1–6. Given two sample cases of form (*numBedrooms*, *location*, *housePrice*),

C1 = (*4*, *1*, *320000*),
C2 = (*3*, *2*, *300000*),

the differences between them can be calculated and stored in a *difference-case*:

Δ(**C1**, **C2**) = **C1** – **C2** = (*1*, *-1*, *20000*).

This difference-case states that an increase of 1 in *numBedrooms* and a decrease of 1 in *location* results in an increase of 20000 in *housePrice*. It encapsulates specific adaptation knowledge that may be applied to solve new problems. (It is equivalent to a context-free adaptation rule in CBR-AR—see Section 4.3.1.1.) A difference-case can be generated from each pair of cases in the CB, $\mathbf{C_i} - \mathbf{C_j}$. All difference-cases can then be stored together in their own *Difference-CB*. As with CBR-AR, storage requirements can be reduced by only comparing each case with a limited number of neighbours; for a CB of size *n*, comparing each case with its 10 NNs produces a Difference-CB with size *n* x *10*. Note that although this reduces the problem solving capability of the Difference-CB, the adverse effect is minimal provided that the CB is representative of the cases typically encountered during operation.

The Difference-CB constitutes the heart of the embedded CBR system that is used for adaptation. Each difference-case has the same structure (i.e., the same attributes) as cases in the original CB; vocabulary knowledge is therefore transferred directly from the original to the Difference-CB. Because of this, the mechanism for assessing similarity in the original CB also works for the Difference-CB, and it too can be appropriated directly. This only leaves the question of adaptation to be resolved: how does the embedded system adapt the solutions of retrieved difference-cases? The answer is that it doesn't—the embedded system performs null adaptation (i.e., it is retrieval-only). Figure 5.1 shows the overall structure of a CBR-CD system and the sources of knowledge for its constituent parts.

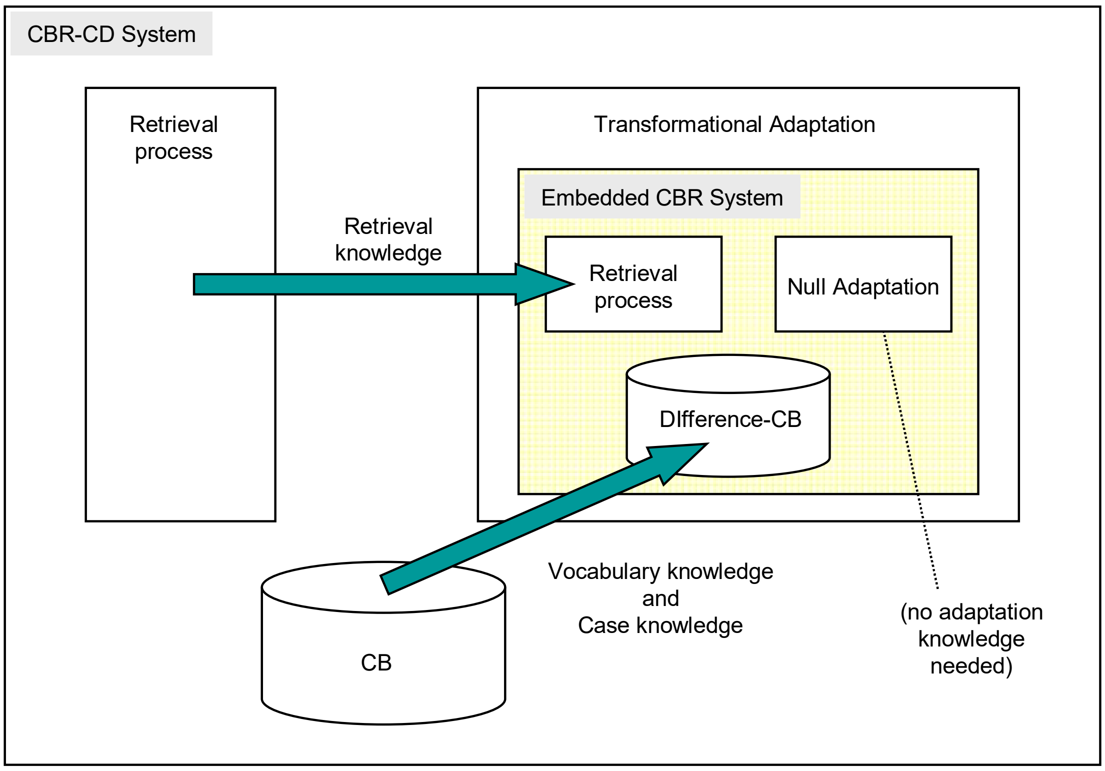


Figure 5.1: Structure of a CBR-CD system

5.1.2 First attempt at using case differences to make a prediction

Case differences can be used to solve a new query problem as follows: calculate the differences between the query and a neighbouring case, then account for these differences using a stored difference-case.

Let us take an example from the housing domain: suppose we receive a query case

$\mathbf{Q} = (4, 2, ?)$,

so that our task is to predict the value of a house with 4 bedrooms in location 2. The problem solving process then proceeds as follows (see Figure 5.2):

1. Retrieve the nearest neighbour to **Q**: $\mathbf{NN} = (5, 1, 350000)$
2. Find the difference between their problem descriptions: $\Delta(\mathbf{NN}, \mathbf{Q}) = (1, -1)$
3. Retrieve a difference-case from the Difference-CB to account for these differences: $\Delta(\mathbf{C}_1, \mathbf{C}_2) = (1, -1, 20000)$ for some $\mathbf{C}_1$, $\mathbf{C}_2$ in the original CB
4. Predict the target value for **Q**: *350000 – 20000 = €330,000*

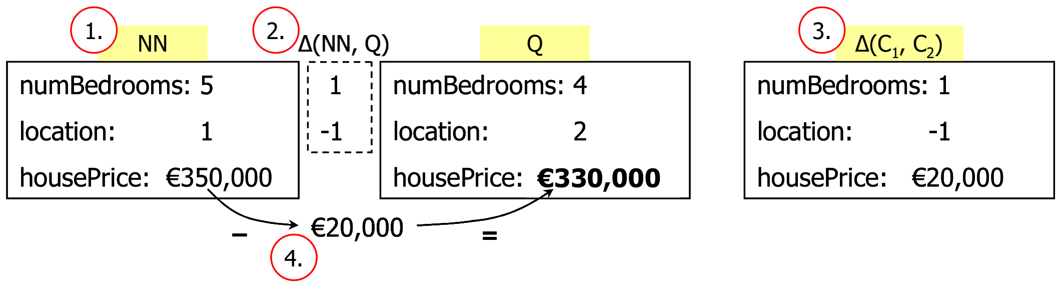


Figure 5.2: Solving a query using case differences

The problem solving process involves calculating the differences between **Q** and **NN**, then using a difference-case to predict the effect that these differences will have on the solution. We will refer to the three cases used for each prediction, (**NN**, **C**$_1$, **C**$_2$), as an *adaptation triple*. (These cases are referred to as an 'estimation triple' in [McSherry 1998].)

Figure 5.3 shows the flow of information within the system during the problem solving process; circled numbers 1–4 refer to the four steps in Figure 5.2. The important point to note is that transformational adaptation in the main CBR system is achieved by performing retrieval and null adaptation in the embedded system. This is a simple form of *recursive CBR* [Stahl and Bergmann 2000]: the original problem (predicting the target value for **Q**) is reduced to the sub-problem of predicting the impact that **Δ(NN, Q)** will have on the solution, and this sub-problem is solved using an embedded CBR system.

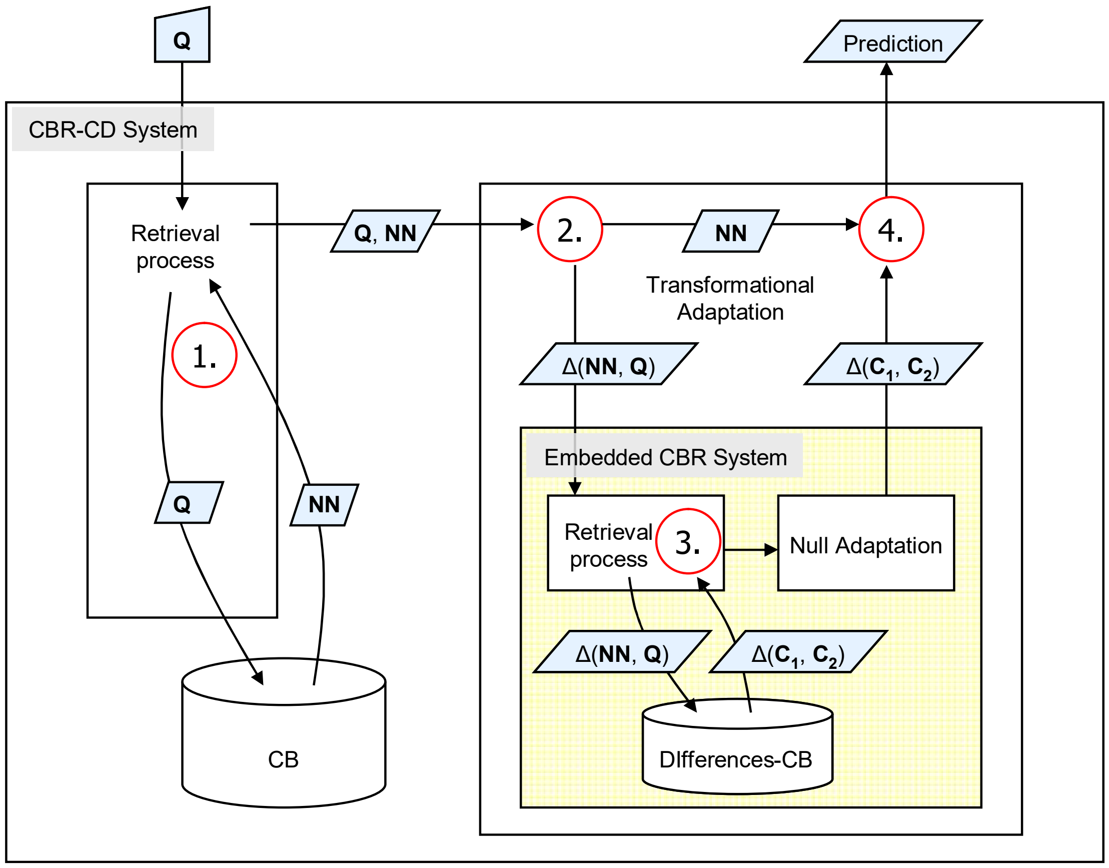


Figure 5.3: Flow of data through a simplified CBR-CD system during problem solving

5.1.3 Problems associated with naïve application of case differences

The basic principles and structures of CBR-CD are now in place. Differences between stored cases are used to generate a Difference-CB that forms the core of an embedded CBR system used for adaptation. As presented above, however, the algorithm is subject to the same drawbacks as CBR-AR: it will only work correctly in linear domains, and it is not robust in noisy domains.

It has already been noted that many real-world domains do not exhibit global linearity; non-linearity can be caused by interactions between attributes and by variable relevance throughout an attribute's range. The simple algorithm presented above makes the strong assumption that the target domain is linear—it assumes that the differences between any pair of stored cases can be used to account for the differences between the query and a neighbouring case. This limitation restricts the applicability of the algorithm, and must be addressed if CBR-CD is to fulfil its goal of operating successfully in all regression domains. Previous research involving case differences has suggested taking steps to *avoid* the problem of non-linearity by restricting the search for adaptations cases to the region of domain space close to the query. As pointed out in Section 4.3.3, this approach greatly reduces the search space for adaptation knowledge in an inflexible way that fails to take account of the individual properties of either the domain or query. What is needed is some way of dealing with non-linearity that maximizes the number of difference-cases that can be considered when searching for solutions, and that adapts automatically to the topology of the problem domain and the specific characteristics of the query.

Lack of robustness is also a problem because each prediction is calculated from three stored cases in such a way that the total error variance among predictions is three times the error variance among cases (more on this later in Section 5.2.2). This will result in poor performance when predictions are based on cases from noisy datasets. (Note that the problem is not as severe as for CBR-AR, which chains together adaptation rules to make a prediction—see Section 5.3.2.) Again, this limitation has to be addressed if CBR-CD is to achieve its goal of performing well on real-world datasets.

5.2 Problem Solving Methodology of CBR-CD

Difference-cases have been shown to provide a simple method for performing regression using CBR. Unfortunately, their naïve application may not perform well in non-linear domains, because *differences between cases in one part of domain space may not have the same effect on the solution as differences in another*.

Having calculated the differences between the query and a neighbour, then, our aim is to find a difference-case that correctly accounts for these differences while also taking non-linearity in the problem domain into account. Locally weighted linear regression (LWLR) can help us achieve this goal. LWLR can act as a useful heuristic in two ways:

1. LWLR can identify those difference-cases most likely to be useful for solving any particular query;
2. LWLR can reduce prediction error by helping to avoid noisy cases.

Sections 5.2.1 and 5.2.2 discuss each of these aspects in turn. Section 5.2.3 describes how overall prediction error can be reduced by combining several predictions. Section 5.2.4 completes the description of CBR-CD by looking at how non-numeric attributes can be accommodated in the problem solving process.

5.2.1 Using LWLR to help choose difference-cases

Let us begin by re-examining the example used in Section 5.1.2. The differences between query **Q** and neighbouring case **NN** were calculated as

$\mathbf{\Delta(NN, Q)} = (1, -1)$.

A difference-case was found to predict the effect of these differences on *housePrice*:

$\mathbf{\Delta(C_1, C_2)} = (1, -1, 20000)$.

The question is, Should this difference-case actually be applied here? Logically, the answer is that it applies if the impact that changes in *numBedrooms* and *location* have on *housePrice* is the same in the area of domain space around **Q** as in the area around $\mathbf{C_1}$ and $\mathbf{C_2}$. More generally, *difference-cases can be applied to solve a query if the cases used in their construction come from an area of domain space similar to that around the query, where similar areas of domain space are those where changes in problem attributes have similar impact on the solution.*

That is, similar areas of domain space have similar rates of change of solution with respect to each problem attribute.

Suppose the problem domain has numeric problem attributes $a_1, a_2, \ldots, a_n$ and solution y, i.e. $y = f(a_1, a_2, \ldots, a_n)$. Each case then occupies a particular point in domain space given by its vector of attribute and solution values $(a_1, a_2, \ldots, a_n, y)$, and f represents a scalar field mapping $\Re^n$ to $\Re$. At any point, the rate of change of f with respect to each problem attribute a_i is defined as the *gradient* of f:

$$grad(f) \equiv \nabla f = \left(\frac{\partial y}{\partial a_1}, \frac{\partial y}{\partial a_2}, \ldots, \frac{\partial y}{\partial a_n} \right)$$

This vector cannot be calculated precisely because the actual form of the target function is unknown. However, since the gradient is defined as the best linear approximation to f at any particular point in $\Re^n$, *it can be approximated at any point by constructing a local linear model using LWLR and taking the slope for each problem attribute.* So local linear modelling allows us to estimate the gradient of the target function at any point, and gradients help us to choose difference-cases that are likely to be applicable to the query.

Calculating the gradient at a particular point **P** involves retrieving a number of nearby cases and fitting a local linear model to them as closely as possible. Cases are weighted by distance so that the closest fit is provided for cases closest to **P**. The linear model has the form shown in Equation 2.1: $\hat{y} = \alpha + \beta v_1 + \gamma v_2 + \ldots + \delta v_n$. Parameters β, γ, δ in this equation give the slope of each problem attribute and can be combined in a vector to give the gradient at **P:** $\nabla f(\mathbf{P}) = (\beta, \gamma, \ldots, \delta)$. Note that the optimal number of cases to use when constructing the linear model will vary with the dataset concerned—in the experiments in Chapter 7, 5%–20% of the cases in each dataset were used. (See Section 2.3 for more information on LWLR.)

The principle behind this approach is demonstrated in Figure 5.4, which shows a simple domain in which solution *y* is a function of a single problem attribute, *a*. Local linear models L1, L2 and L3 are constructed from cases in areas A1, A2 and A3 respectively. (Note that linear models are lines in simple two-dimensional domains, planes in three-dimensional domains, and hyperplanes in domains of higher dimension.) Difference-cases derived from cases in A1 are likely to be useful in solving query **Q** because they come from its immediate vicinity where a degree of local linearity exists. Difference-cases derived from cases in A3 are also likely to be useful because they come from a region of domain space where changes in *a* have a similar effect on *y* as in A1; this is shown by the fact that the slopes of L1 and L3 are similar. Cases from A2, on the other hand, are unlikely to be useful because the slopes of L1 and L2 are dissimilar.

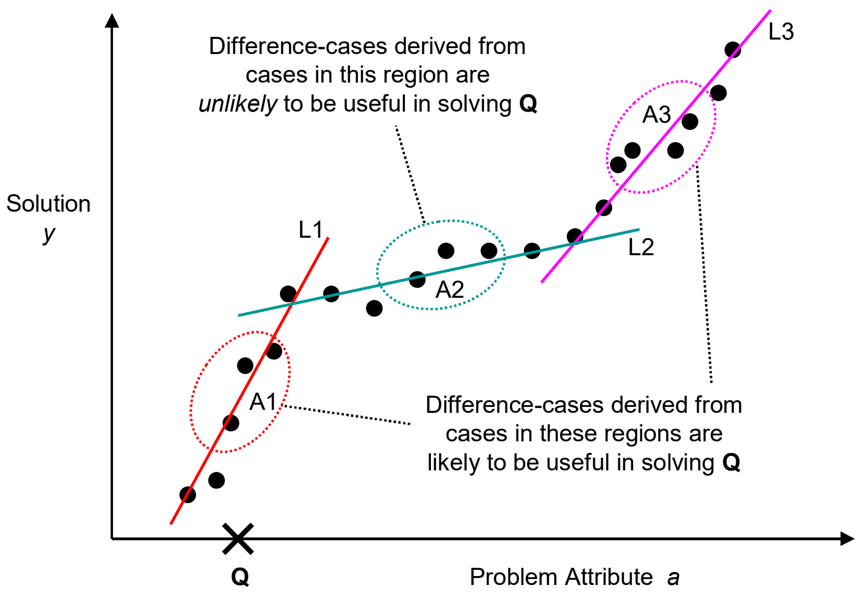

Difference-cases derived from cases in this region are *unlikely* to be useful in solving **Q**
L3
A3
Solution *y*
L1
A2
L2
A1
Difference-cases derived from cases in these regions are likely to be useful in solving **Q**
Q
Problem Attribute *a*

Figure 5.4: Using gradients to help find a useful difference-case (Approach 2)

This simple domain is globally non-linear, but also contains three distinct regions that are approximately linear. Previous algorithms based on case differences have used two alternative approaches to solving **Q**:

1. Assume the entire domain is linear, and use the closest matching difference-case derived from any pair of cases;
2. Recognize that the domain is non-linear, and only use difference-cases derived from cases close to the query in area A1.

Using gradients to guide the search for difference-cases allows CBR-CD to be used in problem domains with any topology. In linear domains, all areas will have a similar gradient and difference-cases from anywhere can be used effectively. As non-linearity increases in the problem domain, the area of domain space where applicable difference-cases may be found gradually reduces in size.

Similarly, the query may come from an area with a similar gradient to many others, in which case the search for difference-cases will be conducted throughout broad regions of domain space. If, on the other hand, the query is located in an area whose gradient is unique, the search will largely be restricted to its immediate neighbourhood.

These properties of CBR-CD ensure that the maximum number of difference-cases is considered for each individual query and problem domain; this is the flexibility that was called for in Section 5.1.3. The algorithm takes care to ensure that when global adaptation knowledge derived from cases some distance from the query is used, it is actually applicable to the problem at hand.

The updated problem solving process can be summarized as follows:

> *To solve a query case, use a difference-case from an area of domain space with similar gradient to that around the query, where gradients are approximated using LWLR.*

The gradient in those regions of domain space containing **Δ(NN, Q)** and **Δ($\mathbf{C_1}$, $\mathbf{C_2}$)** can be approximated by taking the gradient at points **Q** and $\mathbf{C_2}$ respectively.

This section has outlined the rationale for including gradients in the search for difference-cases. It has also introduced a practical algorithm that involves searching for difference-cases whose gradients match that around the query. This is the simplest way to incorporate gradients into the search process, and is described in greater detail in Section 5.2.1.2. More sophisticated approaches to choosing difference-cases are also possible in which difference-cases and gradients are combined during the search; these are described in Sections 5.2.1.3 and 5.2.1.4. Section 5.2.1.1 specifies a basic variant of CBR-CD that does not consider gradients at all; this approach serves as a theoretical basis for the others and as a useful benchmark during the experimental evaluation in Chapter 7.

5.2.1.1 Approach 1: gradients not considered at all

Although many real-world domains are non-linear, many others *are* roughly linear and can be tackled using the basic case-differences algorithm described in Section 5.1.2.

Let us present this algorithm a little more formally. Given query **Q**, the problem solving process involves two search steps and an adaptation step:

1. A neighbouring case to the query, **NN**, is found in the original CB;
2. A difference-case **Δ($\mathbf{C_1}$, $\mathbf{C_2}$)** matching **Δ(NN, Q)** is found in the Difference-CB for some $\mathbf{C_1}$, $\mathbf{C_2}$ in the original CB;

3. Solutions from **NN**, **C$_1$** and **C$_2$** are used to predict a target value for **Q**.

Steps 1 and 2 involve searches in the original and Difference-CBs, during which a search case is compared with each stored case to find the most similar. Both CBs have the same set of problem attributes, *A*. The similarity between any two cases **τ** and **ρ** in either CB can be calculated as follows:

$$Sim(\boldsymbol{\tau}, \boldsymbol{\rho}) = \sum_{a \in A} sim(\tau_a, \rho_a) \qquad \textbf{(5.1)}$$

This similarity function is commonly used in CBR; it differs from Equation 3.1 only in that global attribute weights are not used (or equivalently, all weights are set to 1).

Taken together, steps 1 and 2 constitute a twin-objective search where the optimal adaptation triple (**NN**, **C$_1$**, **C$_2$**) for solving a particular query **Q** simultaneously maximizes the similarity between

- Cases: **NN** and **Q**
- Difference-cases: **Δ(NN, Q)** and **Δ(C$_1$, C$_2$)**.

These objectives can be incorporated into a score for each potential triple:

$$Score_1 = Sim(\mathbf{NN}, \mathbf{Q}) + Sim(\boldsymbol{\Delta}(\mathbf{NN}, \mathbf{Q}), \boldsymbol{\Delta}(\mathbf{C}_1, \mathbf{C}_2)) \qquad \textbf{(5.2)}$$

In Step 3, the adaptation triple with the highest score is used to predict the target value for the query, $\hat{y}^{\mathbf{Q}}$:

$$\hat{y}^{\mathbf{Q}} = y^{\mathbf{NN}} - (y^{\mathbf{C}_1} - y^{\mathbf{C}_2}) \qquad \textbf{(5.3)}$$

This approach to choosing difference-cases can be summed up as follows:

The optimal difference-case is most similar to the difference between the query and an NN.

