Department of Computer Science

The University of Waikato
Te Whare Wānanga o Waikato

Hamilton, New Zealand

Designing Similarity Functions

Leonard Trigg

This thesis is submitted in partial fulfilment of the requirements for the degree of Doctor of Philosophy at The University of Waikato.

October 1997

Abstract

The concept of similarity is important in many areas of cognitive science, computer science, and statistics. In machine learning, functions that measure similarity between two instances form the core of instance-based classifiers. Past similarity measures have been primarily based on simple Euclidean distance. As machine learning has matured, it has become obvious that a simple numeric instance representation is insufficient for most domains. Similarity functions for symbolic attributes have been developed, and simple methods for combining these functions with numeric similarity functions were devised. This sequence of events has revealed three important issues, which this thesis addresses.

The first issue is concerned with combining multiple measures of similarity. There is no equivalence between units of numeric similarity and units of symbolic similarity. Existing similarity functions for numeric and symbolic attributes have no common foundation, and so various schemes have been devised to avoid biasing the overall similarity towards one type of attribute. The similarity function design framework proposed by this thesis produces probability distributions that describe the likelihood of transforming between two attribute values. Because common units of probability are employed, similarities may be combined using standard methods. It is empirically shown that the resulting similarity functions treat different attribute types coherently.

The second issue relates to the instance representation itself. The current choice of numeric and symbolic attribute types is insufficient for many domains, in which more complicated representations are required. For example, a domain may require varying numbers of features, or features with structural information. The framework proposed by this thesis is sufficiently general to permit virtually any type of instance representation—all that is required is that a set of basic transformations that operate on the instances be defined. To illustrate the framework's applicability to different instance representations, several example similarity functions are developed.

The third, and perhaps most important, issue concerns the ability to incorporate domain knowledge within similarity functions. Domain information plays an important part in choosing an instance representation. However, even given an adequate instance representation, domain information is often lost. For example, numeric features that are modulo (such as the time of day) can be perfectly represented as a numeric attribute, but simple linear similarity functions ignore the modulo nature of the attribute. Similarly, symbolic attributes may have inter-symbol relationships that should be captured in the similarity function. The design framework proposed by this thesis allows domain information to be captured in the similarity function, both in the transformation model and in the probability assigned to basic transformations. Empirical results indicate that such domain information improves classifier performance, particularly when training data is limited.

Acknowledgments

Researching and writing a DPhil thesis is a long and difficult process, and I would like to thank those people who have helped me during the course of my study. Their support has made this process enjoyable.

I would like to thank John Cleary—I could not have hoped for a better supervisor. His encouragement, ideas, and assistance with the trickiest mathematical problems has been inspirational. I must confess, after some of our discussions, my brain felt like it was about to explode ☺.

I have received unending support from my family. For this, I am eternally grateful. Tracy has provided emotional support, supported me financially, and put up with my grad-student ways. My parents have encouraged me throughout my life, and have provided support whenever needed.

Many people have helped in the transformation of this thesis from first draft to final copy. In particular, I would like to thank the following for their comments—John, Bill Teahan, Stu Inglis, Geoff Holmes, Kai Ming Ting, and Ian Witten. Thank you all.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Consider the task of comparing two weather maps. There are many reasons why we might make such a comparison. For example, one map may represent the current weather situation, while the other represents the weather at some time in the past. If the two maps are sufficiently similar, looking at what happened after the historical situation may help predict how the current weather will change.

Weather maps contain high-level features, for example, high and low-pressure systems. To compare two images for similarity, we must first have an idea of which features in one image correspond to which features in the second. The maps in Figure 1.1 are from consecutive time periods, so it is easy to determine likely feature correspondences. When the placement of the features varies it is less obvious which correspondence should be used. Consider the effect of incremental changes to one of the images. At some point a small