Looking at Figure 5.4, the difference-case used to solve **Q** can come from any part of domain space; it will be derived from two cases which differ in their values for attribute *a* by the same extent as **Q** and its NN.

Note that once the highest-scoring adaptation triple (**NN**, **C$_1$**, **C$_2$**) has been found, the three cases' solutions are combined in a simple formula (Equation 5.3) to produce a prediction for **Q**: **NN** gives a good starting point, and **C$_1$** and **C$_2$** provide an adjustment to allow for the differences between **NN** and **Q**. Approaches 2–4 below use different score formulae for finding the best adaptation triple, but once found, the predicted solution is always calculated in the same way. That is, CBR-CD always produces a prediction using only the solutions from three retrieved cases.

5.2.1.2 Approach 2: difference-cases and gradients considered separately

In arbitrary domains not known to be globally linear, performance is likely to be improved if difference-cases are chosen from areas of domain space with similar gradients to that around the query. Finding the optimal adaptation triple (**NN**, $\mathbf{C}_1$, $\mathbf{C}_2$) then involves a multi-objective search with three objectives: maximize the similarity between

- Cases: **NN** and **Q**
- Difference-cases: $\mathbf{\Delta}$(**NN**, **Q**) and $\mathbf{\Delta}(\mathbf{C}_1, \mathbf{C}_2)$
- Gradients: $\nabla f(\mathbf{Q})$ and $\nabla f(\mathbf{C}_2)$.

As previously discussed, each gradient is a vector that contains the slope of each problem attribute at a particular point. The similarity between two gradients can therefore be calculated as the sum of the local similarities between each pair of attribute slopes. For the gradients at points **Q** and $\mathbf{C}_2$,

$$Sim(\nabla f(\mathbf{Q}), \nabla f(\mathbf{C}_2)) = \sum_{a \in A} sim\left(\left.\frac{\partial y}{\partial a_i}\right|_{\mathbf{Q}}, \left.\frac{\partial y}{\partial a_i}\right|_{\mathbf{C}_2} \right)$$

where $\left.\frac{\partial y}{\partial a_i}\right|_{\mathbf{Q}}$ and $\left.\frac{\partial y}{\partial a_i}\right|_{\mathbf{C}_2}$ are the slopes of attribute a_i at points **Q** and $\mathbf{C}_2$ respectively, and local similarity function $sim(\text{slope}_1, \text{slope}_2)$ is that shown in Equation 3.3 in Section 3.2.1.3.

This strategy can be supported by adding an additional term to Equation 5.2:

$$Score_2 = Score_1 + Sim(\nabla f(\mathbf{Q}), \nabla f(\mathbf{C}_2)) \qquad \textbf{(5.4)}$$

This is the simplest approach to incorporating the gradient into the search for the optimal difference-case; it is the approach described in Section 5.2.1 and shown in Figure 5.4. As mentioned previously, Equation 5.4 defaults to Equation 5.2 in linear domains where all gradients are very similar. As non-linearity increases, the area of domain space where applicable difference-cases may be found gradually gets smaller. The search strategy can be summed up as follows:

The optimal difference-case is similar to the difference between the query and an NN, and comes from an area of domain space with similar gradient to that around the query.

This approach increases the search space for difference-cases over previous algorithms based on case differences. The two approaches described below increase the search space further by considering difference-cases and gradients together.

5.2.1.3 Approach 3: difference-cases and gradients combined in a vector

Looking more closely at difference-cases and gradients, it can be seen that the two are intimately related to one another. Figure 5.5, for example, shows two pairs of cases in two-dimensional space: (**C**$_1$, **C**$_2$) and (**C**$_3$, **C**$_4$). Differences between the problem parts of these two pairs are the same: **Δ**(**C**$_1$, **C**$_2$) = **Δ**(**C**$_3$, **C**$_4$). Because of different variances in both regions of domain space, however, differences between solutions are *not* the same: $\mathbf{\Delta}(y_1, y_2) \neq \mathbf{\Delta}(y_3, y_4)$.

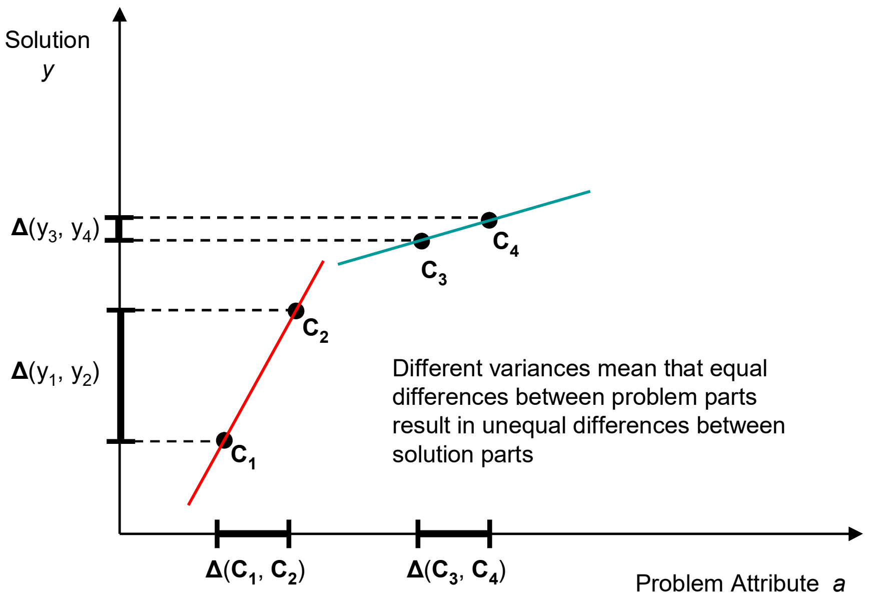

Solution
y
Δ(y3, y4)
Δ(y1, y2)
C4
C3
C2
C1
Different variances mean that equal differences between problem parts result in unequal differences between solution parts
Δ(C1, C2)
Δ(C3, C4)
Problem Attribute a

Figure 5.5: Relationship between difference-cases and gradients

More generally, suppose we achieve a perfect match between **Δ**(**NN**, **Q**) and **DC** = **Δ**(**C**$_1$, **C**$_2$) for some query **Q** and neighbouring case **NN**. If the gradient at **C**$_2$ is higher than that at **Q** for each attribute, then **DC** is likely to overestimate the effect of attribute differences at **Q** on the solution. If it is lower, **DC** is likely to underestimate their impact.

From this we can see that difference-cases and gradients may offset one another, so that even if **DC** has smaller attribute differences than those in **Δ**(**NN**, **Q**), it may still yield a correct prediction if the gradient at **C**$_2$ is higher than that at **Q**. It therefore makes sense to consider difference-cases and gradients together when searching for the optimal adaptation triple.

As already described, each difference-case simply contains the attribute differences between two cases. For two arbitrary cases **τ** and **ρ**, this can be represented as follows:

$$\mathbf{\Delta}(\boldsymbol{\tau},\boldsymbol{\rho}) = (\tau_1 - \rho_1, \tau_2 - \rho_2, \ldots, \tau_n - \rho_n) = (\mathbf{\Delta}\, a_1^{\boldsymbol{\tau},\boldsymbol{\rho}}, \mathbf{\Delta}\, a_2^{\boldsymbol{\tau},\boldsymbol{\rho}}, \ldots, \mathbf{\Delta}\, a_n^{\boldsymbol{\tau},\boldsymbol{\rho}})$$

The *predicted* change in solution *y* resulting from each attribute difference can be calculated from the linear model constructed using LWLR by multiplying attribute differences by attribute slopes. Predicted changes in *y* can be assembled into a vector:

$$\mathbf{\Delta\hat{y}}^{\boldsymbol{\tau},\boldsymbol{\rho}} = (\Delta\hat{y}_1^{\boldsymbol{\tau},\boldsymbol{\rho}}, \Delta\hat{y}_2^{\boldsymbol{\tau},\boldsymbol{\rho}}, \ldots, \Delta\hat{y}_{\mathrm{n}}^{\boldsymbol{\tau},\boldsymbol{\rho}})$$

where $\Delta\hat{y}_i^{\boldsymbol{\tau},\boldsymbol{\rho}} = \left.\frac{\partial y}{\partial a_i}\right|_{\boldsymbol{\rho}} .\Delta a_i^{\boldsymbol{\tau},\boldsymbol{\rho}}$, and $\left.\frac{\partial y}{\partial a_i}\right|_{\boldsymbol{\rho}}$ is the slope of attribute a_i at point $\boldsymbol{\rho}$.

In looking for the optimal adaptation triple (**NN**, **C**$_1$, **C**$_2$), the search process again has two objectives: maximize the similarity between

- Cases: **NN** and **Q**
- Predicted changes in solution resulting from differences in each problem attribute: $\mathbf{\Delta\hat{y}}^{\mathbf{NN,Q}}$ and $\mathbf{\Delta\hat{y}}^{\mathbf{C_1,C_2}}$ (these two vectors combine difference-cases and gradients).

Once again, these objectives can be combined in a score for each potential adaptation triple:

$$Score_3 = Sim(\mathbf{NN},\mathbf{Q}) + Sim(\mathbf{\Delta\hat{y}}^{\mathbf{NN,Q}}, \mathbf{\Delta\hat{y}}^{\mathbf{C_1,C_2}}) \qquad \textbf{(5.5)}$$

This strategy can be summed up as follows:

The optimal difference-case is one where each of its attribute differences has the same predicted impact on the solution as each of the attribute differences in **Δ(NN**, **Q)**.

The effect of this strategy is to further increase the size of the search space for difference-cases. For a given query and NN, Approach 2 searched for matching difference-cases in areas with a matching gradient. Difference-cases meeting both criteria may be difficult to find in practice. Approach 3 awards high scores to all difference-cases that received high scores under Approach 2, but also considers difference-cases from areas of domain space with very different gradients to that around the query. By increasing the size of the search space, Approach 3 improves the likelihood of finding a good solution.

This is illustrated in Figure 5.6. The *predicted* changes in solution resulting from changes in attribute *a* are similar for **Δ(NN**, **Q)**, **Δ(C**$_1$, **C**$_2$**)** and **Δ(C**$_3$, **C**$_4$**)**. (Note that the vertical axis shows *predictions* from linear models in attribute *a*, not actual solution values.) Difference-cases **Δ(C**$_1$, **C**$_2$**)** and **Δ(C**$_3$, **C**$_4$**)** are therefore equally useful for solving **Q** even though they come from areas of domain space with different gradients (shown by the slopes of lines L2 and L3). Approach 2 would only have considered applying **Δ(C**$_3$, **C**$_4$**)**, which comes from an area with similar slope to the query.

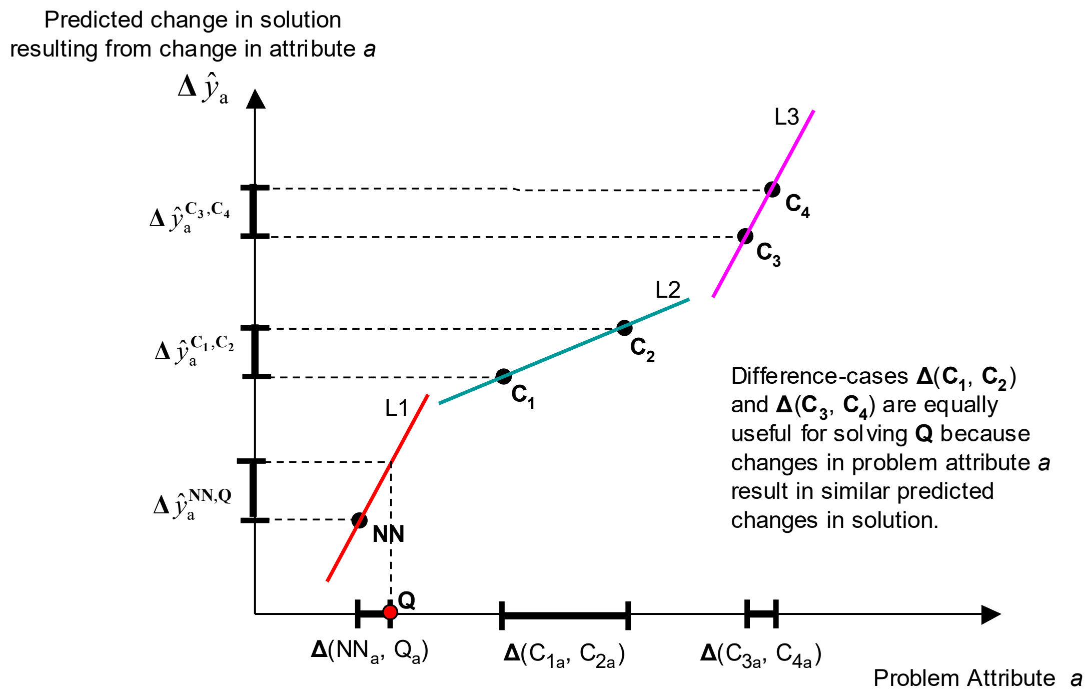

Predicted change in solution resulting from change in attribute a
$\Delta \hat{y}_a$
$\Delta \hat{y}_a^{C_3,C_4}$
$\Delta \hat{y}_a^{C_1,C_2}$
$\Delta \hat{y}_a^{NN,Q}$
L1
L2
L3
NN
Q
C_1
C_2
C_3
C_4
$\Delta(NN_a, Q_a)$
$\Delta(C_{1a}, C_{2a})$
$\Delta(C_{3a}, C_{4a})$
Problem Attribute a
Difference-cases $\Delta(C_1, C_2)$ and $\Delta(C_3, C_4)$ are equally useful for solving Q because changes in problem attribute a result in similar predicted changes in solution.

Figure 5.6: Approach 3: combining difference-cases and gradients in a vector

It is interesting to note that Approach 3 caters for arbitrary non-linear domains, whereas Approach 2 is theoretically limited in the range of domains in can operate in. Consider, for example, the simple domain shown in Figure 5.7. Here the target function *f*(*a*) increases exponentially with problem attribute *a*. This situation is not uncommon in real-world domains; in the housing domain, for example, house price may increase exponentially with location so that upgrading at the more exclusive end of the market commands a bigger premium than upgrading at the lower end. Using Approach 2 is problematic because in an exponential domain, no two areas of domain space will have the same gradient. This strongly biases the search for difference-cases to the region surrounding the query. With Approach 3, however, difference-cases from any part of domain space may be applied because changes in gradient can be offset by changes in attribute differences. In Figure 5.7, for example, difference-cases $\Delta(\mathbf{C_1}, \mathbf{C_2})$ and $\Delta(\mathbf{C_3}, \mathbf{C_4})$ are equally useful for problem solving despite coming from areas with different gradients. Note that in practice, Approach 2 may still be applicable in domains such as this—gradients only provide a heuristic to help in the search for difference-cases, and only constitute one element in a multi-objective search for the optimal adaptation triple.

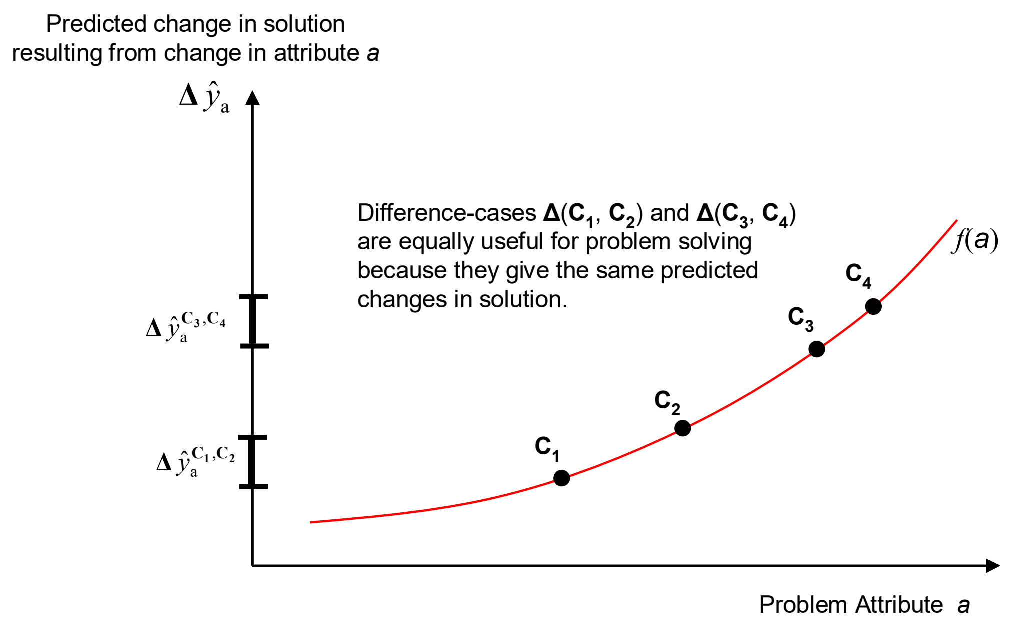

Predicted change in solution resulting from change in attribute *a*
$\Delta\hat{y}_a$
$\Delta\hat{y}_a^{C_3,C_4}$
$\Delta\hat{y}_a^{C_1,C_2}$
Difference-cases **Δ(C$_1$, C$_2$)** and **Δ(C$_3$, C$_4$)** are equally useful for problem solving because they give the same predicted changes in solution.
f(*a*)
C$_1$
C$_2$
C$_3$
C$_4$
Problem Attribute *a*

Figure 5.7: Validity of Approach 3 for arbitrary target functions

5.2.1.4 Approach 4: difference-cases and gradients combined in a scalar

Let us restate the objective of our search: given query **Q** and neighbour **NN**, we are looking for a difference-case **Δ(C$_1$, C$_2$)** that correctly predicts the impact of **Δ(NN, Q)** on the solution.

One way to estimate the overall impact that **Δ(NN, Q)** will have on the solution is to make a prediction using the local linear model constructed around **Q**. Similarly, the impact of each difference-case **Δ(C$_1$, C$_2$)** can be predicted from the linear model constructed around **C$_2$**. If a difference-case can be found that has the same *predicted* impact as **Δ(NN, Q)**, then the *actual* difference in solution given by $y^{C_1} - y^{C_2}$ can be used to solve **Q** (using Equation 5.3).

For any two cases **τ** and **ρ**, the impact of a difference in attribute a_i on solution y was predicted in Section 5.2.1.3 as $\Delta\hat{y}_i^{\tau,\rho} = \Delta a_i^{\tau,\rho} \,.\, \left.\frac{\partial y}{\partial a_i}\right|_{\rho}$ (i.e., the attribute difference multiplied by the slope of the attribute at **ρ**). The overall impact on the solution from *all* attribute differences can be predicted simply by summing their individual contributions:

$$\Delta\hat{y}_{tot}^{\tau,\rho} = \sum_{a \in A} \Delta a_i^{\tau,\rho} \,.\, \left.\frac{\partial y}{\partial a_i}\right|_{\rho} = \mathbf{\Delta(\tau,\rho)} \bullet \nabla f(\boldsymbol{\rho})$$

(Note that this sum is equivalent to the scalar product of **Δ(τ, ρ)** and the gradient of the target function at **ρ**.)

The search for the optimal adaptation triple again has two objectives: maximize the similarity between

- Cases: **NN** and **Q**
- Overall predicted changes in solution resulting from **Δ(NN, Q)** and **Δ(C$_1$, C$_2$)**: $\Delta\hat{y}_{tot}^{\mathbf{NN,Q}}$ and $\Delta\hat{y}_{tot}^{\mathbf{C_1,C_2}}$ (these two scalars combine difference-cases and gradients).

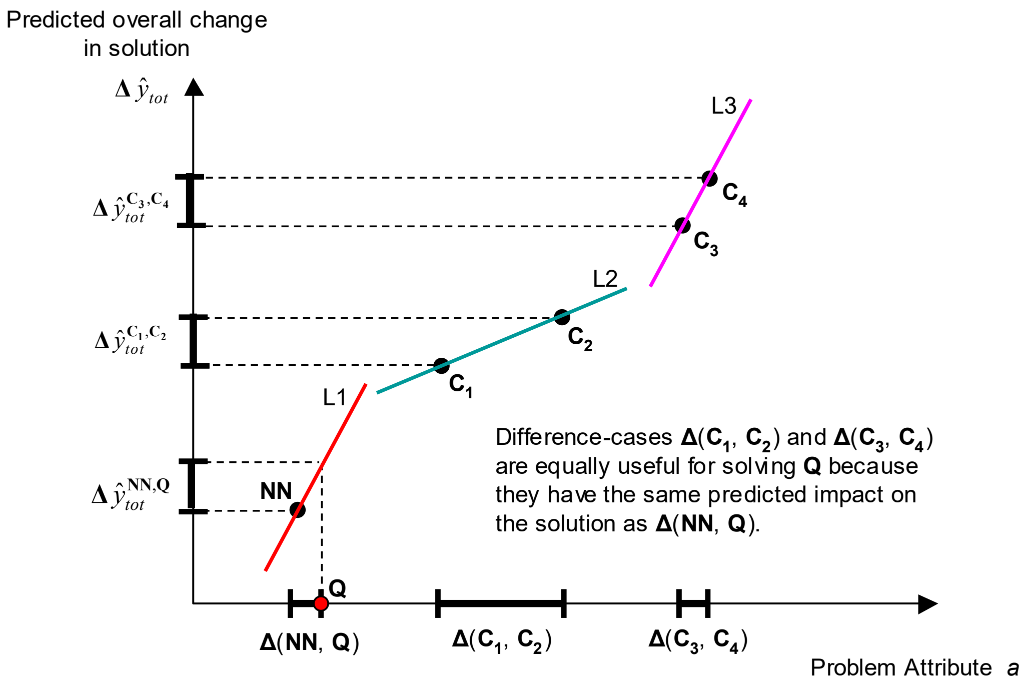
Combining these objectives in a score formula for each potential adaptation triple:

$$Score_4 = Sim(\mathbf{NN,Q}) + Sim(\Delta\hat{y}_{tot}^{\mathbf{NN,Q}},\ \Delta\hat{y}_{tot}^{\mathbf{C_1,C_2}}) \qquad \textbf{(5.6)}$$

This approach can be summed up as follows:

The optimal difference-case has the same predicted overall impact on the solution as **Δ(NN, Q)**.

This is illustrated in Figure 5.8. Problem attribute changes in **Δ(NN, Q)**, **Δ(C$_1$, C$_2$)** and **Δ(C$_3$, C$_4$)** result in similar *predicted* changes in solution *y*. Difference-cases **Δ(C$_1$, C$_2$)** and **Δ(C$_3$, C$_4$)** are therefore equally applicable for solving **Q**. Note that as with Approach 3, the search space for difference-cases includes areas with different gradients from the query and from each other. For this reason, Approach 4 can also be used in arbitrary non-linear domains including those with an exponential target function.