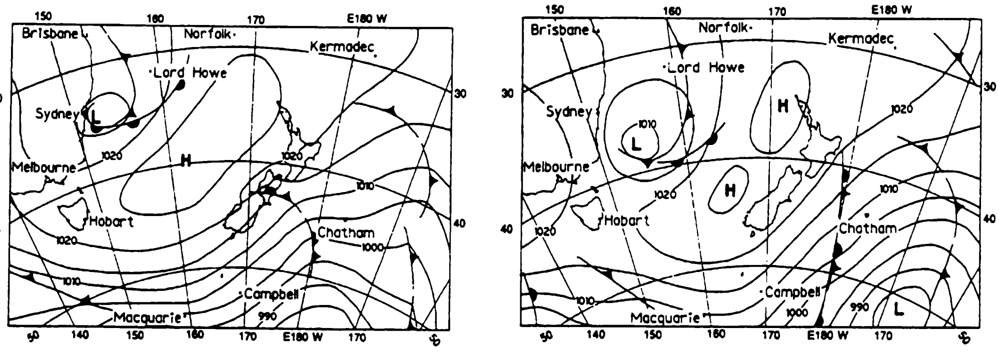

Figure 1.1: Two barometric pressure contour maps taken 24 hours apart

change in the image will result in a new feature correspondence being considered best. This in turn will give a sudden change in predictions made by the system, even though only a small change in the image itself has occurred.

A further complication arises when the number of features in the two images is different. In this case a simple one-to-one correspondence cannot be carried out. One feature in the first image may have moved outside the boundaries of the second image, or it may have merged with another feature to become one. In the left-hand image of Figure 1.1 the high-pressure system splits in two in the right hand image, so somehow the similarity between a single feature and two features must be evaluated.

Missing information presents another problem. This may occur when features are beyond the image borders, such as the low-pressure feature in the lower right of the images in Figure 1.1. If the remainder of the image is sufficiently similar, the image may still prove useful when making predictions. Similarity is rarely determined by measurement of one property—often many separate components must be taken into account. In the weather domain position, size and shape of features must be considered and somehow combined to give an indication of overall relatedness. Care must be taken to avoid biasing the measure towards one component or another.

The similarity of two images is often given in terms of a single number calculated by a *similarity function*. It is clear that the similarity function should in some sense depend on the underlying domain from which the images are generated. For example, in the weather domain a difference in the vertical position of a feature is much more significant than a difference in the horizontal position, because weather patterns naturally tend to move from west to east. Similarly, the function should be sensitive to the purpose of the comparisons. If the objective is to predict the weather over New Zealand, features closer to New Zealand should have more importance than features over Australia. Features used to predict wind speed may be different from those used to predict cloud cover. Domain knowledge is crucial to the performance of a similarity function.

The concept of similarity is important in many areas of cognitive science, computer science, and statistics. In cognitive science and psychology, similarity plays an important role in models of knowledge and behaviour. Individuals use similarity when categorizing objects, forming concepts and making generalisations. Similarity is employed in accounts of stimulus and response generalisation in learning, and to explain errors in memory. In computer science and statistics, a common task is pattern recognition. In the most general terms this involves determining whether a group of numbers (or *pattern*) is recognised as some previously observed pattern. The pattern may represent anything from an image of a vehicle, measurements taken from sensors in an industrial process, the waveform produced by a human voice, or information about a person's health. If a pattern is similar enough to a previously observed pattern it is "recognised" and appropriate action can be taken, such as determining whether to issue a speeding ticket, adjusting some process control parameter, allowing access to a secure area, or suggesting a medical diagnosis.

Often similarity is interpreted in a geometric sense—objects are represented as a point in geometric space and the similarity of two objects is the negation of the distance between them as measured by a metric function. Small distances between points result in high similarities and large distances between points result in low similarities. This geometric interpretation implies certain properties that upon closer inspection do not reflect our intuitive understanding of similarity. For example, a geometric interpretation implies that the distance from any object to itself is zero, although we intuitively perceive two identical twins as more similar to each other than two identical cars. This is presumably because there are many identical cars but few identical people. A geometric interpretation also implies that similarity is symmetric. The asymmetry of human similarity judgement is evident in similes and metaphors. We say, "Turks fight like tigers" and not "tigers fight like Turks," since tigers epitomise fighting spirit. Sometimes both directions are used but with different meanings. "Life is like a play," says that people play roles, while "a play is like life" says that a play captures important aspects of our lives.

Tversky (1977) describes several studies in psychology into how humans interpret similarity. The asymmetry of human judgement of similarity is confirmed in many experiments. For example, in an experiment carried out by Rosch (1975), subjects were required to make a statement like "*a* is essentially *b*" when presented with two objects. In a domain such as comparing numbers, objects were divided into prototypical (such as multiples of ten) versus variants (such as other numbers). Subjects overwhelmingly preferred placing the prototype in the position of *b* (the referent) and the variant in the position of *a* (the subject). For instance, the sentence "103 is virtually 100" was preferred over the sentence "100 is virtually 103." These experiments suggest that if a similarity measure is to correspond with human interpretation of similarity, symmetry is not a requirement.