Predicted overall change in solution
$\Delta\hat{y}_{tot}$
$\Delta\hat{y}_{tot}^{C_3,C_4}$
$\Delta\hat{y}_{tot}^{C_1,C_2}$
$\Delta\hat{y}_{tot}^{NN,Q}$
L3
C$_4$
C$_3$
L2
C$_2$
C$_1$
L1
NN
Q
Difference-cases Δ(C$_1$, C$_2$) and Δ(C$_3$, C$_4$) are equally useful for solving Q because they have the same predicted impact on the solution as Δ(NN, Q).
Δ(NN, Q)
Δ(C$_1$, C$_2$)
Δ(C$_3$, C$_4$)
Problem Attribute *a*

Figure 5.8: Approach 4: combining difference-cases and gradients in a scalar

Indeed, the search space considered under Approach 4 is wider than that of Approach 2 or Approach 3. Although difficult to visualize in two dimensions, it can be easily demonstrated with a three-dimensional example from the housing domain. Suppose we have a query and NN with the following characteristics:

$$\mathbf{Q} = (4, 2, ?), \quad \mathbf{NN} = (5, 1, 350000)$$

$$\Delta(\mathbf{NN}, \mathbf{Q}) = (1, -1), \quad \nabla f(\mathbf{Q}) = (50000, 40000); \quad \Delta \hat{y}_{tot}^{\mathbf{NN},\mathbf{Q}} = (1 \times 50000) + (-1 \times 40000) = 10000$$

That is, for this particular query, increasing *numBedrooms* by 1 and decreasing *location* by 1 is likely to produce an increase of approximately €10,000 in *housePrice*.

Now suppose all difference-cases are assessed using Equation 5.6 and the following two achieve the highest scores:

$$\Delta(\mathbf{C}_1, \mathbf{C}_2) = (2, -2), \quad \nabla f(\mathbf{C}_2) = (25000, 20000); \quad \Delta \hat{y}_{tot}^{\mathbf{C}_1,\mathbf{C}_2} = 10000$$

$$\Delta(\mathbf{C}_3, \mathbf{C}_4) = (0, 4), \quad \nabla f(\mathbf{C}_4) = (5000, 2500); \quad \Delta \hat{y}_{tot}^{\mathbf{C}_3,\mathbf{C}_4} = 10000$$

Despite having dissimilar attribute differences and gradients to the query and to each other, both difference-cases have the same overall predicted impact on *housePrice* as **Δ(NN, Q)**. Because of this, Approach 4 considers both to be equally applicable for solving **Q**. This is illustrated in Figure 5.9: **Δ(NN, Q)** (in the centre) has a predicted impact of 10000 on the solution, and so do $\mathbf{\Delta(C_1, C_2)}$ and $\mathbf{\Delta(C_3, C_4)}$ to either side.

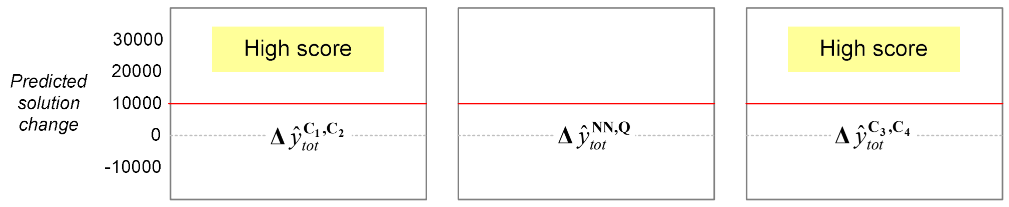


Figure 5.9: Assessing difference-cases using Approach 4

It is instructive to consider how Approaches 2 and 3 would have scored these two difference-cases. Approach 3 differs from Approach 4 in that the predicted impact from each attribute difference is considered separately. For query **Q** and its neighbour **NN**,

$$\Delta(\mathbf{NN}, \mathbf{Q}) = (1, -1), \quad \nabla f(\mathbf{Q}) = (50000, 40000); \quad \Delta \hat{\mathbf{y}}^{\mathbf{NN},\mathbf{Q}} = ((1 \times 50000), (-1 \times 40000)) = (50000, -40000)$$

That is, differences of 1 and −1 in *numBedrooms* and *location* are predicted to cause changes in *housePrice* of 50000 and −40000 respectively. Difference-cases $\mathbf{\Delta}(\mathbf{C}_1, \mathbf{C}_2)$ and $\mathbf{\Delta}(\mathbf{C}_3, \mathbf{C}_4)$ scored equally well under Approach 4, but this is not the case under Approach 3:

$$\mathbf{\Delta}(\mathbf{C}_1, \mathbf{C}_2) = (2, -2), \quad \nabla f(\mathbf{C}_2) = (25000, 20000); \quad \Delta \hat{\mathbf{y}}^{\mathbf{C}_1,\mathbf{C}_2} = (50000, -40000)$$

$$\mathbf{\Delta}(\mathbf{C}_3, \mathbf{C}_4) = (0, 4), \quad \nabla f(\mathbf{C}_4) = (5000, 2500); \quad \Delta \hat{\mathbf{y}}^{\mathbf{C}_3,\mathbf{C}_4} = (0, 10000)$$

Each attribute difference in $\mathbf{\Delta}(\mathbf{C}_1, \mathbf{C}_2)$ can be seen to have the same predicted impact on *housePrice* as each difference in $\mathbf{\Delta}(\mathbf{NN}, \mathbf{Q})$—this difference-case would therefore be awarded a high score. As shown in Figure 5.10, however, individual attribute differences in $\mathbf{\Delta}(\mathbf{C}_3, \mathbf{C}_4)$ have very different predicted impacts on *housePrice* than those in $\mathbf{\Delta}(\mathbf{NN}, \mathbf{Q})$ (i.e., 50000 vs. 0 for *numBedrooms*, −40000 vs. 10000 for *location*). This difference-case would therefore receive a low score under Approach 3.

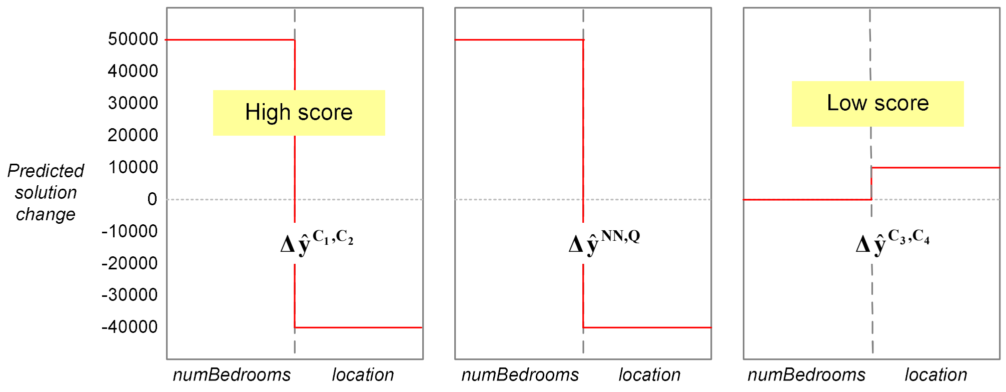


Figure 5.10: Assessing difference-cases using Approach 3

Approach 2 also uses gradients to assess the potential usefulness of cases, but takes a simpler approach that involves comparing difference-cases and gradients separately. In this example,

$$\mathbf{\Delta}(\mathbf{NN}, \mathbf{Q}) = (1, -1) \qquad \mathbf{\Delta}(\mathbf{C}_1, \mathbf{C}_2) = (2, -2) \qquad \mathbf{\Delta}(\mathbf{C}_3, \mathbf{C}_4) = (0, 4)$$

$$\nabla f(\mathbf{Q}) = (50000, 40000) \qquad \nabla f(\mathbf{C}_2) = (25000, 20000) \qquad \nabla f(\mathbf{C}_4) = (5000, 2500)$$

It can clearly be seen in Figure 5.11 that all three difference-cases are dissimilar, and all three gradients are also dissimilar. For this reason, Approach 2 would assign a low score to both difference-cases.

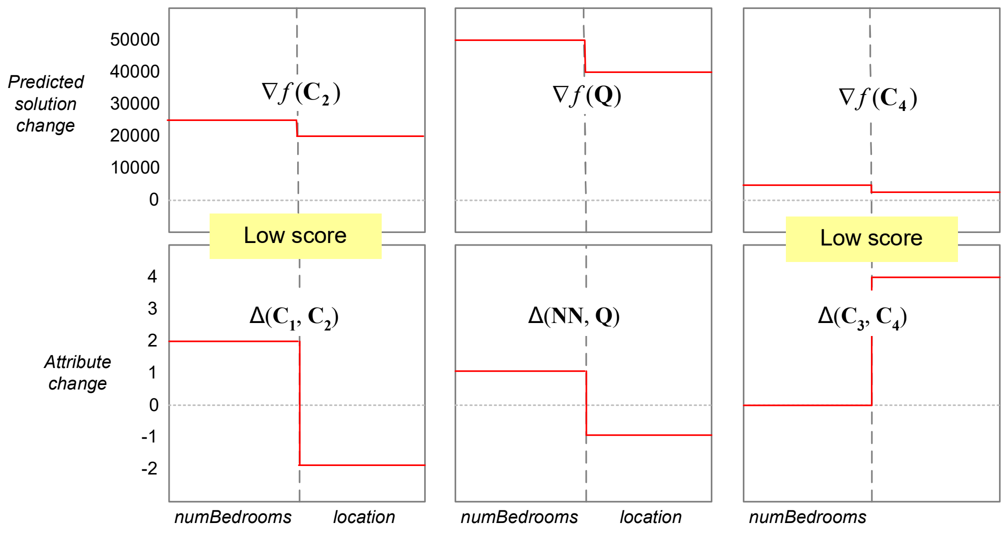


Figure 5.11: Assessing difference-cases using Approach 2

5.2.1.5 Discussion

The preceding subsections describe four different ways to consider gradients when searching for the optimal adaptation triple to solve a query; which one works best will depend on the nature of the problem domain. Approach 1 is simplest and is adequate in linear domains. Constructing a traditional linear regression model is even simpler, however, and is to be preferred when working with domains known to be globally linear.

Approaches 2, 3 and 4 take the gradient of domain space into account, and are likely to give better results in non-linear domains. They differ in the extent to which gradients are integrated into the search for the optimal difference-case. Approach 2 keeps the two separate and tries to find a matching difference-case in an area with a matching gradient. Approach 3 tries to match difference-cases and gradients together on a *per attribute* basis, while Approach 4 matches difference-cases and gradients together on a *whole case* basis. Approaches 3 and 4 take into account the fact that an attribute's slope is only relevant if there are differences in that attribute between two cases. They ignore an attribute's slope if there is no actual difference to account for, and allow the influence of the slope to increase with the size of the difference. Approach 2, on the other hand, always gives equal importance to all attributes' slopes even if there are no (or very small) actual differences between cases for some attributes.

Overall, Approaches 2, 3 and 4 progressively increase the size of the search space for difference-cases, while simultaneously increasing the algorithm's reliance on the gradient heuristic. The appropriateness of each approach will therefore depend on the applicability of this heuristic:

Approach 2: Likely to perform best in highly non-linear or noisy domains, where gradients may offer only limited assistance in choosing difference-cases. These are domains where *k*-NN has also been shown to perform well.

Approach 4: Designed to perform well in domains that exhibit strong local linearity. These are domains where LWLR is also likely to perform well as a prediction algorithm.

Approach 3: Takes a middle path between 2 and 4, and is probably the best general-purpose approach for most problem domains.

The following points regarding the operation of CBR-CD are also interesting to note:

- When gradients are used to help find adaptation triples (in Approaches 2, 3 and 4), the similarity function used to compare cases does not need to use global attribute weights or scaling factors (see Equation 5.1). Attributes' influence on the solution may vary throughout domain space, and the algorithm correctly accounts for this.
- Approach 3 compares difference-cases and gradients on a *per attribute* basis, while Approach 4 performs comparisons on a *whole case* basis. This is strongly reminiscent of the difference between CBR-D and alternative approaches to diversity described in Section 4.2. Approach 4 loses attribute-level information in the process of increasing the search space; whether this trade-off diminishes performance will depend on the domain.
- In the best-scoring adaptation triple (**NN**, $\mathbf{C}_1$, $\mathbf{C}_2$), case **NN** may not be the nearest neighbour to the query. Instead, **NN,** $\mathbf{C}_1$ and $\mathbf{C}_2$ are chosen together so as to maximize their likely usefulness in solving a particular query **Q**. This is another example of adaptation-guided retrieval [Smyth and Keane 1998]—the *adaptability* of neighbouring cases is considered in addition to their proximity to the query.
- Approaches 2, 3 and 4 perform a multi-objective search for adaptation triples by combining different objectives in a single linear score formula. Most multi-objective optimization problems are solved by minimizing a linear cost function $\sum_{i=1}^{n} \alpha_i \, Objective_i$, where weights $0 \leq \alpha_i \leq 1$ are chosen to reflect the relative importance of each objective [Eschenauer et al. 1990]. Different sets of values for α_i produce different minima that are known individually as *Pareto optimal points* and collectively as a *Pareto set.* Plotting these points in a *Pareto curve* allows the optimal set of weights to be found for any particular domain—these weights give best overall performance and express the optimal trade-off between objectives.

In Approaches 2, 3 and 4, all objectives are given equal importance—weight parameters are not present in the score formulae nor are they used in experiments in Chapter 7. While it is important to be aware of the possibility of optimizing performance for each individual problem domain, the intention in this thesis is to present CBR-CD as a general-purpose regression algorithm that performs well without the need for extensive domain-specific tuning.

The basic CBR-CD algorithm is now complete. For any given query, an adaptation triple can be found to predict its target value while taking the characteristics of domain space into account. The first of CBR-AR's two limitations (listed at the end of Section 4.3.2) has therefore been addressed: CBR-CD will work correctly in non-linear domains. The second limitation of CBR-AR, lack of robustness, is addressed below in Sections 5.2.2 and 5.2.3.

5.2.2 Using LWLR to reduce prediction error

Theoretically, CBR-CD will provide good predictive performance in domains with any topology. In practice, however, it suffers from the same robustness problems that beset CBR-AR (see Section 4.3.2). These must be addressed if CBR-CD is to be used with real-world datasets where noise is an unavoidable fact of life.

Let us suppose that we are working with a dataset in which solution errors are distributed normally about their true values with variance v^2. Let us also assume that the error function is homoschedastic (i.e., errors are independent of problem attribute values). As shown in Equation 5.3, each prediction uses solutions from three stored cases: **NN**, $\mathbf{C_1}$ and $\mathbf{C_2}$. Since each has error variance v^2, the prediction error has variance $3 \times v^2$. If we can reduce the error among these three cases, variance in prediction error will be reduced threefold.

Equation 5.3 shows that the prediction mechanism in CBR-CD is highly local in that each prediction is based on only three cases. But in using LWLR to help choose difference-cases, the algorithm also assumes some degree of local linearity in the problem domain. This local linearity can act as a useful guide when choosing cases **NN**, $\mathbf{C_1}$ and $\mathbf{C_2}$. Recall from Section 5.2.1 that estimating the gradient at each case involves using LWLR to construct a local linear model at that point in domain space. This local model can be thought of as an approximation of the target function's mean throughout that area of domain space. Cases lying further from the linear model are more likely to be noisy than those close to it. This provides a simple heuristic:

When choosing (**NN**, $\mathbf{C_1}$, $\mathbf{C_2}$)*, prefer cases that lie closer to the local linear model constructed around them.*

Variance is reduced by biasing the choice of cases towards the local mean.

The basic idea behind this approach is summed up in [Bergmann and Wilke 1998, p. 4]: "applying sound adaptation knowledge to a sound case leads to a sound solution". Here, cases are statistically more likely to be sound if they lie close to the mean solution in their local areas of domain space.

Integrating this strategy into the CBR-CD algorithm is straightforward. For any case $\boldsymbol{\tau}$, the normalized residual r^{τ} is a measure of its closeness to the local linear model constructed around it:

$$r^{\tau} = (y^{\tau} - \hat{y}^{\tau}) \,/\, (y_{\max} - y_{\min})$$

where

y^{τ} is the actual solution value in $\boldsymbol{\tau}$,

$\hat{y}^{\tau}$ is the solution predicted using LWLR,

$y_{\max}$ and $y_{\min}$ are the maximum and minimum solutions values in the CB (used to normalize r^{τ} to the range –1 to +1).

In CBR-CD, we want to avoid noisy cases when choosing **NN**, **C**$_1$ and **C**$_2$. **NN** is a standalone case, and is less likely to be noisy if it has a low residual r^{NN}. This is depicted in Figure 5.12, where selecting **NN** from among the solid points is more likely to result in an accurate prediction because these points lie closer to the local linear model constructed around **Q**.

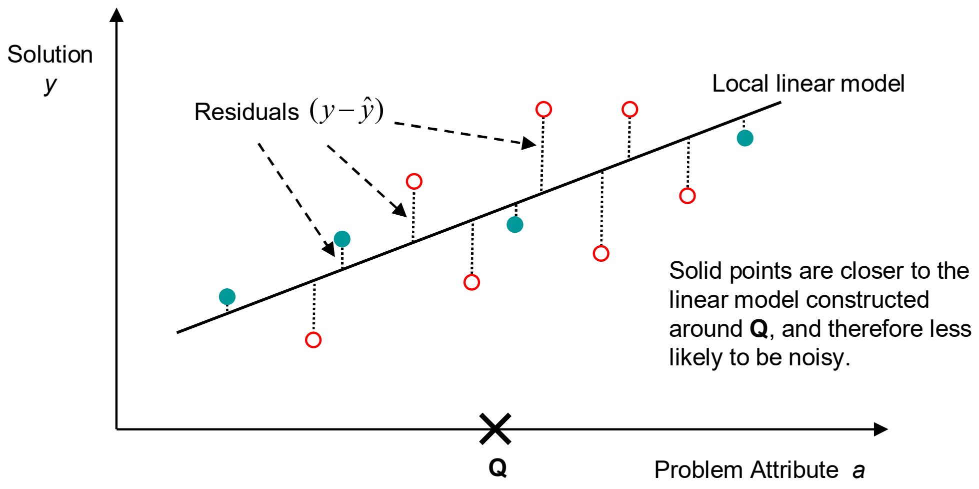

Solution
y
Residuals $(y-\hat{y})$
Local linear model
Solid points are closer to the linear model constructed around **Q**, and therefore less likely to be noisy.
Q
Problem Attribute *a*

Figure 5.12: Using LWLR to avoid noisy cases

Cases **C**$_1$ and **C**$_2$, on the other hand, occur as a pair in domain space, so that if both are offset from their local linear model by the same distance, their predictive quality will not be affected.

For this reason, $\mathbf{C_1}$ and $\mathbf{C_2}$ are likely to give an accurate prediction when the difference of their residuals, $(r^{\mathbf{C_1}} - r^{\mathbf{C_2}})$, is minimized.

Each of the score formulae for Approaches 1–4 in Equations 5.2, 5.4, 5.5 and 5.6 can be altered to take case quality into account:

$$Score_i = Score_i \times (3 - (abs(r^{\mathbf{NN}}) + abs(r^{\mathbf{C_1}} - r^{\mathbf{C_2}})))$$

As before, the adaptation triple with the highest score is used to predict the target value for **Q**.

This completes LWLR's contribution to the CBR-CD algorithm. Previous sections have shown that it plays a key role as a heuristic in guiding the search for applicable difference-cases. This section has shown its usefulness in improving robustness by helping to avoid noisy cases.

5.2.3 Combining multiple predictions to reduce prediction error

When presented with a query case, CBR-CD conducts a search for the highest scoring adaptation triple and uses it to make a prediction. But there may be several adaptation triples with scores almost as high as the one chosen. This raises an opportunity for CBR-CD to increase robustness: make *n* predictions using the best *n* triples, and produce an overall prediction from their weighted average.

If all triples were of equally high quality, averaging the first *n* predictions would reduce the overall variance of the prediction error from v^2 to $\frac{v^2}{n}$. This gain is not in fact achieved because the first prediction uses the highest scoring adaptation triple; predictions based on subsequent triples will, in general, have higher error. The optimal value for *n* will vary with the problem domain—cross validation can be used to choose a value that minimizes overall prediction error.

In terms of implementation, it is a simple matter to alter the search for the optimal adaptation triple so that a list of triples ordered by score is maintained instead. Equation 5.3 can then be modified so that the overall prediction is an average of *n* individual predictions, each weighted by its score:

$$\hat{y}^{\mathbf{Q}} = \frac{\sum_{i=1}^{n} Score_i \times (y^{\mathbf{NN}_i} - (y^{\mathbf{C}_{1i}} - y^{\mathbf{C}_{2i}}))}{\sum_{i=1}^{n} Score_i} \qquad \textbf{(5.7)}$$

5.2.4 Adding support for non-numeric attributes

So far, CBR-CD has assumed that cases are represented as vectors of numeric values. Many practical datasets include non-numeric attributes, however. Nominal attributes that take a number of discrete values, each equally dissimilar to all others, are especially common. CBR-CD can theoretically be extended to accommodate attributes of any type, although it operates best when there is at least one numeric problem attribute.

Let us begin be examining the changes necessary to support unordered nominal (including Boolean) attributes. The first change that is required is in the construction of the Difference-CB. The difference between any two numeric values is found by simply subtracting one from the other. For nominal attributes, the difference between any two values v_1 and v_2 is calculated as follows:

$$\Delta(v_1, v_2) = \begin{cases} 'u' & , \ v_1 = v_2 \\ v_1 ' \to ' v_2 & , \ v_1 \neq v_2 \end{cases} \tag{5.8}$$

In other words, equal nominal values are replaced by the character '*u*', and unequal values are concatenated with the symbol '→' between them.

For example, given two cases $\mathbf{C_1}$ and $\mathbf{C_2}$ with numeric and nominal attribute values,

$\mathbf{C_1} = (\text{'}a\text{'}, 3, \text{'}True\text{'}, 50)$, $\quad$ $\mathbf{C_2} = (\text{'}a\text{'}, 5, \text{'}False\text{'}, 30)$,

the difference between them is

$\mathbf{\Delta(C_1, C_2)} = (\text{'}u\text{'}, -2, \text{'}True \to False\text{'}, 20)$.

Note that the system task is still regression and so the solution remains numeric.