1.1 Motivation

One area of machine learning where similarity functions play an important role is instance-based learning. The usual task in instance-based learning is to classify an instance by relating it to a library of examples for which the classification is known. The representation of an instance is typically in terms of a set of attributes common to all the instances, such as colour or height. An instance is defined by the values it has for each property, such as red or 178cm. The similarity measure compares an attribute value in one instance with the corresponding attribute value in the second instance. The similarities between each of the attributes are combined to give an overall indication of similarity. In a simple instance-based learner, a new instance is given the same category as the most similar instance in the library. Instance-based learning algorithms typically use a similarity measure designed to perform well in a variety of domains with little or no modification.

Case-based reasoning algorithms also employ measures of similarity (Riesbeck and Schank, 1989). These algorithms are more sophisticated than instance-based learners in both the representation used for objects and the task. Objects are called *cases* because an early application of case-based reasoning was to search for historic legal cases that provide a precedent relevant to a current case. Unlike the attribute-value pair representation of instances, there is no simple representation for cases. This is because case-based reasoning applications are typically very domain specific, and the domain determines the case representation. A case may contain features unique to itself or it may contain structural information. For example, a description of a "recipe" case would consist of a list of ingredients and procedural information about how the meal is prepared. The task carried out by case-based reasoning systems is often more than simple classification. An example case-based reasoning system might plan a meal by taking the dietary requirements of guests and modifying existing recipes accordingly.

Many of the issues dealt with in measuring the similarity between two objects are intimately related to the design of evaluation functions. Evaluation functions take a single object and give a number representing the "goodness" of that object. Evaluation functions can be viewed as measuring the similarity between the current object and some unknown "best possible object" (or the distance from an unknown "worst object"). Two common uses of evaluation functions are in game playing and in genetic algorithms.

In game playing, a possible move is considered and the resulting game state is evaluated. It is desirable to be in a state with high goodness, so the move that produces the best next state according to the evaluation function is played. The quality of the evaluation function directly affects the performance of the game-playing program.

Genetic algorithms use the principles of natural selection to search for an optimal solution to some multidimensional problem. An initial "population" of potential solutions is randomly generated. Each is then evaluated as to how well it solves the problem. Solutions with sufficiently high goodness are chosen as parents for a new population. Parents are randomly mutated and merged with other parents to produce new potential solutions. Over a number of generations the solutions represented by the population converge to a solution that is at least locally optimum if not globally optimum. A better function for evaluating individual solutions results in faster convergence and often (since randomness is involved) a better final solution.

The primary goal of machine learning is to bring the practical benefits of learning to computer programs, although this results-driven philosophy can sometimes inhibit the development of solutions that have an overall coherence. Prior to the development of machine learning, similarity functions were designed primarily to handle attributes with real or integer values. This stemmed from the geometric interpretation of similarity—numeric attributes have an obvious distance measure: Euclidean distance. However, not all attributes have such a natural interpretation of similarity, for example, colour or brand name. Researchers in machine learning who attempted to solve real world problems soon discovered that effective treatment of symbolic attributes is essential. Some similarity functions designed specifically with symbolic attributes in mind have been developed. A review of these similarity functions is given in Chapter 2. Functions for dealing with symbolic attributes are often combined in an *ad hoc* manner with functions for dealing with continuous attributes. Very few algorithms have any conceptual basis for the manner in which different attribute types are treated or how the measures for different attributes are combined.

A desirable goal of machine learning is to develop algorithms that can be applied to many problem domains. However, rather than developing a general method for tailoring similarity functions to each domain, the usual approach is to eliminate domain specific information altogether. Typically all attributes are treated as one of the two basic types, numeric and symbolic. Clearly this can cause problems. For example, representing the months in symbolic form loses information about the order in which months occur, and that the months are cyclic. This type of information should be incorporated in similarity functions.

1.2 Objectives

The objective of this thesis is to develop a framework for the formulation of similarity functions that addresses the issues discussed above. These issues may be separated into technical considerations, which are specific problems that must be solved in order to provide any measure of similarity, and performance issues, which highlight desirable properties of a similarity function.

Technical considerations will vary from domain to domain, but the design framework should provide a method for dealing with all of them. The following are the technical issues that we are concerned with:

- Different feature types and complexity. Feature types can vary from simple integers, to sets of symbolic values, to complex structural information. The design framework must provide a consistent approach to developing type specific similarity measures.
- Multiple features. Often a comparison is made between objects with multiple unlabelled features (like the multiple high and low-pressure systems of the weather maps). Some correspondence between features in two objects must be made for any similarity measure. This is made more difficult when the number of features is different in the two objects. The particular feature correspondence can have a large effect on the calculated similarity. It is imperative that the design framework is able to deal with multiple features intelligently.
- Missing information. Objects of which we have incomplete knowledge may still be useful, and so a meaningful comparison should be made. Domain knowledge may play a part in dealing with missing information. For example, when faced with an image of the left half of a

person's face, we can still identify that person on the street because we know that human faces are approximately symmetrical.

- Combining multiple measures of similarity. A frequent problem is that objects being compared often consist of several attributes that are compared individually. Combining individual similarity measures for each attribute must be done carefully and consistently to avoid biasing the overall measure towards one attribute.

Many of these problems have not been solved adequately in past research, and certainly not under a common framework which addresses all of the above considerations.

Performance issues are concerned with ensuring that a similarity measure designed for a particular domain performs well in that domain. For the classification task, performance might be measured solely on classification accuracy. In general, performance refers to how much information the similarity measure provides about the domain, and how robust the measure is over a range of conditions. For example, a reasonably robust similarity measure should not be overly sensitive to small changes in the objects being compared. The following factors are important in ensuring good performance in a similarity measure:

- The design framework must allow the inclusion of domain information. A measure that employs domain specific information intelligently should perform better than methods that are insensitive of domain dependent characteristics.
- The design framework should ensure that the similarity measure is smooth with respect to the underlying attribute space. This smoothness is an important factor in the robustness of a similarity measure.

Similarity measures produced within the design framework must perform comparably to other measures, if not better. Several similarity measures have been developed for general purpose learning methods such as instance-based learners. One expects that a coherent solution to the previous problems should perform at least as well as these methods on standard tasks.

1.3 Thesis Claims

This thesis proposes a framework for the design of similarity functions for which we make the following claims:

1. The framework is general enough to encompass many different object types.
2. The framework allows domain knowledge to be included in similarity function design.
3. The framework permits similarity functions that compare configurations of multiple objects.
4. The framework handles missing information.
5. The framework coherently combines multiple sources of similarity.
6. The resulting similarity functions are smooth with respect to small changes in the object space.

In addition we develop a machine learning application that utilises similarity functions designed within the framework, and we show that the resulting instance-based learner performs well under a variety of conditions.

1.4 Terminology

Instance-base learning and case-based reasoning come under the umbrella of lazy learning, which as a general model involves using a library of past example solutions as a guide in carrying out the current task. These examples are often interchangeably called *exemplars*, *instances* and *cases*. In instance-based learning the task is to classify each instance in a *test set* into a *category* or *class*, using preclassified instances from a *training set*. Because the classification of the training instances is known, this process is called *supervised learning*. Each instance is represented by a fixed set of *attributes* or *features*. These attributes are usually of a number of basic types. *Continuous* attributes (such as real numbers) have values that are ordered and for which there are always intermediate values between any two values. *Discrete* attributes (such as integers) have a number of ordered set values. *Symbolic* (or *nominal*) attributes have a set of unordered possible values. *Boolean* attributes have two values, one representing true, and the other representing false. Instances described by *n* attributes are points in an *n*-dimensional *instance space*.

The terms *similarity* and *distance* are often used interchangeably. A high similarity implies a small distance, and a low similarity implies a large distance. However distance is usually associated with the idea of geometric distance, and exhibits the properties of a metric function. Formally, a distance function d is a *metric* if it obeys the following three principles.

Minimality: $d(a,b) \geq d(a,a) = 0$.

This principle says that the distance from an object to itself is zero, and that no object can be closer to it than itself. It also implies that the distance between an object and itself is the same for all objects.

Symmetry: $d(a,b) = d(b,a)$.

This principle states that the distance from an object a to an object b is the same as if the distance were measured in the other direction, that is, from b to a.

The triangle inequality: $d(a,b) + d(b,c) \geq d(a,c)$.

The triangle inequality puts an upper limit on the distance between object a and object c, given the distance between them via object b.

Throughout this thesis, *distance* may be thought of as *dissimilarity*. Where the geometric interpretation is intended it will be explicitly stated.