Once the Difference-CB has been constructed, the operation of the prediction algorithm is largely unaffected by the presence of nominal problem attributes. The most basic version of CBR-CD (called 'Approach 1' in preceding sections) does not use gradients to guide the search for difference-cases or avoid noisy cases. Adapting Approach 1 to support nominal attributes simply requires new comparators for comparing nominal values during retrieval in the original and Difference-CBs:

1. Given query **Q**, a neighbouring case **NN** is found using Equation 5.1 as before. Nominal attributes may have their own custom comparators, or may be treated more simply (e.g., *sim=1* for equal values, *sim=0* for unequal values).
2. **Δ(NN, Q)** is calculated using Equation 5.8, and matching difference-case $\mathbf{\Delta(C_1, C_2)}$ is found using Equation 5.1. The embedded CBR system uses a simple formula for comparing two nominal values: *sim=1* for equal values, *sim=0* for unequal values.
3. A prediction is made using solution values from **NN**, $\mathbf{C_1}$ and $\mathbf{C_2}$ as before.

Support can be added for an arbitrary non-numeric attribute by defining custom comparators for the original and Difference-CBs and a function to calculate the difference between two of the attribute's values. For example, suppose the original CB has a matrix attribute where each value is an (n x m) table of numbers. Retrieving an **NN** for a given query requires a comparator to calculate the similarity between two matrices: call it *matrixCompFn*. The difference between two matrices can be calculated by subtracting one from the other—this requires a custom difference function analogous to Equation 5.8. Finally, retrieving a difference-case to match **Δ(NN, Q)** re-uses *matrixCompFn* to compare differences between matrices. It is often the case that the difference between two values of a particular type is also a value of that type (i.e., the set of possible attribute values is *closed under subtraction*), and where this occurs, the comparator function can be re-used in this way.

But what of the many improvements that have been made to the basic algorithm to improve accuracy and robustness? Can these still be used in the presence of non-numeric attributes? The answer is that they can, with one minor change required when dealing with gradients.

Using gradients to choose applicable difference-cases (Section 5.2.1): Gradients are calculated by constructing a local linear model around each case in domain space. Since LWLR only works with numeric values, non-numeric attributes are simply *ignored* and local linear models are constructed using only those attributes that are numeric. This approach is reasonable because *local linear models are not used directly to make predictions*. They merely guide the search for cases (**NN**, $\mathbf{C_1}$, $\mathbf{C_2}$), and continue to perform a useful role even when they are constructed using only a subset of problem attributes. This is borne out in the experimental results in Chapter 7.

Using gradients to avoid noisy cases (Section 5.2.2): Again, the only impact is that non-numeric attributes are excluded from gradients.

Averaging several predictions for an overall prediction (Section 5.2.3): This aspect of CBR-CD is not affected by non-numeric attributes at all.

This section opened by stating that CBR-CD works best when at least one problem attribute remains numeric. The reason for this is now clear: one or more numeric attributes are needed if the gradient heuristic is to be used for choosing difference-cases and avoiding noisy cases. In theory, Approach 1 (which doesn't use gradients) can be used for datasets without any numeric problem attributes although this would eliminate one of the algorithm's principal strengths.

5.2.5 Making a prediction using CBR-CD

We are now in a position to describe the full problem solving process of the CBR-CD algorithm, taking into consideration the many improvements that have been made since our first attempt in Section 5.1.2. Let us re-examine the housing example from that section: suppose we receive the query case

$\mathbf{Q} = (4, 2, ?)$,

so that the task is to predict the value of a house with 4 bedrooms in location 2. The full problem solving process proceeds as follows (see Figure 5.13):

1. The *k* nearest neighbours to **Q** are retrieved. In this example, let *k*=2:

   $\mathbf{NN_1} = (4, 3, 370000)$, $\mathbf{NN_2} = (5, 1, 350000)$

2. The differences between **Q** and each of its NNs is found:

   $\mathbf{\Delta(NN_1, Q)} = (0, 1)$, $\mathbf{\Delta(NN_2, Q)} = (1, -1)$

3. The best *n* adaptation triples for solving **Q** are found. Gradients may be used to select difference-cases (using Approaches 2–4) and to assess the quality of each potential adaptation triple (by avoiding cases distant from their local linear model). Note that because each NN's usefulness is always assessed in the context of an adaptation triple, each NN may appear in several of the best *n* triples or in none. As an implementation issue, Figure 5.13 also shows that gradients can conveniently be stored in their own CB and compared using a standard numeric comparator.) For our example, let us suppose that Approach 2 is being used with parameter *n*=2, so that scores are calculated using Equation 5.4. (Note that the scores shown below are for demonstration purposes only; as discussed in Sections 5.2.1.1 and 5.2.1.2, actual scores would be found using local similarity functions for cases, difference-cases and gradients, and the operation of these functions would in turn be determined by the statistical properties of the entire CB.)

   i. The gradient at **Q** is estimated by constructing a local linear model:

      $\nabla f(\mathbf{Q}) = (50000, 35000)$

   ii. The Difference-CB is searched for the best *n* difference-cases $\mathbf{\Delta(C_a, C_b)}$ matching any of $\mathbf{\Delta(NN_i, Q)}$ where gradients at **Q** and $\mathbf{C_b}$ also match:

      $\mathbf{\Delta(C_1, C_2)} = (0, 1, 30000)$, $\nabla f(\mathbf{C_2}) = (46000, 32000)$, $Score_{\mathbf{NN_1,C_1,C_2}} = 5.7$

      $\mathbf{\Delta(C_3, C_4)} = (1, -1, 20000)$, $\nabla f(\mathbf{C_4}) = (51000, 30000)$, $Score_{\mathbf{NN_2,C_3,C_4}} = 5.3$

      for some $\mathbf{C_1, C_2, C_3, C_4}$ in the original CB.

4. The target value for **Q** is found using Equation 5.7, that is, by taking the weighted average of n individual predictions, each made using Equation 5.3:

Prediction 1: $y^{\mathrm{NN}_1} - (y^{\mathrm{C}_1} - y^{\mathrm{C}_2}) = 370000 - 30000 = €340{,}000$

Prediction 2: $y^{\mathrm{NN}_2} - (y^{\mathrm{C}_3} - y^{\mathrm{C}_4}) = 350000 - 20000 = €330{,}000$

Overall Prediction: $\hat{y}^{\mathrm{Q}} = ((5.7 \times €340{,}000) + (5.3 \times €330{,}000)) / 11 = €335{,}182$

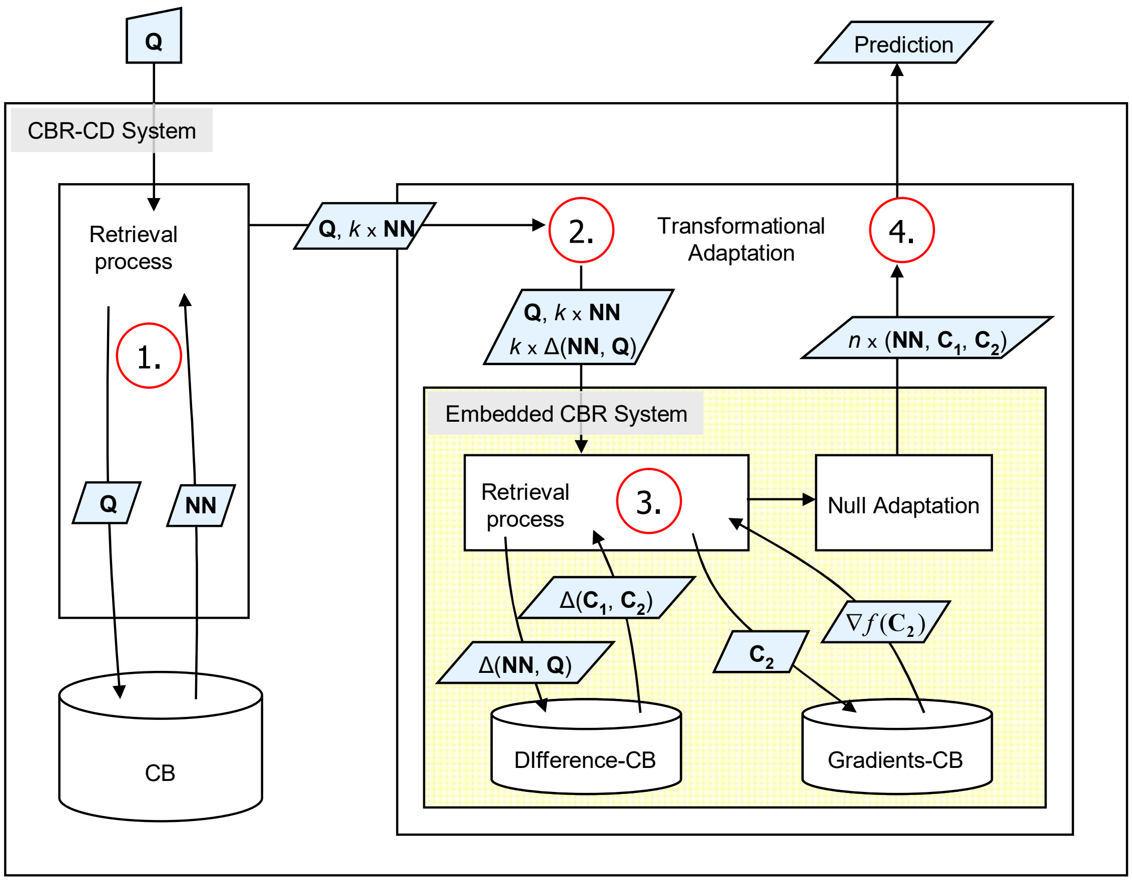


Figure 5.13: Making a prediction using CBR-CD

Most of the processing occurs in Step 3 where the optimal set of adaptation triples is found. This reflects the fact that the various improvements to CBR-CD have had the effect of making the adaptation process more complex. Adaptation is performed using an embedded CBR system that itself performs null adaptation; increased complexity in the adaptation process therefore equates to more complex retrieval in the Difference-CB. The basic problem solving methodology has remained unchanged, however: predictions are still made using pairs of stored cases to bridge the gap between the query and its neighbours. CBR-CD's contribution has been to ensure that the adaptation knowledge used to solve a query is actually applicable and of high quality.

5.3 Advantages and Limitations of CBR-CD

CBR-CD set out to solve new problems by utilizing the differences between stored cases. It also aimed to avoid the limitations of previous approaches, particularly those of CBR-AR (listed in Section 4.3.2). By achieving these goals, CBR-CD can be said to have the following advantages as a problem solving methodology:

Works correctly in arbitrary non-linear domains: CBR-CD can be used to perform regression in linear or non-linear problem domains. The gradient heuristic ensures that difference-cases used to bridge the gap between the query and its neighbours are applicable to the problem at hand. A number of different approaches to using gradients are possible, and the optimal one can be selected for each individual domain.

Offers robust performance in real-world domains: CBR-CD is designed to perform well in noisy, real-world problem domains. It uses the gradient heuristic to avoid noisy cases (i.e., cases that are distant from the local linear models constructed around them), and reduces error by averaging several individual prediction to make an overall prediction.

Knowledge-light approach to adaptation: CBR-CD does not require the addition of explicit expert domain knowledge. Adaptation knowledge is automatically generated from stored cases—from their differences and from their gradients.

Local and global domain knowledge is combined during problem solving: Each prediction uses local knowledge from the vicinity of the query (i.e., the gradient in that region of domain space and a list of the query's NNs) and global knowledge derived from the differences between cases elsewhere in domain space.

CBR-based approach to adaptation: The adaptation process uses an embedded CBR system to store and retrieve difference-cases. Adaptation therefore integrates neatly into the main CBR system, with the same case representation and retrieval mechanisms used for both.

Predictions can be accompanied by good explanations: Users can be provided with an intelligible explanation for each prediction, allowing them to judge the quality of the result for themselves. All predictions are made using the solutions to actual stored cases. Given a query **Q**, the following explanation can accompany the predicted solution:

> Case **NN** is very similar to **Q** and has solution *s*. The differences between the two cases are (d_1, d_2, …). A similar pair of cases, $\mathbf{C_1}$ and $\mathbf{C_2}$, have (almost) the

same differences between them. The difference between their solutions is d_s, therefore the predicted solution for **Q** is ($s + d_s$).

Where predictions from two or more adaptation triples are averaged to produce an overall prediction, each constituent can be explained in this way. The problem solving process is therefore fully transparent to the user, with complete visibility into the actual cases involved and the way in which they are used.

No problem solving methodology is without its limitations, and CBR-CD is no exception:

Relatively complex and computationally expensive: CBR-CD systems require a training phase during which the differences between stored cases are calculated and the gradient at each case is estimated. Solving a query involves calculating the gradient in its region of domain space and conducting a computationally expensive search for the optimal adaptation triple. Both construction and operation of a CBR-CD system are therefore more complex and computationally expensive than simpler approaches such as CBR-B. (But note that the complexity of the search algorithm is not visible to the user—each prediction and its associated explanation are entirely based on a set of concrete cases.)

Gradient heuristic relies on presence of at least one numeric attribute: Although CBR-CD can operate in domains with no numeric problem attributes, the value of the gradient heuristic declines as the number of numeric attributes decreases. This will have a deleterious effect on system performance.

5.4 Possible Extensions to CBR-CD

This section describes two extensions to CBR-CD that appear to offer the potential for improved predictive accuracy but fail to deliver this in practice. Despite their disappointing performance, it is instructive to examine them briefly and to consider the reasons for their failure.

5.4.1 Constructing adaptation paths from several difference-cases

The first potential enhancement to CBR-CD is to use more than one difference-case to solve a query. The principle behind this approach is shown in Figure 5.14, which depicts a query and its NN from the example housing domain. The Difference-CB does not contain a difference-case to bridge the gap between the two cases. However, a *pair* of difference-cases can be found that each cover part of the gap and together cover all of it. A set of difference-cases that combine to cover the difference between the query and an NN will be referred to as an *adaptation path* from **NN** to **Q**. Two such paths are shown in the figure: {$\mathbf{\Delta(C_1, C_2)}$, $\mathbf{\Delta(C_3, C_4)}$} and {$\mathbf{\Delta(C_5, C_6)}$, $\mathbf{\Delta(C_7, C_8)}$} for some $\mathbf{C_1}\ldots\mathbf{C_8}$ in the original CB.

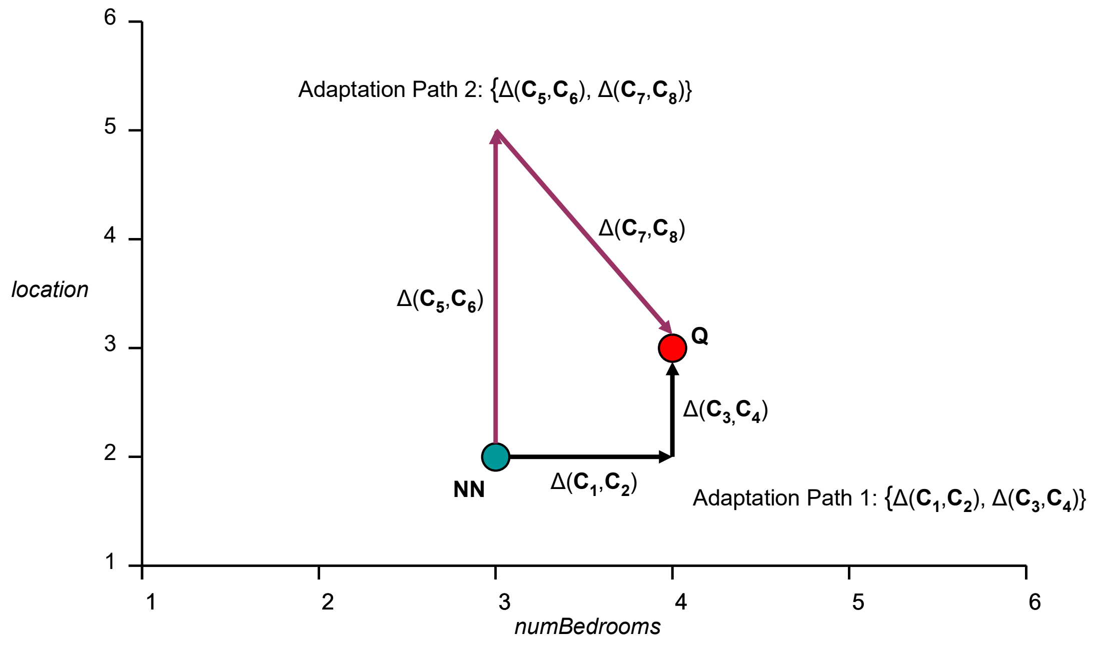

Adaptation Path 2: {Δ(C5,C6), Δ(C7,C8)}
Δ(C7,C8)
Δ(C5,C6)
location
Q
Δ(C3,C4)
NN
Δ(C1,C2)
Adaptation Path 1: {Δ(C1,C2), Δ(C3,C4)}
numBedrooms
1
2
3
4
5
6

Figure 5.14: Using adaptation paths to bridge the gap between the query and an NN

Predicting the target value for a query involves extending Equation 5.3 to accommodate the extra cases used:

$$\hat{y}^{\mathbf{Q}} = y^{\mathbf{NN}} - \sum_{i,j} (y^{\mathbf{C}_i} - y^{\mathbf{C}_j})$$

Note that an adaptation path is not limited to two difference-cases but may comprise any number (including just one).

The quality of an adaptation path may be judged by how completely it bridges the gap between the query and its NN. As Adaptation Path 2 in Figure 5.14 shows, an adaptation path may actually move *further away* from the query before moving back towards it. This suggests that an exhaustive search of the Difference-CB may be necessary to find the best adaptation path. Unfortunately, this approach is not practical for computational reasons. A CB of 1000 cases, for example, would yield a Difference-CB of 10000 cases; finding the best path of length 2 would then involve checking an unreasonable 10000^2 (= 10^{10}) possible paths.

A more practical search algorithm might begin by setting a maximum length for adaptation paths (e.g., 2). It might then find the best *n* adaptation triples as before, and use them to seed *n* adaptation paths with a single difference-case. A greedy search could then be performed whereby a difference-case is only appended to an adaptation path if it reduces the remaining distance to the query by at least 50%. This strategy was implemented and tested on the datasets used for the experimental evaluation in Chapter 7. A significant improvement in performance would be needed to justify the extra complexity involved. As it turned out, predictive results for all test sets were actually worse than those obtained using a single difference-case.

Using adaptation paths composed of several difference-cases to bridge the gap between the query and an NN is clearly analogous to the idea of chaining together adaptation rules in CBR-AR. Both are conceptually sound but suffer from a serious problem when applied to real-world datasets: solution errors accumulate with each additional difference-case used, so that the benefit of bridging the gap between the query and an NN more precisely is more than offset by increased error in the solution. As was the case for CBR-AR, it is counterproductive to try to be too precise when reasoning from noisy cases—better to accept that each prediction is just an estimate and try to minimize the overall statistical error. Additive solutions such as those derived from adaptation paths will always increase error rates and uncertainty and are generally best avoided in real-world domains. (These problems are discussed in greater detail in Section 4.3.2 under the headings 'Additive solution errors when more than one adaptation rule is used' and 'Problem solving is too brittle'.)

5.4.2 Normalizing numeric difference vectors

The second potential extension to CBR-CD is specific to datasets with numeric attributes and is intended for use with Approaches 1 and 2 (recall that in Approach 1, gradients are not used at all, while in Approach 2, difference-cases and gradients are matched separately). The basic idea is to normalize difference-cases to the same length so that differences of scale are eliminated. This should increase the range of problems that each difference-case can be used to solve.

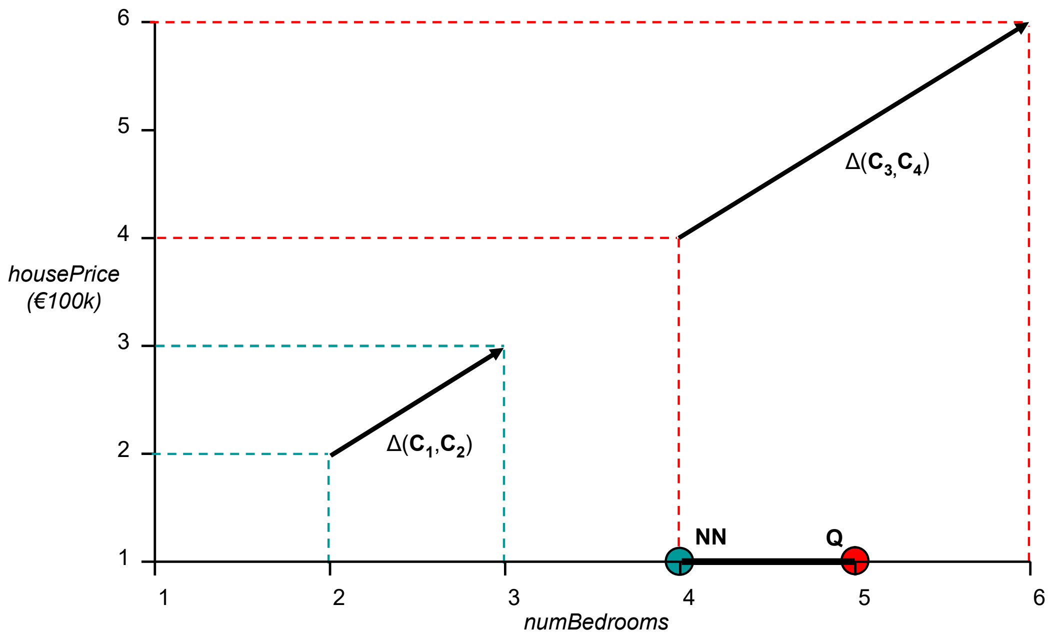


Figure 5.15: Equivalent difference-cases of different length

Figure 5.15 shows a simplified housing domain in which *housePrice* is a function of only one problem attribute, *numBedrooms*. A query and its NN are shown, and it can be seen that **Δ(NN, Q)** = 1. Two difference-cases are also shown; the first, **Δ($\mathbf{C_1}$, $\mathbf{C_2}$)**, suggests that increasing *numBedrooms* by 1 increases *housePrice* by €100,000, while the second, **Δ($\mathbf{C_3}$, $\mathbf{C_4}$)**, suggests that increasing *numBedrooms* by 2 increases *housePrice* by €200,000. It is clear that these two difference-cases make equivalent statements concerning the effect of *numBedrooms* on *housePrice*, and that the second is simply a scaled-up version of the first.

The range of problems that difference-cases can be used to solve can therefore be increased by normalizing the length of their problem parts to 1. Each attribute value is then a *direction cosine* [Stroud 2001] and each normalized difference-case indicates the effect that a particular unit vector in problem space has on the solution. To solve a query **Q**, vector **Δ(NN, Q)** is first normalized and then matched with a normalized difference-case **Δ($\mathbf{C_1}$, $\mathbf{C_2}$)**. This matching process can be performed by taking the dot product of **Δ(NN, Q)** and every possible (normalized) difference-case—the most similar difference-case will form the smallest angle with **Δ(NN, Q)** and hence produce the highest dot product (this is referred to as the *dot product similarity* [Plate 1994]). Solution difference $(y^{C_1} - y^{C_2})$ is then scaled up or down to match the original length of **Δ(NN, Q)**. In Figure 5.15, for example, **Δ($\mathbf{C_3}$, $\mathbf{C_4}$)** would be normalized from (2, 200000) to (1, 100000), and **Δ($\mathbf{C_1}$, $\mathbf{C_2}$)** and **Δ($\mathbf{C_3}$, $\mathbf{C_4}$)** would then be equally applicable for solving **Q**. If **Δ(NN, Q)** = 2 or 3 instead of 1, the predicted changes in *housePrice* $(y^{\mathbf{C}_1} - y^{\mathbf{C}_2})$ and $(y^{\mathbf{C}_3} - y^{\mathbf{C}_4})$ would be doubled or tripled to scale them to the appropriate level.

Solving problems with normalized difference-cases increases the complexity of the algorithm and reduces the usefulness of explanations that accompany predictions. The technique is also restricted to numeric datasets where difference-cases can be reduced to unit vectors, and cannot be used with Approaches 3 and 4 where gradients and difference-cases are considered together. As with adaptation paths, the improvement in performance needed to justify these limitations did not materialize in practice—predictive accuracy was roughly equivalent to that of non-normalized difference-cases for Approaches 1 and 2. This is probably due to the fact that the advantage of increasing the applicability of difference-cases is offset by the loss of information inherent in the normalization process. The effect that a small difference between two cases has on the solution does not necessarily scale up to larger differences, and the effect of large differences does not necessarily scale down. Best results are likely to be obtained when the cases $\mathbf{C}_1$ and $\mathbf{C}_2$ used to provide a solution are genuinely similar to the query and its NN.

5.5 Summary

This chapter introduced a new CBR-based regression algorithm called CBR-CD. This algorithm solves a query case by calculating the differences between it and an NN, then estimating the effect of these differences on the solution by looking at two stored cases with similar differences between them. Care must be taken to ensure that the two stored cases are of high quality and are actually applicable to the query at hand. To this end, a *gradient heuristic* was introduced to help guide the search for suitable cases. The gradient of domain space in the area around a case can be estimated by constructing a local linear model using LWLR. Integrating gradients into the search for problem-solving cases improves the relevance of retrieved cases with a consequent improvement in prediction accuracy. (This claim is verified in the experimental evaluation of Chapter 7.)

Issues of robustness were also addressed to ensure that CBR-CD performs well on noisy, real-world datasets. The algorithm incorporates two measures for this purpose: cases lying close to the local mean of the target function are preferred, and a weighted average of several individual predictions is taken to reduce error variance. The algorithm was also extended to deal with datasets possessing non-numeric problem attributes. Overall, CBR-CD was shown to possess several advantages: it is designed to perform robustly in arbitrary non-linear domains, it is CBR-based and knowledge-light, and each prediction can be accompanied by a useful explanation. Its principal downside is that it is relatively complex and computationally expensive.

Two possible extensions to CBR-CD were examined. The first looked at the possibility of using several pairs of stored cases to bridge the gap between the query and its NN. A second involved normalizing the differences between pairs of cases to increase their applicability. Both were found to increase complexity and decrease transparency for the user without offering improved performance in return.

Chapter 6

Implementation of CBR-CD

In previous chapters, we examined theoretical aspects of a number of regression algorithms. Standard approaches such as *k*-NN and LWLR were presented in Chapter 2. CBR-based algorithms CBR-B and CBR-AR were described in Chapter 4, together with new algorithm CBR-D. An analysis of the shortcomings of CBR-AR led to a set of requirements for a second new algorithm, CBR-CD, which was then described in Chapter 5. Advantages and disadvantages have been proffered for each one of these. But while some advantages may be more important in some operating situations than others (e.g., efficiency and plausibility), one measure of a regression algorithm that is always desirable is a high level of accuracy. CBR-D counts among its disadvantages the fact that it is a little more complex and computationally expensive than CBR-B, and CBR-CD is in a similar situation vis-à-vis CBR-AR. To find out whether they compensate for this with improved predictive performance, it is necessary to implement and test CBR-D and CBR-CD and compare their results with those of other algorithms.

This chapter describes the implementation of a general purpose CBR system. Five regression algorithms are supported within it: linear regression, LWLR, CBR-B, CBR-D and CBR-CD. Section 6.1 describes the overall architecture of the system, while Section 6.2 focuses on the implementation of CBR-CD in more detail. Chapter 7 uses this implementation to test and assess the relative performance of the various regression algorithms.

6.1 Architecture of CBR System

A general purpose CBR framework was implemented within an object-oriented structure using the Python programming language. Python is an ideal language for prototyping machine learning systems [Norvig 2000]. It is extremely easy to use, read and modify—completed code is often as concise and easy to read as the equivalent pseudocode. It has built-in support for list and dictionary data types, which together with dynamic typing, make it very easy to assemble and manipulate datasets with attributes of any type. Because Python is an interpreted language, code can be tested interactively during development without the need for recompilation. The downside of this is that Python usually runs slower than alternatives such as C/C++. This limits its usefulness for computationally intensive applications, but does not present a problem for test systems such as that described here.

The class structure of the system is very simple; all classes are shown in Figure 6.1 together with their public operations. The main responsibilities of each class are as follows:

Dataset: The Dataset base class implements basic functionality for working with datasets. It reads and writes files in the CSV ('comma-separated values') and ARFF ('attribute-relation file format' used for datasets in Weka [Witten and Frank 2000]) formats, as well as serialized (or *pickled*) Python objects. It also supports a standard range of operations for manipulating datasets, including sampling and filtering instances according to user-defined criteria.

Cbr: The Cbr class is derived from Dataset. It provides operations for constructing and operating a CBR system of the type described in Chapter 3. It is capable of retrieving a set of NNs for a given query and using them to make a prediction; for simplicity, retrieval is performed with a simple sequential search through the set of stored cases. Built-in support is provided for classification and regression tasks, and the system can be extended to handle more complex tasks if necessary.

Numeric and nominal attribute types are automatically catered for, and support for others can easily be added. Operations are provided to define difference-similarity graphs (by a set of points) and similarity tables (by a two-dimensional array of values) (see [Stahl and Gabel 2003] for an introduction to these local similarity functions). An arbitrary comparator for any attribute can also be defined at runtime by passing a Python function to Cbr.setComparator as a string—this is dynamically compiled and used for subsequent similarity calculations.

To assist with the task of tuning a CBR system during operation, methods are provided to find suitable values for CBR parameters such as *k* (i.e., number of NNs to use for a prediction) and attribute weights, and for performing cross-validation, leave-one-out, and standard training set/test set testing. From a regression perspective, class Cbr implements the CBR-B and CBR-D algorithms described in Chapter 4.

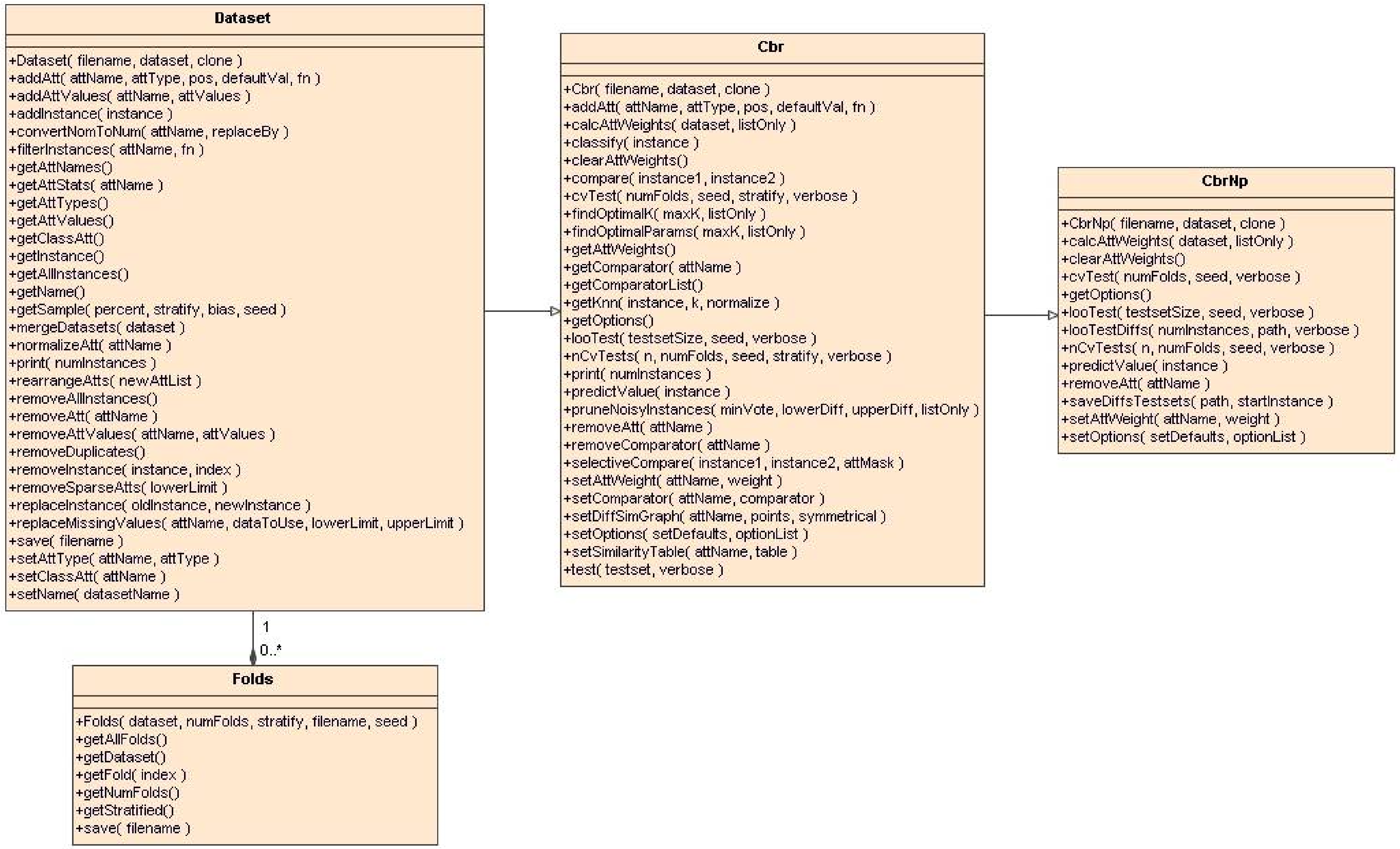

Dataset
+Dataset(filename, dataset, clone)
+addAtt(attName, attType, pos, defaultVal, fn)
+addAttValues(attName, attValues)
+addInstance(instance)
+convertNomToNum(attName, replaceBy)
+filterInstances(attName, fn)
+getAttNames()
+getAttStats(attName)
+getAttTypes()
+getAttValues()
+getClassAtt()
+getInstance()
+getAllInstances()
+getName()
+getSample(percent, stratify, bias, seed)
+mergeDatasets(dataset)
+normalizeAtt(attName)
+print(numInstances)
+rearrangeAtts(newAttList)
+removeAllInstances()
+removeAtt(attName)
+removeAttValues(attName, attValues)
+removeDuplicates()
+removeInstance(instance, index)
+removeSparseAtts(lowerLimit)
+replaceInstance(oldInstance, newInstance)
+replaceMissingValues(attName, dataToUse, lowerLimit, upperLimit)
+save(filename)
+setAttType(attName, attType)
+setClassAtt(attName)
+setName(datasetName)
1
0..*
Folds
+Folds(dataset, numFolds, stratify, filename, seed)
+getAllFolds()
+getDataset()
+getFold(index)
+getNumFolds()
+getStratified()
+save(filename)
Cbr
+Cbr(filename, dataset, clone)
+addAtt(attName, attType, pos, defaultVal, fn)
+calcAttWeights(dataset, listOnly)
+classify(instance)
+clearAttWeights()
+compare(instance1, instance2)
+cvTest(numFolds, seed, stratify, verbose)
+findOptimalK(maxK, listOnly)
+findOptimalParams(maxK, listOnly)
+getAttWeights()
+getComparator(attName)
+getComparatorList()
+getKnn(instance, k, normalize)
+getOptions()
+looTest(testsetSize, seed, verbose)
+nCvTests(n, numFolds, seed, stratify, verbose)
+print(numInstances)
+predictValue(instance)
+pruneNoisyInstances(minVote, lowerDiff, upperDiff, listOnly)
+removeAtt(attName)
+removeComparator(attName)
+selectiveCompare(instance1, instance2, attMask)
+setAttWeight(attName, weight)
+setComparator(attName, comparator)
+setDiffSimGraph(attName, points, symmetrical)
+setOptions(setDefaults, optionList)
+setSimilarityTable(attName, table)
+test(testset, verbose)
CbrNp
+CbrNp(filename, dataset, clone)
+calcAttWeights(dataset, listOnly)
+clearAttWeights()
+cvTest(numFolds, seed, verbose)
+getOptions()
+looTest(testsetSize, seed, verbose)
+looTestDiffs(numInstances, path, verbose)
+nCvTests(n, numFolds, seed, verbose)
+predictValue(instance)
+removeAtt(attName)
+saveDiffsTestsets(path, startInstance)
+setAttWeight(attName, weight)
+setOptions(setDefaults, optionList)

Figure 6.1: Class diagram for CBR system

CbrNp: The CbrNp class is derived from Cbr and provides additional functionality for regression (the 'Np' in CbrNp stands for 'numeric prediction'). It adds support for two standard regression algorithms: linear regression (see Section 2.1) and LWLR (see Section 2.3). Linear regression is performed analytically using the numarray numerical Python package[*]. LWLR's local linear models are also derived analytically using function lm from the R statistical system [R Core Development Team 2005]. As well as these standard algorithms, CbrNp implements the CBR-CD algorithm described in Chapter 5, and provides functions for testing all three regression algorithms.

Folds: The Folds class divides a dataset's instances into a number of folds. These can then be used by Cbr and CbrNp objects for cross validation testing.

6.2 CBR-CD Implementation Notes

This section looks at the implementation of CBR-CD in greater detail. This algorithm can be divided into two parts:

1. A construction phase during which the Difference-CB is constructed and the gradient at each case is estimated.
2. An operation phase during which queries' numeric target values are predicted. For each query,
   i. The *k* most similar stored cases (i.e., NNs) are retrieved;
   ii. *n* adaptation triples are found using one of four possible problem solving approaches and their solutions are averaged to give a final prediction.

The behaviour of each of the two phases is governed by a number of potentially tuneable parameters that are described in the following sub-sections. Some parameters are assigned default values, some are set automatically, and some must be set by the user.

6.2.1 Parameters for the construction phase

This section discusses parameters that are relevant to the construction phase of CBR-CD. Most have already been examined in previous chapters. All of them are either set to default values or

[*] numarray is a product of the Space Telescope Science Institute, which is operated by AURA for NASA.

set automatically by the system. Since the user's involvement is not required during the training phase, setup of a new CBR-CD system is straightforward.

Number of cases in the Difference-CB: *Set to default value*

To construct the Difference-CB, each stored case is compared with a number of others. For a CB of size n, comparing each case with all others in the CB would result in a Difference-CB of size $n \times (n - 1)$. Storage requirements are reduced in CBR-CD by comparing each case with its 10 NNs so that the Difference-CB contains $(n \times 10)$ difference-cases. The choice of the number 10 is arbitrary.

Distance formula used for retrieval of a case's NNs: *Set to default*

The distance between pairs to cases is calculated using the standard formula shown in Equation 4.1. As discussed in Section 4.1.1, this formula can be tuned in various ways to improve performance in individual domains. CBR-CD does not attempt to optimize the distance formula for different datasets, but always uses the same values:

- A Gaussian weighting function (shown in Equation 4.2) is used without a smoothing parameter.
- Attribute weights that reflect the global relevance of individual problem attributes are not used. The reasons for this are discussed in Sections 3.2.1.3 and 5.2.1.5.

Number of cases used to estimate gradients in domain space: *Set automatically*

The gradient around a particular case is estimated by using LWLR to construct a local linear model. The proportion of the CB used for constructing linear models is chosen from the set {0.05, 0.1, 0.15, 0.2} to minimize cross-validation error for each dataset. The contribution of each case is weighted by distance using the Gaussian weighting function shown in Equation 4.2. Linear models constructed in this way not only provide gradients for the CBR-CD algorithm, but also provide LWLR predictions for experiments in Chapter 7.

6.2.2 Parameters for the operation phase

This section looks at parameters that govern the behaviour of CBR-CD in its operation phase. Again, it summarizes decisions that have been discussed in previous chapters. All but one of the parameters receive default values or are set automatically by the system during an initial training phase. One parameter that determines how gradients are used during problem solving must be set manually by the user. The primary reason for this is to allow all four possible approaches to be tested; it would of course have been possible to configure the system to automatically use the approach that gave the lowest cross-validation error.

Number of matching NNs to retrieve for a query (*k*): *Set to default value*

This is set to the arbitrary value 10. This is high enough to allow the algorithm some freedom in selecting NNs for adaptation triples, but low enough to exclude NNs that are unlikely to be useful.

Approach to gradients: *Set manually by the user*

Sections 5.2.1.1–5.2.1.4 describe four different approaches to using gradients during the search for adaptation triples. (Approach 1 does not use gradients at all, Approach 2 considers gradients and difference-cases separately, and Approaches 3 and 4 consider the two together.) As discussed in Section 5.2.1.5, the best-performing approach is likely to vary with the problem domain. In the experiments of Chapter 7, all four approaches are tested for each dataset.

Regardless of the approach used, finding suitable difference-cases involves a multi-objective search in which case triples receive an overall score that reflects their applicability. As discussed in Section 5.2.1.5, predictive performance might be improved by weighting the different objectives with parameter α_i. In the implementation of CBR-CD, this approach is not used and all objectives are assigned equal importance.

Number of matching difference-cases to retrieve for Δ(NN, Q): *Set to default value*

Difference-cases, gradients, nearest neighbours—all are considered together in the score formulae used to assess adaptation triples. To limit computational complexity, only the 10 closest-matching difference-cases are considered for each (**NN**, **Q**) pair. Since $k = 10$, this means that the algorithm will always assess a total of 100 adaptation triples for each query. The choice of the number 10 is arbitrary.

Number of adaptation triples to use for a prediction (*n*): *Set automatically*

The value of n from range 1–10 that yields the lowest cross-validation error is used for each dataset. So if $n = 5$, for example, a total of 100 adaptation triples will be considered and predictions from the highest-scoring 5 will be averaged to produce an overall prediction.

6.3 Summary

This chapter described the implementation of a simple CBR framework in the Python programming language. Constructed using an object-oriented architecture, this framework comprises four classes that provide all of the normal functionality expected in a CBR system. Implementations are also included for a number of regression algorithms that will be tested in Chapter 7: CBR-B, CBR-D, CBR-CD, linear regression, and LWLR.

As the most sophisticated of the regression algorithms described in this thesis, CBR-CD has a number of parameters that govern its behaviour. All but one of these receive default values or are set automatically during system construction and training. This simplifies the deployment of new CBR-CD-based applications.

Chapter 7

Experimental Evaluation

This thesis has introduced two new CBR-based regression algorithms. The first, CBR-D, predicts a query's target value by retrieving a diverse set of neighbouring cases and taking the weighted average of their solutions. The second, CBR-CD, is more substantial and attempts to solve new cases using the differences between stored cases. This chapter evaluates these algorithms by comparing their predictive performance with that of several standard algorithms.

7.1 Datasets Used for Experimental Evaluation

Five datasets from the UCI [Hettich et al. 1998] and StatLib* repositories were used for the experimental evaluation. They were chosen on the basis that they represent a reasonable cross-section of the sort of regression problems that are encountered in practice. They cover five different problem domains with target functions of varying degrees of linearity. They have varying numbers and types of problem attributes, numbers of instances, and levels of noise. As discussed in Section 2.4, different algorithms can be expected to perform better on some datasets than others. Evaluating the different regression algorithms on a number of contrasting datasets should therefore provide a good overall assessment of their capabilities.

The five datasets are summarized in Table 7.1.

* StatLib dataset archive is hosted by the Department of Statistics at Carnegie Mellon University, URL: http://lib.stat.cmu.edu/.

Table 7.1: Datasets used for experimental evaluation

Dataset	*Description*	*Problem Att.s (Num + Nom)*	*No. Instances (Train + Test)*	*Source*
Boston Housing	Predicting housing prices in suburbs of Boston	13 + 0	506	UCI
Tecator	Predicting the fat content of meat samples[*]	10 + 0	195	StatLib
Abalone	Predicting the age of abalone from physical measurements	7 + 1	4177 (3759 + 418)	UCI
CPU	Predicting CPU performance from PC characteristics	7 + 1	209	UCI
Servo	Predicting the rise time of a servomechanism	2 + 2	167	UCI

Two sets of factors influenced the choice of datasets, the first pertaining to the problem domains dealt with:

Contrasting problem domains. As seen in Table 7.1, the five datasets represent five very different problem domains ranging from housing to servomechanisms. Knowledge-light regression algorithms such as those tested in this chapter should be capable of adapting easily to different domains while maintaining a consistently high level of predictive accuracy.

Varying degrees of linearity. When discussing different problem domains, it is useful to assume the existence of an unknown *target function* that accurately maps problem attributes to solutions. The goal of regression algorithms is then to approximate this function as closely as possible, either globally (e.g., in a neural network) or locally (e.g., using *k*-NN). One measure of the complexity of a domain's target function is its degree of global linearity, that is, how accurately it can be approximated by a linear

[*] Tecator dataset was originally compiled by Hans Henrik Thodberg, Danish Meat Research Institute, Maglegaardsvej 2, Postboks 57, DK-4000 Roskilde, Denmark. Available from StatLib at http://lib.stat.cmu.edu/datasets/tecator. Dataset contains 100 numeric problem attributes (absorbency levels for different infrared frequencies). The most important 22 (found using principal component analysis) are also listed, and the first 10 of these are used in the experimental evaluation described here.

regression model. The more non-linear the target function, the more complex it can be considered to be.

A related measure of a target function's complexity is its degree of local linearity. If the solution changes smoothly with respect to problem attributes over local areas of domain space, it may be approximated well by local linear models in those regions. Such target functions are less complex than those that are highly non-linear even in local sub-sections of domain space.

The five datasets chosen for the experimental evaluation have different degrees of global and local linearity. This is shown in the Experimental Setup (Section 7.3.1), and also in the results obtained for global and local linear modelling in Experiment 2 (Section 7.4.2). Varying levels of linearity among target functions tests the versatility of the different algorithms in coping with problem domains with different characteristics.

The second set of factors influencing the choice of datasets relate to the characteristics of the datasets themselves:

No missing values. Linear regression and LWLR do not work in the presence of missing values. CBR-CD uses LWLR as a heuristic during problem solving, and so it doesn't work with missing values either. These algorithms can only be used with incomplete datasets following *imputation* of missing data (i.e., filling in missing data with plausible values). Multiple imputation (MI) [Rubin 1996] is one of a number of standard algorithms used for this purpose.