1.5 Thesis Overview

The framework for similarity function design proposed by this thesis has its roots in algorithmic complexity. The next chapter provides the reader with a brief introduction to inductive inference, complexity theory, and their relationships.

Chapter 3 discusses the role of similarity functions in machine learning. In particular, instance-based learning is examined in detail. Work related to the ideas of smoothness and robustness in a similarity measure is presented. Previous research related to treating different attribute types coherently is examined.

Chapter 4 develops a framework for uniform similarity measure formulation. The key idea is to interpret the similarity between two objects as the probability of the first object transforming to the second object. The higher the transformation probability the higher the similarity. Objects that are extremely unlikely to transform to each other are very dissimilar. A method for designing such similarity measures is presented, based on ideas from complexity theory. Example similarity measures for several simple domains are also developed.

The implementation of a nearest neighbour classifier that incorporates a transformation based similarity measure is presented in Chapter 5. Some problems specific to the application are addressed within the framework. The classification performance of the resulting instance-based learner is evaluated, and it is shown that the measure compares favourably with other machine learning algorithms.

Chapter 6 gives a summary of the thesis and its conclusions, and explores areas for future research.

Chapter 2

Inductive Inference and Algorithmic Complexity

This chapter presents background information from fields relevant to the development of complexity-based similarity. A brief introduction to inductive inference and associated problems is provided. The major problem is the difficulty of objectively comparing the simplicity of competing hypotheses, and this was a motivating factor in the development of algorithmic complexity. Algorithmic complexity defines a measure for the information content of objects. The minimum information encoding inference procedure (which is based on algorithmic complexity) has since become popular, and an introduction to this procedure is provided.

2.1 Introduction

Induction is defined in the Oxford English Dictionary as "the process of inferring a general law or principle from the observations of particular instances." This is also called *inductive inference*. Induction is the primary method by which we understand and explain the world. How we carry out induction is vitally important if we wish to form sound conclusions about how the world operates. An alternative definition of inductive inference may be expressed as: "given a data set D and a set of hypotheses $H=\{H_1, H_2, ...\}$ choose the hypothesis that best explains the data." By this definition a single hypothesis is accepted as the best explanation of the phenomena and all others are rejected. *Inductive reasoning* is the more general concept of assigning a probability (or credibility) to a particular hypothesis—all hypotheses have some degree of belief associated with them.

A fundamental difficulty in induction is evaluating which hypothesis is "best." By what criterion should hypotheses be judged? A good hypothesis will obviously explain most of the data accurately. If two hypotheses explain the data equally well, which one should be preferred (or should they both be used)? Philosophers have been aware of these problems for a long time. Two approaches to the problem of multiple hypotheses are common today: Epicurus' principle of multiple explanations; and Occam's principle of the simplest explanation (known as Occam's razor).

> Epicurus' Principle of Multiple Explanations: *if more than one theory is consistent with the data, keep them all.*

The following information is from Oates (1957), and Li and Vitanyi (1992). The Greek philosopher of science Epicurus (*circa* 342–270 BC) maintained that if several explanations are equally in agreement with a phenomena, we must keep them all for two reasons. First, by making use of multiple explanations it may be possible to achieve a higher degree of precision. Second, it would be unscientific to choose one explanation over another when both explain the phenomena equally well. Epicurus claims this would be to "abandon scientific inquiry and resort to myth." His follower Lucretius (95–55 BC) illustrates the utility of the principle of multiple explanations with the following example.

> There are things too, not a few for which it is not sufficient to assign one cause; you must give several, one of which at the same time is the real cause. For instance should you see the lifeless body of a man lying at some distance, it would be natural to mention all the different causes of death, in order that the one real cause of that man's death be mentioned among them. Thus you may not be able to prove that he died by steel or cold or from disease or haply from poison; yet we know that it is something of this kind which has befallen him; and so in many other cases we may make the same remark.

When calculating probabilities a related intuition leads to "the principle of indifference." If there is absolutely no other evidence of the conditions under which a group of events occur, the principle of indifference suggests that they be considered equally likely. What the principle of indifference really highlights is the need to make as much use as possible of prior information. Consider an urn containing white, green, and red balls. The principle of indifference would estimate a probability of 1/3 for drawing a ball of each colour. For a colour-blind person to whom red and green both appear the same however, the principle of indifference would estimate that the probability of drawing a white ball is 1/2, and the probability of drawing a dark (i.e. red or green) ball is 1/2. The principle of indifference suggests different probability distributions even though the balls in the urn are the same! Prior knowledge is vital if we are to avoid this dilemma.