Varying number and types of problem attributes. The number of problem attributes among the datasets varies from 4 to 14. All datasets contain at least one numeric problem attribute; this is necessary to allow linear regression, LWLR, and the full CBR-CD algorithm to be used. Three of the five datasets also contain one or more nominal attributes. This allows CBR-CD's support for non-numeric attributes to be tested.

Varying numbers of instances. The chosen datasets vary in size from 167 up to 4177 instances to test how algorithms respond to different levels of case coverage. Note that the implementation of the various regression algorithms (as described in Chapter 6) is not optimized for computational efficiency; it is intended as a prototype rather than an operational system and is not designed to support very large datasets.

Varying levels of noise. All real-world datasets are subject to noise, and all real-world regression algorithms must therefore deal with it. Noise may be caused by inaccurate measurement (or recording) of problem attributes or solutions. Alternatively, it may be caused by problem attributes that have some affect on the solution but that are not included in the dataset. In medical domains, for example, full details of a patient's physiological characteristics and medical history cannot possibly be included in any dataset. There is therefore an inevitable degree of uncertainty involved in mapping problem attributes to solutions, and this is reflected in a certain level of 'noise' within each dataset.

All five datasets represent real-world problem domains and possess moderate to high levels of noise. The Abalone dataset is particularly noisy, reflecting the fact that it is difficult to predict the age of an adult mollusc on the basis of a few physical measurements.

Freely available and widely used. All datasets are publicly available, and so all results can be benchmarked against those obtained with alternative implementations or algorithms if necessary.

7.2 Overview of Experiments

Two sets of experiments are presented here. The first set focuses on the algorithm that is the primary contribution of this thesis, CBR-CD. It comprises an ablation study in which different variants of the algorithm are compared to show how improvements in performance are related to different parts of the algorithm. The second set of experiments compares the performance of several regression techniques, paying particular attention to the relative performance of the two new algorithms, CBR-D and CBR-CD.

The following points are relevant to the conduct of both sets of experiments:

- All numeric solutions and problem attributes are normalized to the range 0–1 for all datasets. This makes them easier to work with and allows more meaningful comparison of results for different algorithms and datasets.
- Leave-one-out cross validation is used for all testing, with the exception of the Abalone dataset where a 90%/10% training/test split is made.

7.3 Experiment 1—Comparing Different Variants of CBR-CD

This experiment involves an ablation study of the CBR-CD algorithm. In an ablation study, different parts of an algorithm are systematically removed to see which result in a substantial reduction in performance. This makes it possible to identify which aspects of the algorithm are responsible for its predictive power, and which can potentially be dispensed with without adversely affecting it. In this experiment, the ablation study assesses the merit of the different ways to use gradients and to improve robustness within CBR-CD.

7.3.1 Experiment 1: Experimental setup

The behaviour of the CBR-CD algorithm is governed by a number of parameters. As discussed in Section 6.2, two of these parameters are automatically set to values that minimize cross-validation error over each dataset. They are *n* (i.e., the number of adaptation triples whose predictions are averaged to produce a final prediction), and the proportion of each dataset used to construct local linear models. The values assigned to these parameters for each dataset are shown in Table 7.2.

Table 7.2: Values of parameters set automatically in CBR-CD

Dataset	*n*	*% of dataset used for local linear models*[*]
Boston Housing	5	10%
Tecator	5	20%
Abalone	10	10%
CPU	5	10%
Servo	2	5%

Note that the well-populated but noisy Abalone dataset received the maximum value of 10 for parameter *n*—averaging a large number of predictions reduced error variance. On the other hand, *n* is set to 2 for the highly non-linear Servo dataset—the benefits of averaging predictions from lower-scoring adaptation triples was more than offset by their reduced applicability. A high degree of non-linearity also explains the relatively low number of cases used to construct local linear models for Servo.

[*] These values are also used for the LWLR algorithm in Experiment 2.

Six different variants of CBR-CD are compared in this experiment. Each is defined by a particular approach to using gradients (Approach 1–4) and by whether or not LWLR and weighted averaging are used to increase robustness:

1. **Approach 1.** This is the most basic version of CBR-CD (see Section 5.2.1.1). A single adaptation triple is used for each prediction, and difference-case **Δ(C$_1$, C$_2$)** is chosen from any part of domain space to provide the closest match to **Δ**(**NN**, **Q**).
2. **Approach 1 with NN Heuristic.** This is similar to the basic algorithm except that cases from close to the query are preferred when choosing difference-case **Δ(C$_1$, C$_2$)**. This 'nearest-neighbour heuristic' has been suggested in prior research and serves as a useful baseline for comparison (see Section 4.3.3).
3. **Approach 1 + Increased Robustness.** Predictions from several adaptation triples are averaged for each prediction (see Section 5.2.3). The number used for each dataset is listed in Table 7.2. In a second variation of this experiment, LWLR is also used to avoid noisy cases (see Section 5.2.2).
4. **Approach 2 + Increased Robustness.** As 3, but with a preference for difference-cases from areas of domain space with a gradient similar to that around the query (see Section 5.2.1.2).
5. **Approach 3 + Increased Robustness.** As 3, but with gradients and difference-cases combined in a *vector* and considered together during the search for adaptation triples (see Section 5.2.1.3).
6. **Approach 4 + Increased Robustness.** As 3, but with gradients and difference-cases combined in a *scalar* and considered together during the search for adaptation triples (see Section 5.2.1.4).

7.3.2 Experiment 1: Results

Each variant of CBR-CD was tested on each of the five datasets. Results were then aggregated to show the overall performance of each variant relative to the basic algorithm (variant 1). Results are shown in Figures 7.1–7.6, with the six variants shown left to right in each chart.

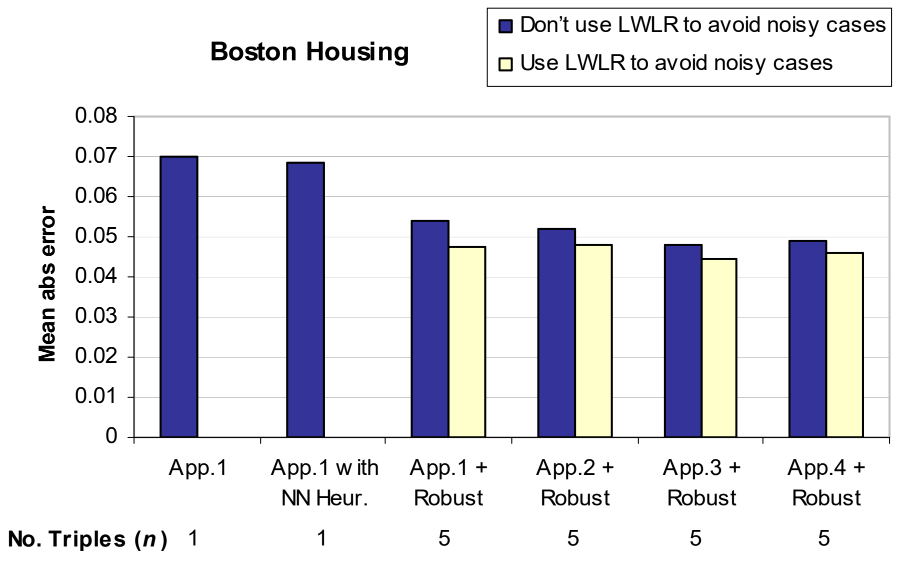


Figure 7.1: Experiment 1—Results for Boston Housing dataset

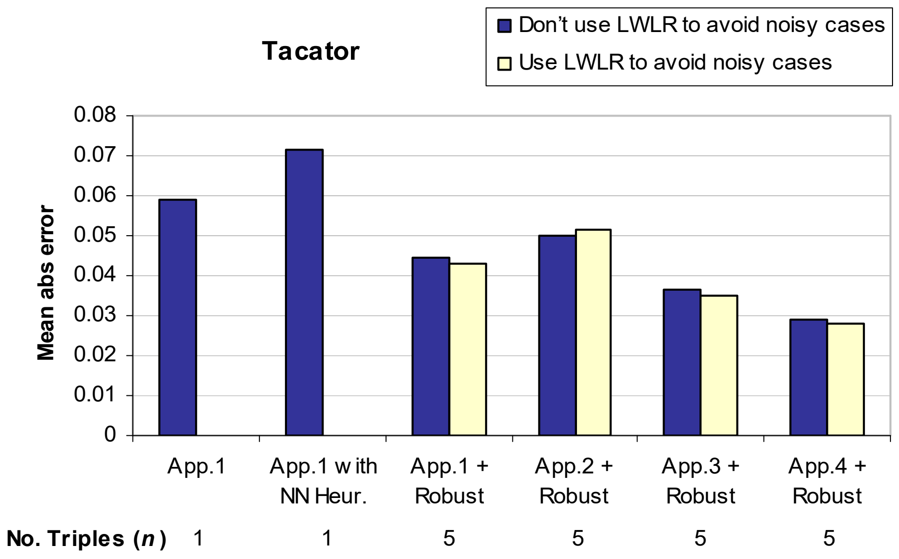


Figure 7.2: Experiment 1—Results for Tecator dataset

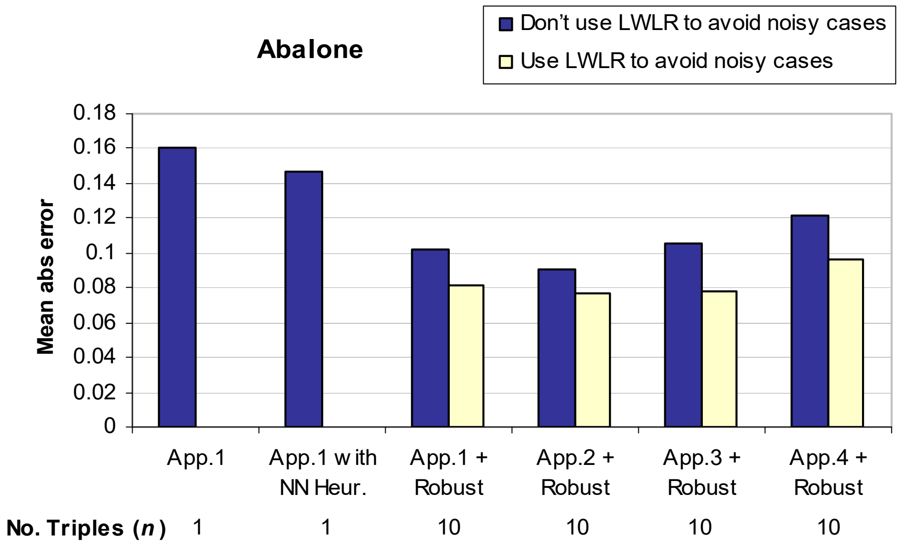


Figure 7.3: Experiment 1—Results for Abalone dataset

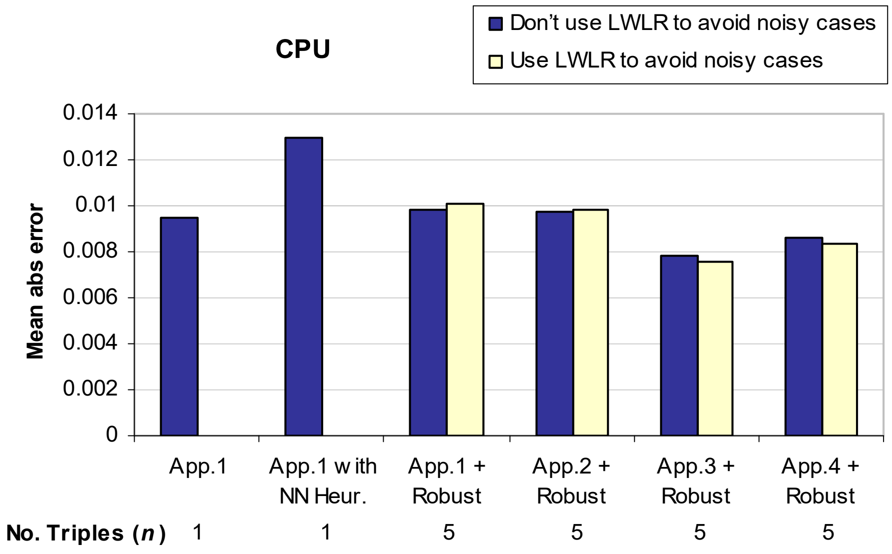


Figure 7.4: Experiment 1—Results for CPU dataset

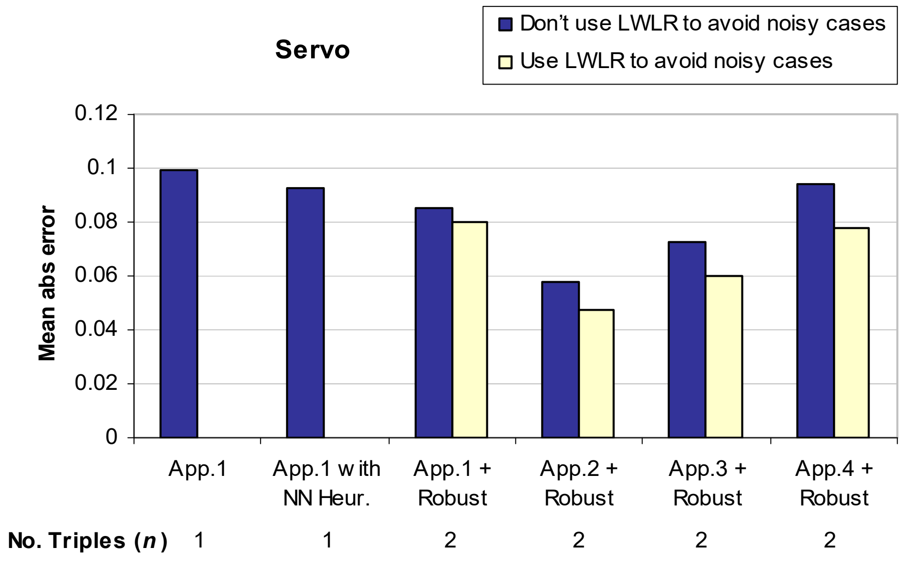


Figure 7.5: Experiment 1—Results for Servo dataset

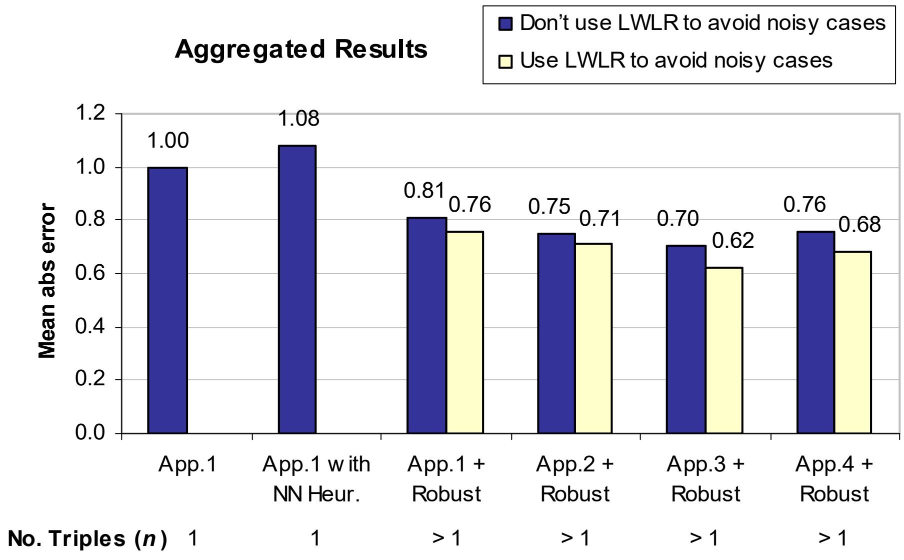


Figure 7.6: Experiment 1—Aggregated Results

7.3.3 Experiment 1: Conclusions

The following conclusions may be drawn from Experiment 1:

1. The NN Heuristic did not improve results over the basic CBR-CD algorithm. As discussed in Section 4.3.3, the benefit of choosing difference-cases from close to the query was more than offset by the reduction in search space for these cases.

2. All enhancements to the basic algorithm described in Section 5.2 improved results over the basic algorithm:
   - Using gradients to guide the search for difference-cases (i.e., using Approaches 2–4 instead of Approach 1);
   - Increasing robustness by using LWLR to avoid noisy cases;
   - Increasing robustness by averaging predictions from several adaptation triples to produce a final prediction.
3. The best approach to using gradients depended on the dataset concerned: Approaches 2, 3 and 4 were best twice, twice and once for the five datasets. Overall, the best-performing was Approach 3 (described in Section 5.2.1.3). It is not surprising that different approaches are more appropriate for some datasets than for others; as discussed in Sections 2.4 and 5.2.1.5, no algorithm is universally better than all others in all circumstances. It is for this reason that CBR-CD was designed with the flexibility to vary its approach to suit the particular characteristics of each individual dataset.

7.4 Experiment 2—Comparing Different Regression Algorithms

The second set of experiments assesses the performance of CBR-D and CBR-CD relative to a number of standard regression techniques. As discussed at the beginning of Chapter 6, CBR-CD is more computationally intensive than some alternative approaches and needs to justify this with good predictive performance. Similarly, CBR-D is a little more elaborate than CBR-B and should compensate for this with improved accuracy.

7.4.1 Experiment 2: Experimental setup

The following regression algorithms are tested in Experiment 2:

CBR-B, *k* = 1: Each query is solved by simply copying the solution from its nearest neighbour. This is the simplest instance-based algorithm, and serves as a useful baseline for comparing predictive performance.

CBR-B: Each query is solved by taking the weighted average of solutions from its *k*-NNs (see Section 4.1). The value of *k* that minimizes cross-validation error is used for each dataset. The contribution from each neighbour is weighted by distance from the query using a Gaussian weighting function (shown in Equation 4.2).

Linear Regression: A single linear model is constructed over the entire dataset and used to make predictions for new cases (see Section 2.1). Linear regression only works with numeric attributes, and so the algorithm was run twice for the three datasets with nominal attributes: once with nominal attributes removed, and once with each nominal attribute replaced by an ordered set of synthetic binary attributes [Witten and Frank 2000, p. 204]. The better of the two results is shown in the Results section below.

LWLR: Local linear models fitted to each query's NNs are used to make predictions (see Section 2.3). The number of cases used in their construction varies with the dataset, and is chosen to minimize cross-validation error. Gaussian distance weighting reduces the influence of cases more distant from the query. LWLR does not support non-numeric attributes, and so nominal attributes are treated as for linear regression.

M5′ (M5P): This algorithm produces model trees with linear regression equations at the leaves (see Section 2.5.1). The Weka implementation is used with default parameter settings [Witten and Frank 2000, pp. 202–208].

CBR-D: Queries are solved by taking the weighted average of solutions from *k* diverse cases chosen from among a larger set of NNs (see Section 4.2). As with CBR-B, the value of *k* is set by minimizing cross-validation error over each dataset and Gaussian distance weighting is used.

CBR-CD: Queries are solved using case differences as described in Chapter 5. All variants of CBR-CD in Experiment 2 use the two robustness measures discussed in Section 5.2 (i.e., LWLR is used to help avoid noisy cases, and the weighted average of predictions from several adaptation triples is taken). For each dataset, three results are shown for CBR-CD that reflect different approaches to choosing adaptation triples:

1. **Diffs—No Grad.** Gradients are not used to guide the search for adaptation triples (i.e., Approach 1 is used).
2. **Diffs—Mean Grad.** The three gradient-based approaches to selecting adaptation triples (i.e., Approaches 2–4) are used and their results averaged.
3. **Diffs—Best Grad.** The best performing of the three gradient-based approaches to choosing adaptation triples is used. (The best approach for any particular dataset can be found by performing cross-validation testing on a subset of cases.)

Four of these algorithms, CBR-B, LWLR, CBR-D and CBR-CD have parameters that are automatically set during an initial training phase. Parameter values for LWLR and CBR-CD are shown in Table 7.2. The values of *k* used by CBR-B and CBR-D are shown in Table 7.3.

Table 7.3: Values of *k* used by CBR-B and CBR-D

Dataset	*k (CBR-B)*	*k (CBR-D)*
Boston Housing	3	4
Tecator	3	5
Abalone	10	10
CPU	6	5
Servo	1	1

These parameter values reflect the nature of the different datasets. Abalone is well-populated but very noisy; best results are therefore obtained by averaging solutions from a high number of NNs. Servo is extremely non-linear, and so the usefulness of neighbouring cases declines rapidly with distance from the query—hence the low values for *k*. The other datasets lie between these two extremes.

7.4.2 Experiment 2: Results

Each regression algorithm was tested on each of the five datasets. Results for all datasets were then aggregated to show the overall performance of each algorithm relative to 1-NN. Results are shown in Figures 7.7–7.12.