It is important to explain the difference between the principle of multiple explanations and the principle of indifference. Returning to Lucretius' example, the probability of each cause of death could be estimated, either from prior knowledge, or from the principle of indifference in the lack of such knowledge. The principle of multiple explanations requires the use of all possible explanations rather than simply selecting the most probable.

The second and more sophisticated principle is Occam's razor. This often cited principle is attributed to William of Ockham (*circa* 1290–1349). In contrast to the principle of multiple explanations, it states

> Occam's Razor Principle: *entities should not be multiplied beyond necessity.*

The typical interpretation of this is: if there are several hypotheses that explain the observed data equally well, prefer the simplest hypothesis. This makes sense from a practical viewpoint—if several theories are equally good, why not use the simplest? A difficulty arises when we try to determine which hypothesis is the simplest—is x^{1000} simpler than $ax^{12}+bx^{5}+c$? It seems there can be no objective answer to this problem. The field of algorithmic complexity however, tells us there is an objective measure of the complexity of a theory. Although this measure is incomputable, it does provide useful information about how a good measure should behave.

The most well known method of evaluating the probability of various hypotheses is Bayesian inference. For a set of competing hypotheses H, and the observed data D, the inferred probability of a particular hypothesis H_i given that data, is given by Bayes' formula

$$P(H_i \mid D) = \frac{P(D \mid H_i) P(H_i)}{P(D)},$$

where

$$P(D) = \sum_i P(D \mid H_i) P(H_i).$$

The probability of the data P(D) can be thought of as a normalising term to ensure that $\sum_i P(H_i \mid D) = 1$. The term $P(H_i)$ is called the *prior* probability, that is, the probability that H_i is true before we have seen any data. The issue of prior probabilities raises some problems. For large enough data sets the prior probabilities become almost irrelevant to the accuracy of the inferred probabilities. Conversely, prior probabilities closer to the actual probabilities require less data to infer accurate posterior probabilities. Since we wish to infer accurate probabilities with as little data as possible it is important to choose a sensible prior distribution. In the case where the set of hypotheses is limited, a reasonable assumption might be to give them equal probabilities. When the set of hypotheses is infinite there seems to be no clear method. An ideal solution would be a universal prior that gives satisfactory results no matter what the real prior is. The search for a universal prior is partly responsible for the birth of algorithmic complexity theory. An introduction to these fields is presented in the next section.

2.2 Complexity, Information, and Probability

This section presents background information on encoding data efficiently. The development of Kolmogorov complexity is outlined, as well as its role in the development of a universal prior. The use of minimum information encoding approaches in machine learning is also presented.

d	f	h	e	e	b	e	h	g	e
h	f	e	h	c	h	e	h	e	b
g	b	h	f	e	e	h	a	f	h
h	h	d	f	b	h	e	g	h	b

Figure 2.1: A sequence of symbols

There are several books and papers that give a more detailed coverage of the following material. A good explanation for those unfamiliar with complexity theory is found in Legg (1995) and Li and Vitanyi (1992). In a series of three technical reports aimed at introducing minimum information encoding to statisticians, Oliver and Hand (1994), Oliver and Baxter (1994), and Baxter and Oliver (1994) present coding and minimum information encoding. Li and Vitanyi (1993) have written an excellent book covering algorithmic complexity.

2.2.1 Coding Data

A single digit (a *bit*) of binary information may be either a 1 or a 0. A binary string is a sequence of bits such as '00101010'. The length of a binary string x is denoted $l(x)$. To store information in a computer, or to communicate

a: 000	b: 001	c: 010	d: 011
e: 100	f: 101	g: 110	h: 111

Figure 2.2: Code dictionary A

```
011 101 111 100 100 001 100 111 110 100
111 101 100 111 010 111 100 111 100 001
110 001 111 101 100 100 111 000 101 111
111 111 011 101 001 111 100 110 111 001
```

Figure 2.3: Encoded sequence A

information from one computer to another, it must be represented in binary. For example, to describe the sequence of symbols shown in Figure 2.1 from the set $S = \{a, b, c, d, e, f, g, h\}$, this information must somehow be represented in binary. A common approach is to construct a *code dictionary* with each symbol represented by a corresponding binary *codeword.* Figure 2.2 gives a possible code dictionary for the symbols in S.

To ensure that the resulting message is unambiguous, the code dictionary should be prefix-free, that is, no codeword may consist of another codeword followed by one or more bits. For example, if the codeword for 'g' is '11' and 'h' is '111', it is impossible to determine whether the string '11111' represents 'gh' or 'hg'. In code dictionary A, there is no ambiguity because the codewords all have the same length. Since the end of each codeword of a prefix-free code can be recognised as such it can be decoded immediately. A code dictionary that is prefix-free is called a *prefix-code* or *instantaneous code.* Figure 2.3 shows the sequence from Figure 2.1 encoded using code dictionary A.

The total length of this message is 120 bits. The code dictionary given in Figure 2.2 is inefficient in that other code dictionaries can encode the sequence in fewer bits. For example, by assigning short codewords to more

a: 0111111	b: 011	c: 1111111	d: 011111
e: 01	f: 0111	g: 01111	h: 0

Figure 2.4: Code dictionary B

frequently occurring symbols we might construct dictionary shown in Figure 2.4.

Although this code dictionary is not prefix-free, it is still uniquely decodable—we know a new codeword has started as soon as we see a '0' bit or once the previous codeword exceeds 7 bits. The sequence in Figure 2.1 encoded with this dictionary gives the message in Figure 2.5, with a total message length of only 109 bits.

The code dictionaries described so far are concerned with encoding from a finite set of symbols, but what about an infinite set of symbols, such as the natural numbers? To send the (base two) number n requires $\log_2 n$ bits, but this is not self-delimiting. One simple prefix-free coding scheme is to first encode n's length as $l(n)$ 1's followed by a 0 and then give n itself (denoted as $1^{l(n)}0n$). This requires a total of $2\log_2 n+1$ bits. But $l(n)$ can be encoded more efficiently in the same manner as n. With this method, n is encoded as

```
011111 0111 0 01 01 011 01 0 01111 01
0 0111 01 0 0111111 0 01 0 01 011
01111 011 0 0111 01 01 0 0111111 0111 0
0 0 011111 0111 011 0 01 01111 0 011
```

Figure 2.5: Encoded sequence B

$1^{l(l(n))}0l(n)n$ with a total length of $\log_2 n+2\log_2(\log_2 n)+1$ bits. This is a recursive problem because $l(l(n))$ could itself be encoded more efficiently. The solution is called log* coding, which can encode n in about $3+\log_2 n+\log_2(\log_2 n)+\log_2(\log_2(\log_2 n))+\ldots$ bits (for details on the actual encoding see Baxter and Oliver 1994, page 12).

The idea of assigning short codewords to more frequent symbols is important. If each element x_i of the set of symbols S has probability of occurring $P(x_i)$, it can be proven that the shortest expected message length is obtained when the length of the codeword for each symbol is equal to $-\log_2(P(x_i))$. The quantity $-\log_2(P(x_i))$ is called the *entropy* in bits of the symbol x_i, which is a measure of the amount of information provided by the symbol. For example, if a bookie gives you a tip that a horse with favourable odds will win a certain race, the tip provides little information (since you would probably predict that horse anyway). If the bookie tips you to a horse with unfavourable odds, the tip provides more information (because you would not normally have predicted that horse).

One coding method that encodes symbols given a probability distribution is the Shannon-Fano code. The set of n symbols is ordered by decreasing probability with probabilities $p_1,\ldots,p_n$. Let $P_r=\sum_{i=1}^{r-1}p_i$, for $r=1,\ldots,n$. Let $E(r)$ be the binary expansion of P_r. The codeword for symbol r is obtained by truncating $E(r)$ at length $l(E(r))$ such that

$$-\log_2 P_r \le l(E(r)) < 1-\log_2 P_r.$$

The Shannon-Fano code achieves the minimum message length on average. The Shannon-Fano code is also a prefix-code. For more information on the Shannon-Fano code see Li and Vitanyi (1993), page 63.

a: 0000001	b: 001	c: 0000000	d: 000001
e: 01	f: 0001	g: 00001	h: 1

Figure 2.6: Code dictionary C

A relation known as the Kraft inequality puts precise constraints on the lengths of the codewords in a prefix-code. If c is a prefix-code with n codewords with lengths $l_1,\ldots,l_n$ then $\sum_n 2^{-l_n} \leq 1$. Conversely, if $l_1,\ldots,l_n$ is a sequence of positive integers that satisfy the inequality, there is a prefix-code which has these codeword lengths. The Kraft inequality stays valid for uniquely decodable codes, which means that every uniquely decodable code can be replaced by a prefix-code without changing the set of codeword lengths. Code dictionary B could therefore be replaced by the following prefix-code shown in Figure 2.6.