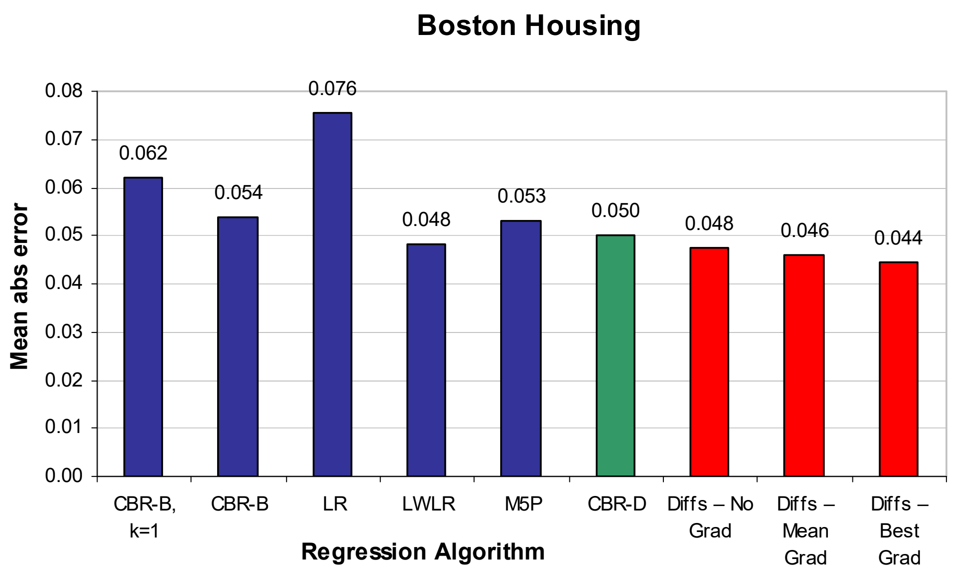


Figure 7.7: Experiment 2—Results for Boston Housing dataset

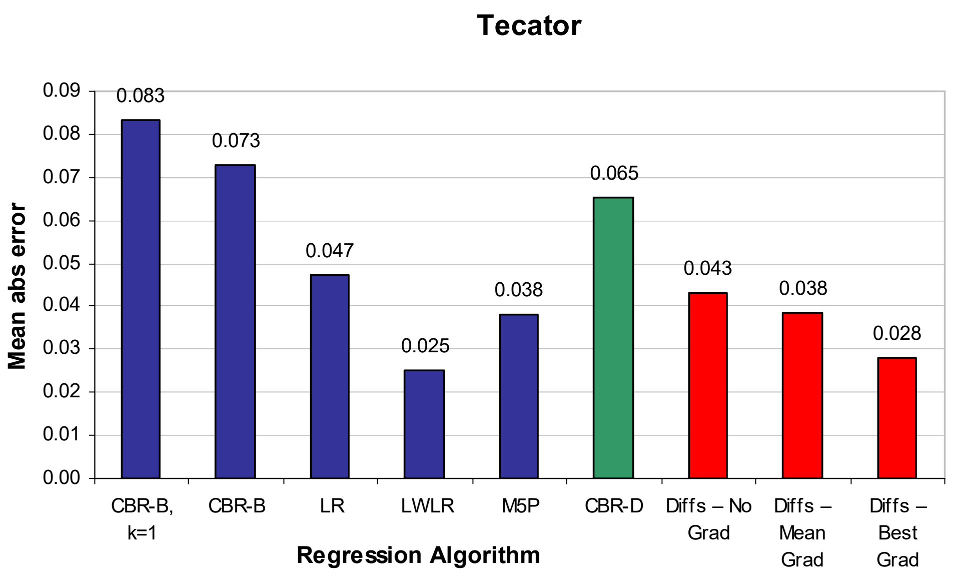


Figure 7.8: Experiment 2—Results for Tecator dataset

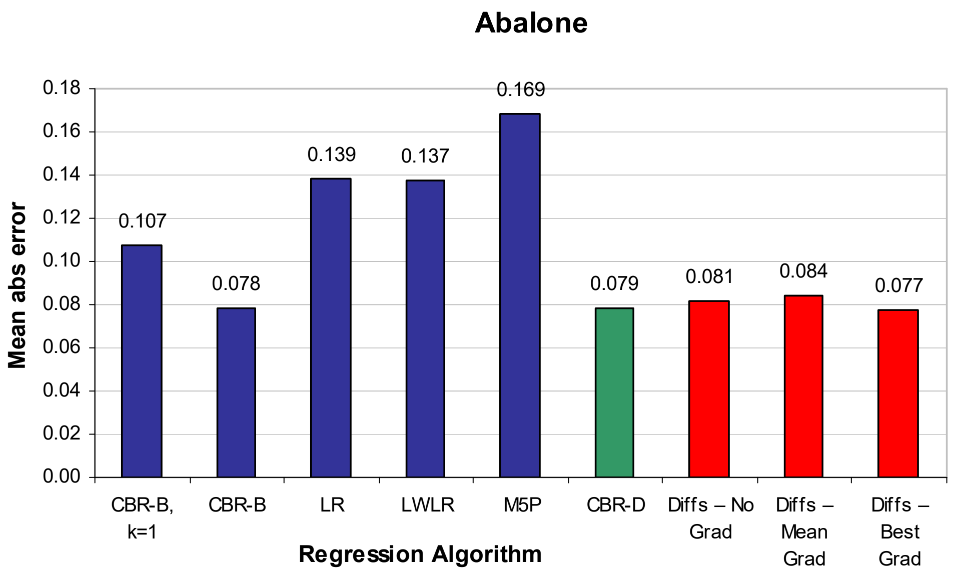


Figure 7.9: Experiment 2—Results for Abalone dataset

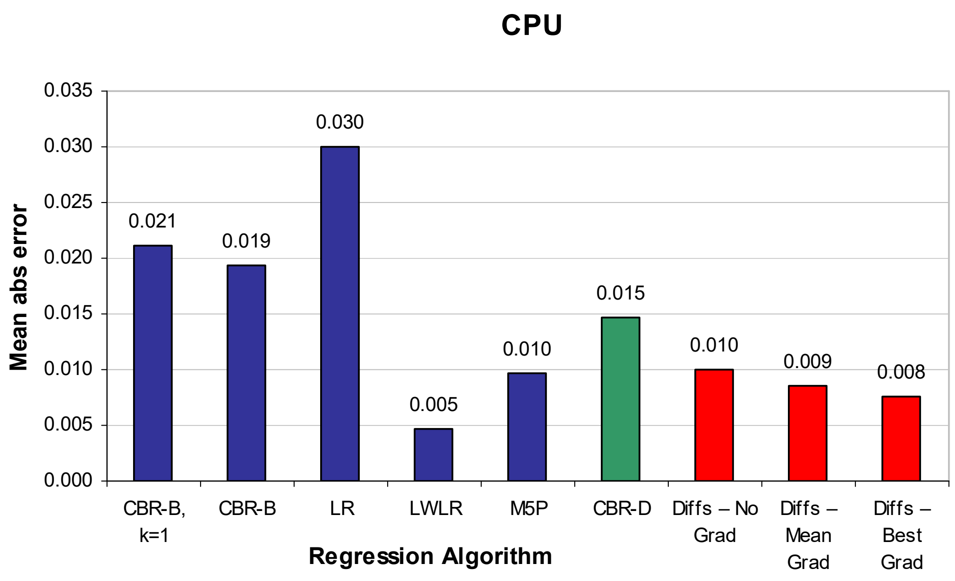


Figure 7.10: Experiment 2—Results for CPU dataset

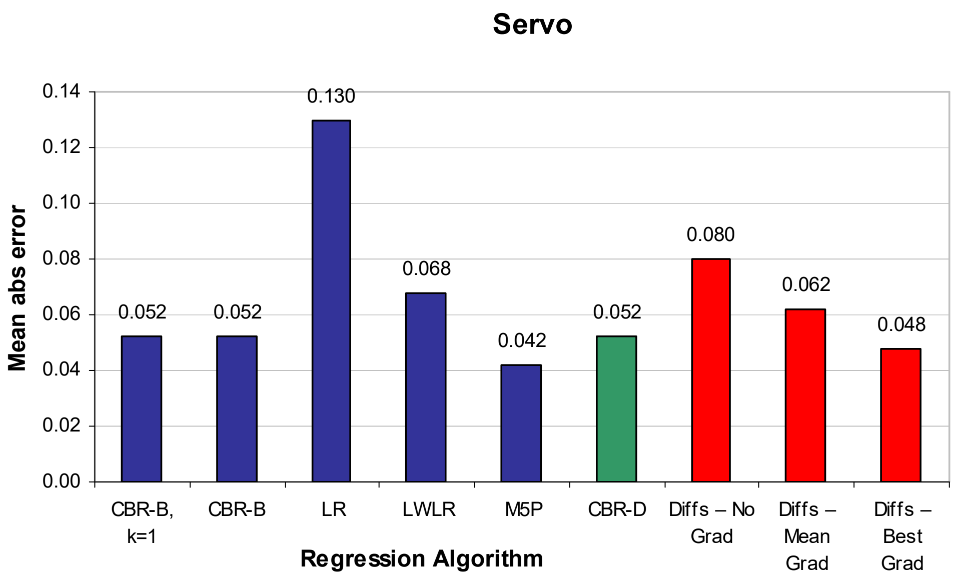


Figure 7.11: Experiment 2—Results for Servo dataset

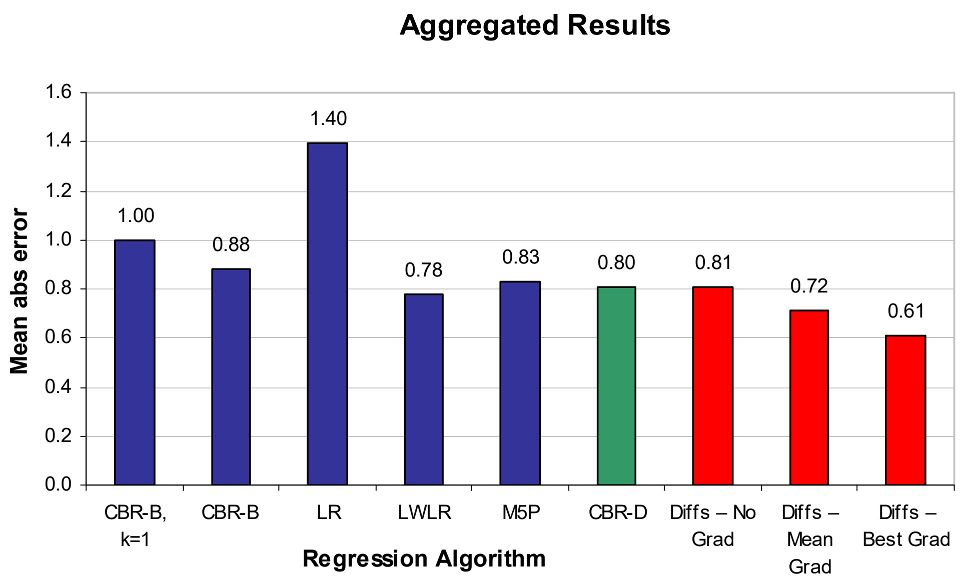


Figure 7.12: Experiment 2—Aggregated Results

7.4.3 Experiment 2: Conclusions

The following conclusions may be drawn from Experiment 2:

1. CBR-D significantly outperformed CBR-B on two of the five datasets tested, and performed at a statistically equivalent level on the other three. Note that the high level of non-linearity in the Servo dataset meant that using a single NN gave best—and equal—results for both CBR-B and CBR-D. These results are summarized in Table 7.4. (The *Significant?* column indicates whether one outperformed the other to a statistically significant degree on a one-tailed t-Test with $P < 0.01$.)

Table 7.4: CBR-D vs. CBR-B

Dataset	*Best*	*Significant?*
Boston Housing	CBR-D	Yes
Tecator	CBR-D	No
Abalone	CBR-B	No
CPU	CBR-D	Yes
Servo	=	No

Overall, these results suggest that CBR-D's strategy of aiming for diversity among the query's NNs does improve the accuracy of predictions.

2. One of the central propositions of this thesis is that CBR-CD improves on previous case-differences-based algorithms by using gradients to guide the search for difference-cases. Figure 7.12 shows that substantially better results are obtained using gradients than without them (Diffs—Mean Grad vs. Diffs—No Grad). This demonstrates the merit of employing gradients as a search heuristic, regardless of the detail of how they are actually used (i.e., which of Approaches 2–4 is taken). It can also be seen that the best of the three gradient-based approaches for each dataset (Diffs—Best Grad) outperformed the mean (Diffs—Mean Grad); this shows the benefit of choosing the correct approach prior to problem solving.

3. CBR-CD performed well relative to other regression techniques on the datasets tested. It was statistically best-performing on one of the five datasets and statistically joint best on three others (LWLR was best performing on the fifth). These results are shown in Table 7.5 (again, the *Significant?* column indicates the result of a one-tailed t-Test with $P < 0.01$).

Table 7.5: CBR-CD vs. Other Algorithms

Dataset	*Best*	*2nd Best*	*Significant?*
Boston Housing	CBR-CD	LWLR	Yes
Tecator	LWLR	CBR-CD	No
Abalone	CBR-CD	CBR-B	No
CPU	LWLR	CBR-CD	Yes
Servo	M5′	CBR-CD	No

As already noted, it is to be expected that some algorithms will perform better in some circumstances than others. Overall, however, these results show CBR-CD's performance to be competitive with that of alternative state-of-the-art algorithms.

The results of Experiment 2 also highlight some general points of interest:

- The 'no free lunch' theorem, introduced in Section 2.4, states that different regression algorithms can be expected to have variable performance across different problem domains. This is clearly demonstrated in Experiment 2. LWLR, for example, performs best on Tecator and CPU but badly on Abalone, while CBR-B shows an opposite bias by performing relatively well on Abalone but badly on Tecator and CPU. This underlines the difficulty of comparing regression algorithms with one another: the optimal algorithm to use in any situation will always depend on the nature of the problem domain and the dataset describing it. M5', for example, is best performing on Servo and therefore the algorithm of choice in this domain. The fact that it is worst performing on Abalone has no bearing on this decision. With this in mind, the 'Aggregated Results' graph in Figure 7.12 should be interpreted as a measure of the different algorithms' consistency across different domains, not a measure of their overall worth.

- Chapter 2 discussed the notion that more complex regression algorithms are often needed to achieve good predictive performance in more complex domains. All five datasets used in Experiment 2 have target functions that are globally non-linear. For this reason, simple linear regression does not perform particularly well. The next simplest algorithm, CBR-B (which is equivalent to *k*-NN), also has relatively poor results. The more complex algorithms LWLR and CBR-CD, on the other hand, perform relatively better. This highlights the trade-off that often exists between simplicity and performance: more complex algorithms may be preferred where accuracy is of paramount importance, while greater transparency and slightly lower levels of accuracy may be more appropriate in other circumstances.

7.4 Discussion

Experiments 1 and 2 concentrated on assessing the performance of CBR-CD, as befits its central place in this thesis. They have shown that the complete CBR-CD algorithm performs well relative to other regression techniques. The question, posed at the beginning of Chapter 6, is whether this performance justifies its computational intensiveness. The answer must be that it depends on the application at hand. Good predictive performance is not the be-all and end-all for machine learning algorithms, but it is a most desirable characteristic. It is also the first of the three requirements that were placed on CBR-CD in Table 4.3; the other two being robustness (i.e., the ability to perform well in noisy domains) and simplicity (i.e., ease of construction and maintenance for new CBR-CD systems, and results that can be easily explained to users). Having described the operation and performance of CBR-CD, we are in a position to assess it against these three requirements:

Accuracy: Experiment 2 showed that CBR-CD provided good predictive performance across the datasets tested. Part of the reason for this may be due to its flexibility. It includes both 'lazy' and 'eager' elements: difference-cases and gradients are calculated in advance, but most of the problem solving process is delayed until a query is received. It also incorporates local and global elements: it is highly local in that each prediction is ultimately based on only a few cases, but global in that difference-cases may come from any part of domain space.

Robustness: The five datasets used in the experimental evaluation contrast with one another in several ways: number and type of attributes, number of instances, level of noise, degree of non-linearity in the target function, etc. The performance of most algorithms in Experiment 2 can be seen to vary across the datasets; LWLR, for example, is best-performing on the CPU dataset but among the worst on Abalone. CBR-CD, on the other hand, was either statistically best or second best on all five datasets. This consistency is an important point in its favour: applying CBR-CD to a new dataset is likely to give good performance regardless of the problem domain.

Simplicity: CBR-CD systems undergo an initial training period during which parameters are automatically set to suitable values. (The only parameter set manually is the approach taken to using gradients, but as stated previously, it would be a straightforward matter to automatically set this too.) New CBR-CD systems are therefore simple to construct. Although not a focus of this research, each prediction could also be accompanied by the cases used to generate it. The prediction process uses only the solution values of stored cases and is entirely transparent; much of the algorithm's complexity comes from the search process for applicable, non-noisy cases, and this process is not visible to users. (See Section 5.3 for more on user explanations in CBR-CD.)

In short, the CBR-CD algorithm was designed to place CBR at the centre of an accurate, robust regression system. The overall assessment must be that it successfully meets these design requirements.

Experiment 2 also assessed the performance of the second new CBR-based algorithm, CBR-D. This was shown to significantly outperform CBR-B on the datasets tested, thereby demonstrated the benefit of aiming for diversity among a query's NNs. The improvement shown was in line with expectations arising out of the theoretical discussion in Section 4.2, and would seem to justify the algorithm's modest increase in complexity over CBR-B.

7.5 Summary

This chapter presented the results of an experimental evaluation of the CBR-D and CBR-CD regression algorithms. Two sets of experiments were conducted. The first, Experiment 1, tested different variants of CBR-CD and found that performance was improved by all of the enhancements to the basic algorithm described in Chapter 5. The second, Experiment 2, compared the performance of CBR-D and CBR-CD against that of a number of standard regression algorithms. CBR-D was found to outperform CBR-B, confirming the value of encouraging diversity among a query's NNs. CBD-CD performed competitively relative to all other algorithms, with a high degree of consistency across different datasets. An assessment of CBR-CD found that it meets the requirements set down for it at the end of Chapter 4: it successfully applies CBR to regression in a manner that achieves accurate, robust performance in diverse domains.

Chapter 8

Conclusions and Future Work

The purpose of this research has been to apply case-based reasoning (CBR) to regression, that is, to bring one of the principal paradigms of modern machine learning to bear on one of the primary tasks in computational modelling. The account contained in this thesis has been presented in four parts:

1. A description of the regression task and the CBR approach to problem solving;
2. An analysis of previous approaches to using CBR for regression;
3. The proposal of two new CBR-based algorithms that address some of the limitations of previous approaches;
4. An account of the implementation and evaluation of these algorithms to determine whether or not they constitute an improvement over what came before.

These four parts are briefly summarised in the following section. The thesis then concludes with a short discussion and some possible ideas for future work.

8.1 Thesis Summary

8.1.1 Introduction to regression and CBR (Chapters 2 & 3)

The regression task simply involves predicting the value of a numeric variable, for example, predicting the value of a house given its location and number of bedrooms. Regression has been an active area of research since the late nineteenth century, when Francis Galton found that the heights (and intelligence levels) of parents were linearly correlated with those of their children. By graphing the heights of parents and children in a linear regression model, the height of a parent could be predicted from the height of its child (and vice versa). Many more sophisticated approaches to numeric prediction have since been developed, among them *k*-NN, locally-weighted linear regression (LWLR) and model trees. These algorithms often involve a trade-off whereby improved predictive accuracy is achieved at the expense of greater complexity and a need for larger amounts of training data.

Case-based reasoning, the approach taken to regression in this thesis, involves solving new problems by re-using the solutions to similar past problems. It was originally investigated as a model of human problem solving—people often tackle new problems by recalling how they dealt with similar situations in the past. As a machine learning paradigm, CBR is characterised by an instance-based approach whereby old problems (and their solutions) are stored as a set of cases in a case base. When a new problem arrives, one or more similar past problems are retrieved. Their solutions may be applied directly to the new problem, or may need to be adapted to fit it more precisely. CBR systems have a number of advantages over alternative approaches, chief among them the fact that they are often relatively simple to construct, use and understand. As a technology, CBR has been widely adopted for applications such as help-desk support and web-based recommender systems.

8.1.2 Previous approaches to using CBR for regression (Chapter 4)

The simplest way to use CBR for regression is referred to as CBR-B (CBR-Basic) in this thesis. Given a new problem (or *query case* to use CBR terminology), this algorithm retrieves a number of similar past cases and takes the weighted average of their solutions. A distance weighting function is used so that cases most similar to the query have the greatest influence on the predicted solution.

A more complex alternative approach was proposed in the 1990's by Kathleen Hanney and Mark Keane. Problem solving starts by finding the differences between the query and a similar past case. A search is then conducted for pairs of stored cases with the same differences between them; these cases indicate the effect that these differences have on the solution. For example, suppose we want to predict the value of a house that differs from the most similar stored case by a single bedroom. Suppose too that the case base contains a pair of cases that also differ by a single bedroom, and that this resulted in a price increase of €50,000. The query's value can then be predicted by taking the price of the most similar case and adding €50,000 to account for the difference between them.

Hanney and Keane proposed converting the differences between stored cases into a set of adaptation rules during an initial training phase. Their approach is referred to as CBR-AR (CBR-AdaptationRules) in this thesis.

8.1.3 New CBR-based regression algorithms (Chapters 4 & 5)

Both prior approaches to using CBR for regression have much to recommend them, but unfortunately, they also suffer from a number of drawbacks. CBR-B's principal problem is that the cases used to generate a solution may all be offset from the query in a particular direction. Suppose, for example, that we want to predict the value of a house with five bedrooms. If all of the retrieved cases have three or four bedrooms, then the predicted house price is likely to be too low. The solution is to try to choose a *diverse* set of similar cases with the query at their centre. This is the approach taken by CBR-D (CBR-Diverse), the first of the two new regression algorithms presented in this thesis.

The second new approach, referred to as CBR-CD (CBR-CaseDifferences), builds on the principles underlying CBR-AR and is more substantial. In common with its predecessor, it solves a query by looking at how it differs from a similar past case and then using a pair of stored cases to account for these differences. Its primary contribution lies in recognising that not all stored cases are useful for solving each particular query. In the housing example in Section 8.1.2 above, for example, it is not enough to know that the difference between two houses is a single bedroom. The impact of this difference on house price will depend on the location of the property as well as on the precise number of bedrooms involved (a change from one to two bedrooms may have a very different impact on house price than a change from five to six, for example). To address this problem, CBR-CD uses a second regression algorithm, LWLR, as a heuristic to guide the search for cases that are likely to be useful for solving each individual query. It also takes steps to ensure robust performance when predictions are based on noisy, real-world datasets.

8.1.4 Implementation and evaluation of new algorithms (Chapters 6 & 7)

A working CBR system was constructed that implemented several of the regression algorithms mentioned above: linear regression, LWLR, CBR-B, CBR-D, and CBR-CD. The system was designed in a simple object-oriented framework using the Python programming language.

Based on this implementation, an experimental evaluation was conducted to find out how the two new CBR-based regression algorithms—CBR-D and CBR-CD—perform in practice. Two sets of experiments were performed, each using five contrasting, publicly available datasets. The first looked at different variations of the CBR-CD algorithm and showed that each of its constituent parts makes a contribution towards improving predictive performance. The second compared CBR-D and CBR-CD with the following standard algorithms: linear regression, LWLR, CBR-B, and model trees (the Weka implementation of M5′ model trees was used). Good results were achieved for both: CBR-D provided an improvement over CBR-B, and CBR-CD performed well relative to all other algorithms. The overall conclusion from these experiments was that the two algorithms lived up to theoretical expectations and fulfilled the requirements set down for them.

8.2 Conclusions

The aim of this thesis was summed up in the Introduction:

> *Design a regression system using CBR that provides effective, robust performance across different problem domains.*

This aim has been met with two new algorithms, CBR-D and CBR-CD. The latter, in particular, demonstrated good performance relative to alternative algorithms and holds out the promise of being a useful general-purpose approach to regression.

Designing a new algorithm that outperforms all existing regression algorithms was not the primary goal of this research, however. Several innovative new algorithms have been developed over the past few decades; two that were mentioned in Chapter 2 are neural networks and radial basis function networks. There is no doubt that in many (and perhaps most) domains, these or other algorithms would yield more accurate results than CBR-D and CBR-CD. No, the value of this research is that an effective regression system was constructed *using case-based reasoning*, an approach that solves each new problem by retrieving similar past cases and re-using their solutions. There are two reasons why this approach is a useful one to take.

First, a regression system founded on case-based reasoning will inherit all of the advantages that have contributed to CBR's success in recent years. Chapter 3 discussed these advantages in some detail. CBR systems are often straightforward to construct and maintain since their main repository of knowledge is a set of historical cases. That is, CBR systems are generally *knowledge-light*. A second advantage is that CBR systems offer excellent performance during operation because each prediction is based on specific past cases rather than on a generalized model. These cases can also be presented to users to show how the solution was arrived at, making the problem solving process more transparent and allowing users to assess the quality of each proposed solution for themselves. Ease of construction and maintenance, good performance, transparent problem solving—these are all characteristics of the two new regression algorithms proposed in this thesis, CBR-D and CBR-CD.

The second reason that applying CBR to the problem of regression is useful is to demonstrate the viability of this approach. CBR systems have traditionally performed regression by retrieving some old cases similar to the new problem and taking the (weighted) average of their solutions. This approach is referred to as CBR-B in this thesis, and as shown in the experimental evaluation in Chapter 7, it does not perform particularly well. An alternative approach based on case differences was proposed by various researchers in the late 1990s. Unfortunately, this approach (referred to as CBR-AR in Chapter 4) suffered from a number of theoretical limitations that rendered it unsuitable for use in real-world domains. For this reason, the only evaluations of its performance that were carried out were based on artificial linear datasets. But as shown in the description and evaluation of CBR-CD in Chapters 5 and 7, these limitations *can* be overcome to produce an algorithm that provides robust, competitive performance on real-world datasets. Moreover, CBR-CD achieves good results without abandoning the core values of CBR, and in particular, without eschewing the knowledge-light approach that endows CBR with its chief advantage: simplicity. Domain knowledge contained within the CB is too often overlooked when knowledge-intensive approaches (i.e., those that advocate the addition of domain-specific, rule-based adaptation knowledge) are proposed as the only means of achieving good performance for more complex system tasks and domains. CBR-CD demonstrates that there is a great deal of implicit adaptation knowledge in the CB that, when properly harnessed, can improve the quality of predictions over pure local learning. Its problem-solving approach is straightforward: given the difference between the query and a neighbouring case, what can we learn from similar pairs of cases about how to bridge this gap? This approach is also applicable beyond regression, and more complex system tasks could be tackled along similar lines.

The majority of CBR applications in operation today are simple, retrieval-only systems. The case-differences approach described in this thesis and exemplified in CBR-CD shows CBR's potential for tackling a wider range of problems, and for doing so in a knowledge-light manner. Therein lies the primary contribution of this research.

8.4 Future Work

This thesis has introduced a practical, working CBR system that uses case differences for regression. As suggested above, the underlying principle might usefully be applied to more complex tasks such as planning and synthesis. Recent research has supported the idea that CBR may constitute a viable approach to solving planning problems [Kuchibatla and Muñoz-Avila 2006]. This suggests a promising direction for future research: assess whether or not case differences can be used for more complex forms of adaptation, and if so, build a working prototype system. A useful starting point for investigations in this direction might be provided by Section 5.4.1, which proposed bridging the gap between the query and a neighbouring case in several discrete steps. This approach did not work well for regression, but might prove more useful for planning and synthesis tasks where possible solutions can be verified for correctness.

Bibliography

Aamodt, A.: 1993, A Case-Based Answer to Some Problems of Knowledge-Based Systems, *Scandinavian Conference on Artificial Intelligence*, IOS Press, pp. 168–182.

Aamodt, A.: 1994, Explanation-driven case-based reasoning, *Wess, S., Althoff, K., Richter, M., eds., Topics in Case-Based Reasoning*, Springer Verlag, pp. 274–288.

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

Aha, D. W.: 1992, Tolerating noisy, irrelevant, and novel attributes in instance-based learning algorithms, *International Journal of Man-Machine Studies* **36**, pp. 267–287.

Aha, D.W.: 1997, Editorial, *Artificial Intelligence Review* **11**(1-5), *Special Issue on Lazy Learning*, pp. 1–6.

Aha, D, W.: 1998, The omnipresence of case-based reasoning in science and applications. *Knowledge Based Systems* **11**(5–6), pp. 261–273.

Aha, D. W., Kibler, D. F. and Albert, M. K.: 1991, Instance-Based Learning Algorithms, *Machine Learning* **6**, pp. 37–66.

Aha, D. W. and Wettschereck, D.: 1997, Case-based learning: Beyond classification of feature vectors, *Proceedings of the Ninth European Conference on Machine Learning*, Springer, pp. 329-336.

Althoff, K. D. and Richter, M. M.: 2001, Similarity and Utility in Non-Numerical Domains, *Mathematische Methoden der Wirtschaftswissenschaften*, Physika-Verlag, pp. 403–413.

Andrews, R., Diederich, J. and Tickle, A.: 1995, A survey and critique of techniques for extracting rules from trained artificial neural networks, *Knowledge Based Systems*, pp. 187–202.

Atkeson, C., Moore, A. and Schaal, S.: 1996, Locally weighted learning, *AI Review*.

Barzilay, R., McCullough, D., Rambow, O., DeChristofaro, J., Korelsky, T. and Lavoie, B.: 1998, A New Approach to Expert System Explanations, *Proceedings of INLG-1998*, pp. 78–87.

Bartsch-Spörl, B., Lenz, M. and Hübner, A.: 1999, Case-Based Reasoning: Survey and Future Directions, *Proceedings of XPS-99, volume 1570 of LNCS*, pp. 67–89, Springer.

Bergmann, R. and Wilke, W.: 1998, Towards a New Formal Model of Transformational Adaptation in Case-Based Reasoning, *Proceedings of the European Conference on Artificial Intelligence (ECAI '98)*, John Wiley and Sons.

Bergmann, R., Richter, M. M., Schmitt, S., Stahl, A. and Vollrath, I.: 2001, Utility-oriented matching: A new direction for case-based reasoning, *Proceedings of the 9th German Workshop on Case-Based Reasoning (GWCBR '01)*, pp. 264–274.

Bland, J. M. and Altman, D. G.: 1994, Statistic Notes: Regression towards the mean, *British Medical Journal* **308**, p. 1499.

Breiman, L., Freidman, J. H., Olshen, R. A. and Stone, C. J.: 1984, *Classification and Regression Trees*, Wadsworth.

Campbell, C., Evgeniou, T., Heisele, B. and Pontil M.: 2000, Machine Learning Strategies for Complex Tasks, *Proceedings of First IEEE-RAS International Conference on Humanoid Robots*, MIT, Springer Verlag.

Carbonell, J. G.: 1983, Derivational Analogy and its Role in Problem Solving, *Proceedings of the 3rd Annual National Conference on Artificial Intelligence AAAI-83*, Morgan Kaufmann.

Carroll, R. J. and Ruppert D.: 1988, *Transformation and Weighting in Regression*, Chapman and Hall, New York.

Cleveland, W.S.: 1979, Robust Locally Weighted Regression and Smoothing Scatterplots, *Journal of the American Statistical Association, Vol. 74*, pp. 829–836.

Cleveland, W. S. and Devlin, S. J.: 1988, Locally Weighted Regression: An Approach to Regression Analysis by Local Fitting, *Journal of the American Statistical Association, Vol. 83*, pp. 596–610.

Cleveland, W. and Loader, C.: 1995, Smoothing by Local Regression: Principles and Methods (with discussion), *Computational Statistics*.

Craw, S.: 2001, Case-Based Reasoning for Tablet Formulation, *Proceedings of the British Pharmaceutical Science Conference*, Cambridge University Press.

Cunningham, P., Nowlan, N., Delany, S. J. and Haahr, M.: 2003a, A case-based approach to spam filtering that can track concept drift, *The ICCBR'03 Workshop on Long-Lived CBR Systems, Trondheim, Norway.*

Cunningham, P., Doyle, D. and Loughrey, J.: 2003b, An Evaluation of the Usefulness of Case-Based Explanations, *Proceedings of the 5th International Conference on Case-Based Reasoning*, Springer, pp. 122–130.

d'Aquin, M., Badra, F., Sandrine, L., Lieber, J., Napoli, A. and Szathmary, L.: 2006, Adaptation Knowledge Discovery from a Case Base, *Proceedings of the European Conference on Artificial Intelligence (ECAI '06).*

Doyle, D., Cunningham, P., Bridge, D. and Rahman, Y.: 2004, Explanation Oriented Retrieval, *Proceedings of the 7th European Conference on Case-Based Reasoning*, Springer, pp. 157–168.

Draper, N. R. and Smith, H.: 1998, *Applied Regression Analysis*, Wiley Series in Probability and Statistics.

Duda, R. and Hart, P.: 1973, *Pattern Classification and Scene Analysis*, John Wiley & Sons, New York.

Eschenauer, H. A., Koski, J. and Osyczka, A.: 1990, *Multicriteria Design Optimization: Procedures and Applications*, Springer.

Fan, J. and Gijbels, I.: 1996, *Local Polynomial Modeling and its Applications*, Chapman and Hall, London.

Fan, J. and Marron, J. S.: 1993, Comment on [Hastie and Loader 1993], *Statistical Science* **8**(2), pp. 129–134.

Faries, J. and Schlossberg, K.: 1994, The effect of similarity on memory for prior problems, *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pp. 178–282, Hilldale, NJ: Lawrence Erlbaum.

Forsythe, D. and Buchanan, B.: 1989, Knowledge acquisition for expert systems: Some pitfalls and suggestions, *IEEE Transactions on Systems, Man, and Cybernetics* **19**(3), pp. 435–442.

Fuchs, B. and Mille, A.: 1999, A Knowledge-Level Task Model of Adaptation in Case-Based Reasoning, *Lecture Notes in Computer Science*, *Volume 1650*, Springer, p. 118.

Gabel, T. and Stahl, A.: 2004, Exploiting Background Knowledge when Learning Similarity Measures, *Proceedings of the 7th European Conference on Case-Based Reasoning*, Springer, pp. 169–183.

Galton, F.: 1886, Regression towards mediocrity in hereditary stature, *Journal of the Anthropological Institute* **15**, pp. 246–263.

Gentle, J. E.: 1998, QR Factorization, *Numerical Linear Algebra for Applications in Statistics*, Springer-Verlag, Section 3.2.2, pp. 95–97.

Hanney, K. and Keane, M. T.: 1996, Learning Adaptation Rules from a Case-Base, *Proceedings of the 3rd European Workshop on Case-based Reasoning*, pp. 179–192, Springer Verlag.

Hanney, K. and Keane, M. T.: 1997, The Adaptation Knowledge Bottleneck: How to Ease it by Learning from Cases, *Proceedings of the 2nd International Conference on Case-Based Reasoning*, pp. 359–370.

Harter, H. L.: 1983, Least Squares, *Encyclopedia of Statistical Sciences*, Kotz, S. and Johnson, N.L., eds., John Wiley & Sons, New York, pp. 593–598.

Hastie, T. and Loader, C.: 1993, Local regression: Automatic kernel carpentry, *Statistical Science* **8**(2), pp. 120–143.

Hettich, S., Blake, C. L. and Merz, C. J.: 1998, UCI Repository of machine learning databases. University of California, Irvine, CA.

Howe, N. and Cardie, C.: Examining Locally Varying Weights for Nearest Neighbor Algorithms, *Proceedings of the Second International Conference on Case-Based Reasoning*, Springer, pp. 455–466.

Hüllermeier, E.: 2005, Cho-k-NN: A method for combining interacting pieces of evidence in case-based learning, *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI '05)*.

Jain, A., Sarda, P. and Haritsa, J.: 2003, Providing Diversity in K-Nearest Neighbor Query Results, Tech. Report TR-2003-04, DSL/SERC, Indian Institute of Science.

Jarmulak, J., Craw, S. and Rowe R.: 2001, Using Case-Base Data to Learn Adaptation Knowledge for Design, *Proceedings of the 17th International Conference on Artificial Intelligence*, Morgan Kaufmann.

John, G., Kohavi, R. and Pfleger, K.: 1994, Irrelevant features and the subset selection problem, *Proceedings of the International Conference on Machine Learning*.

Kira, K. and Rendell, L.: 1992, A practical approach to feature selection, *Proceedings of the Ninth International Conference on Machine Learning*, Morgan Kaufmann, pp. 249–256.

Kohavi, R. and John, G., 1997a, Wrappers for feature subset selection, *Artificial Intelligence, Vol. 97, No. 1–2*, pp. 273–324.

Kohavi, R., Langley, P. and Yun, Y.: 1997b, The utility of feature weighting in nearest-neighbor algorithms, *Proceedings of the European Conference on Machine Learning (ECML–97)*.

Kolodner, J.: 1992, An introduction to case-based reasoning, *Artificial Intelligence Review* **6**(1), pp. 3–34.

Kolodner, J.: 1993, *Case-based reasoning*, San Mateo, CA: Morgan Kaufmann.

Kuchibatla, V. and Muñoz-Avila, H.: 2006, An Analysis on Transformation Analogy: General Framework and Complexity, *Proceedings of the 8th European Conference on Case-Based Reasoning*, Springer, pp. 458–473.

Leake, D.: 1995, Adaptive similarity assessment for case-based explanation, *Journal of Experimental and Theoretical Artificial Intelligence* **7**, pp. 407–428.

Leake, D.: 1996, CBR in Context: The Present and Future, *Case-Based Reasoning: Experiences, Lessons, and Future Directions*, Menlo Park: AAAI Press/MIT Press.

Leake, D., Kinley, A. and Wilson, D.: 1995, Learning to Improve Case Adaptation by Introspective Reasoning and CBR, *Proceedings of the 1st International Conference on Case-Based Reasoning*, Springer-Verlag, pp. 229–240.

Leake, D. and Wilson, D.: 1998, Categorizing case–base maintenance: Dimensions and directions, *Proceedings of EWCBR–98*, *Advances in Case–Based Reasoning*, Springer–Verlag.

Lehman M. M. and Belady, L.: 1985, *Program Evolution – Processes of Software Change*, Academic Press.

Lenat, D. and Feigenbaum, E.: 1991, On the thresholds of knowledge, *AI Journal* **47**(1–3), pp. 185–250.

Lenz, M.: 1999, *Case Retrieval Nets as a model for building flexible information systems*, Ph.D. thesis, Faculty of Mathematics and Natural Sciences, Humboldt University, Berlin.

Lenz, M. and Burkhard, H. D.: 1996, Case retrieval nets: Basisc ideas and extensions, *KI-96: Advances in Artificial Intelligence*, number 1137 in *Lecture Notes in Artificial Intelligence*, Springer Verlag, pp. 227–239.

Lippmann, R. P.: 1987, An Introduction to Computing with Neural Nets, *IEEE ASSP Magazine, April 1987*.

Loader, C.: 1999, *Local Regression and Likelihood*, Springer; LOCFIT local regression software available at http://cm.bell-labs.com/cm/ms/departments/sia/project/locfit/index.html

Lorenzi, F. and Ricci, F.: 2005, Case-Based Recommender Systems: a Unifying View, *Intelligent Techniques for Web Personalization*, Springer Verlag, pp. 89–113.

Loughrey, J. and Cunningham, P.: 2005, Using early stopping to reduce overfitting in wrapper-based feature weighting, *Technical Report TCD-CS-2005-41*, Department of Computer Science, Trinity College Dublin.

McGinty, L. and Smyth, B.: 2003, On the Role of Diversity in Conversational Recommender Systems, *Proceedings of the Fifth International Conference on Case-Based Reasoning (ICCBR-03)*, Springer, pp. 276–290.

McSherry, D.: 1998, An adaptation heuristic for case-based estimation, *Proceedings of the 4th European Workshop on Case-Based Reasoning*, pp. 184–195, Springer.

Michalski, R. S.: 1983, A theory and methodology of inductive learning, *Machine Learning, an A. I. Approach*, Tioga publishing company, pp. 83–134.

Mitchell, T. M.: 1997, *Machine Learning*, McGraw-Hill.

Moore, J.: 1994, *Participating in Explanatory Dialogues*, MIT Press.

Motulsky, H. J.: 1999, Curvefit.com, the complete guide to nonlinear regression, GraphPad Software, available at http://www.curvefit.com.

Noether, G. E.: 1984, Nonparametrics: The Early Years-Impressions and Recollections, *American Statistician* **38**(3), pp. 173–178.

Norvig, P.: 2000, *Python for Lisp Programmers*, online essay available at http://www.norvig.com/python-lisp.html.

Nugent, C. and Cunningham, P.: 2004, A case-based explanation system for ‘black-box’ systems, *ECCBR 2004 Workshop Proceedings*, pp. 155–164.

Orr, M. J.: 1996, Introduction to radial basis function networks, *Technical report*, Institute for Adaptive and Neural Computation of the Division of Informatics at Edinburgh University, Scotland, UK.

Plate, T. A.: 1994, Estimating analogical similarity by dot-products of Holographic Reduced Representations, *Advances in Neural Information Processing Systems* **6**, Morgan Kaufmann, pp. 1109–1116.

Porter, B.: 1989, Similarity assessment; Computation vs. representation, *Proceedings from the Case-Based Reasoning Workshop, Pensacola Beach, Florida, May-June 1989*, Morgan Kaufmann, pp. 82–84.

Porter, B. W., Bareiss, R. and Holte, R. C.: 1990, Concept learning and heuristic classification in weak theory domains, *Artificial Intelligence* **45**(1-2), pp. 229–263.

Quinlan, J. R.: 1992, Learning with continuous classes, *Proceedings of the Fifth Australian Joint Conference on Artificial Intelligence*, World Scientific, pp. 343–348.

Quinlan, J. R.: 1993, *C4.5: Programs for machine learning*, Morgan Kaufmann.

R Development Core Team: 2005, R: A language and environment for statistical computing, R Foundation for Statistical Computing, Vienna, Austria, ISBN 3-900051-07-0, URL http://www.R-project.org.

Rendell, L.: 1986, A general framework for induction and a study of selective induction, *Machine Learning* **1**, pp. 177–226.

Richter, M. M.: 1995, The knowledge contained in similarity measures, Invited Talk at the 1st International Conference on Case-Based Reasoning, ICCBR-95.

Richter, M. M.: 1998, Introduction, *Case-Based Reasoning Technology: From Foundations to Applications*, LNAI 1400, Springer.

Roth-Berghofer, T. and Iglezakis, I.: 2001, Six Steps in Case-Based Reasoning: Towards a Maintenance Methodology for Case-Based Reasoning Systems, *Proceedings of the 9th German Workshop on Case-Based Reasoning (GWCBR-2001)*, Shaker-Verlag, pp. 198–208.

Rubin, D. B.: 1996, Multiple imputation after 18+ years (with discussion), *Journal of the American Statistical Association* **91**, pp. 473–489.

Runyon, R. P. and Haber, A.: 1991, *Fundamentals of Behavioral Statistics*, 7th Edition, McGraw-Hill International Editions, Singapore.

Russell, S. J. and Norvig, P.: 2002, *Artificial Intelligence: A Modern Approach*, 2nd Edition, Prentice Hall.

Schank, R.: 1982, *Dynamic Memory: A Theory of Learning in Computers and People*, Cambridge, England: Cambridge University Press.

Seber, G. A. F. and Wild, C. J.: 1989, *Nonlinear Regression*, New York, John Wiley and Sons.

Schmidt, H., Norman, G. and Boshuizen, H.: 1990, A cognitive perspective on medical expertise: Theory and implications, *Academic Medicine* **65**(10), pp. 611–621.

Schmidt, R. and Gierl, L.: 2001, Case-Based Reasoning for Medical Knowledge-based Systems, *International Journal of Medical Informatics* **64**(2-3), pp. 355–367.

Shannon, C. E.: 1948, A mathematical theory of communication, *Bell System Technical Journal* **27**, pp. 379–423 and 623–656.

Smyth, B. and McKenna, E.: 2001, Competence models and the maintenance problem, *Computational Intelligence* **17**(2), pp. 235–249.

Smyth, B. and Keane, M. T.: 1995, Remembering to forget: A competence-preserving deletion policy for case-based reasoning systems, *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, pp. 377–382.

Smyth, B. and Keane, M. T.: 1996, Using adaptation knowledge to retrieve and adapt design cases, *Journal of Knowledge Based Systems* **9**(2), pp. 127–135.

Smyth, B. and Keane, M. T.: 1998, Adaptation-guided retrieval: Questioning the similarity assumption, *Artificial Intelligence* **102**, pp. 249–293.

Spalazzi, L.: 2001, A Survey on Case-Based Planning, *Artificial Intelligence Review* **16**(1), pp. 3–36.

Stahl, A. and Bergmann, R.: 2000, Applying recursive CBR for the customization of structured products in an electronic shop, *Proceedings of the 5th European Workshop on Case-Based Reasoning*, Springer Verlag.

Stahl, A. and Gabel, T.: 2003, Using Evolution Programs to Learn Local Similarity Measures, *Proceedings of the 5th International Conference on Case-Based Reasoning*, Springer, pp. 537-551.

Stroud, K. A.: 2001, *Engineering Mathematics (5th ed.)*, Palgrave Macmillan, p. 585.

Thagard, P.: 2005, *Mind: Introduction to cognitive science (2nd ed.)*, Cambridge, MA: MIT Press, p.45.

Turing, A. M.: 1950, Computing machinery and intelligence, *Mind* **59**, pp. 433–460.

Upton, G. and Cook, I.: 2002, *A Dictionary of Statistics*, Oxford University Press, entry: ‘normal distribution’, p. 261.

Uysal, İ. and Güvenir, H. A.: 1999, An overview of regression techniques for knowledge discovery, *Knowledge Engineering Review* **14**(4), pp. 319–340.

Wang, Y. and Witten, I. H.: 1997, Inducing model trees for continuous classes, *Proceedings of the Poster Papers of the Ninth European Conference on Machine Learning*, pp. 128–137.

Wettschereck, D., Aha, D. W. and Mohri, T.: 1997, A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms, *Artificial Intelligence Review Special Issue on Lazy Learning Algorithms*.

Widmer, G. and Kubat, M.: 1996, Learning in the Presence of Concept Drift and Hidden Contexts, *Machine Learning* **23**, pp. 69–101.

Wilke, W. and Bergmann, R.: 1998, Techniques and Knowledge Used for Adaptation During Case-Based Problem Solving, *Proceedings of IEA-98-AIE*, Springer-Verlag.

Wilke, W., Vollrath, I., Althoff, K. D, and Bergmann, R.: 1997, A framework for learning adaptation knowledge based on knowledge light approaches, *Proceedings of the 5th German Workshop on Case-Based Reasoning*.

Wiratunga, N., Craw, S. and Rowe, R.: 2002, Learning to Adapt for Case-Based Design, *Proceedings of the 6th European Conference on Case-Based Reasoning*, Springer.

Witten, I. H. and Eibe, F.: 2000, *Data Mining: Practical machine learning tools with Java implementations*, Morgan Kaufmann.

Wolpert, D. and Macready, W. G.: 1997, No free lunch theorems for optimization, *IEEE Transactions on Evolutionary Computation* **1**(1), pp. 67–82.

Zhang, J., Yim, Y. and Yang, J.: 1997, Intelligent selection of instances for prediction in lazy learning algorithms, *Artificial Intelligence Review* **11**. pp. 175–191.