A uniquely decodable code is *complete* if the addition of any new codeword results in an ambiguous code. The Kraft inequality must therefore be satisfied with equality for a complete code (Li and Vitanyi 1993, page 70).

So far the discussion has centred on the amount of information in bits that are needed to represent an object from a known set of alternatives. Only one bit is needed to distinguish between two objects, regardless of whether one object is the text of Hamlet and the other of Othello. Algorithmic complexity theory is concerned with the amount of information in an object when the set of objects is universal, that is, the information innate to the object.

2.2.2 Algorithmic Complexity

Occam's Razor requires that we have a measure of the complexity of competing theories in order to choose the simplest one. Algorithmic complexity theory is concerned with finding an absolute measure of the complexity of an object. It is based around the length of description required to completely describe an object with no outside information. If an object can only be described by a very long description, it has a high complexity. The length in bits of the smallest possible description of an object is its *Kolmogorov complexity*. These ideas were developed independently by R.J. Solomonoff (1964), A.N. Kolmogorov (1965), and G.J. Chaitin (1969). In the early 1960's Solomonoff worked on a completely general theory of inductive inference, and "Kolmogorov complexity" was presented as an aside in formulating a universal prior probability distribution. In the mid-1960's, Kolmogorov independently obtained similar results to Solomonoff. However, because Kolmogorov's objectives were purely to obtain an algorithmic measure of the information content in individual objects, the complexity measure came to be known as Kolmogorov complexity. Around the same time, Chaitin also independently proposed similar invariant definitions of complexity. Although both Chaitin and Kolmogorov explored Kolmogorov complexity after Solomonoff, they were unaware of his work until years later.

The Kolmogorov complexity of a string is the size in bits of the smallest program which, using no additional input, computes the string and terminates. Thus, a sequence of 10,000 1's can be represented by a short program such as

```
FOR I := 1 TO 10,000
    PRINT 1
```

This program is approximately log(10,000) bits in size. There are strings however that cannot be calculated by a short program—in this case there is no way to describe the string except literally. The shortest program is to simply print out the entire string itself, and this program has length approximately 10,000 bits. Strings that do not have a shorter description than themselves are *incompressible*. Some strings that appear on the surface to be random may actually have a short description, for example, the infinite sequence representing $\pi = 3.14159265...$ can be produced by a relatively short program. This explanation would lead one to believe that the complexity calculated is heavily dependent on the programming language chosen. After all, programming languages such as LISP favour problems requiring symbolic computation, while languages such as FORTRAN are better suited to numeric tasks. The notion of the information content of an object is only useful if it is a property of the object itself, and not the description language. Fortunately, it has been shown that for any reasonable choice of programming language, the amount of "innate" information in an object is fixed up to an additive constant (think of this constant as the length of an interpreter for one language written in the other language).

We now delve into Kolmogorov complexity more formally. For a string p, $l(p)$ denotes the length (that is, the number of zeros and ones) of p. The machines which decode the descriptions are Turing machines. The binary input strings to the Turing machines are called programs. Let $\mathrm{n}(x)$ be a standard enumeration of all objects x onto the natural numbers. For a particular Turing machine T, a program p is a description of x if, on input p, T outputs $\mathrm{n}(x)$, which we write as $\mathrm{T}(p) = \mathrm{n}(x)$. The complexity of x with respect to machine T is defined as

$$\mathrm{K}_{\mathrm{T}}(x) = \min\{l(p) : \mathrm{T}(p) = \mathrm{n}(x)\},$$

where p is a prefix-free program. This value may change depending on the particular Turing machine used. However, we can construct a Turing machine that for all x assigns a complexity no higher than the minimum complexity returned by any Turing machine, to within a constant.

Universal Turing machines are a subset of Turing machines capable of enumerating all other Turing machines. Thus, any Turing machine T_m may be identified by a number m with respect to a particular universal Turing machine. Let U be the universal Turing machine such that when started on the input string $0^m 1p$, U simulates T_m on input p. If T_m is the Turing machine that returns the minimum complexity for x, U assigns the same complexity plus the $m+1$ bits needed to specify T_m.

$$K_U(x) = K_{T_m}(x) + c_m,$$

where in this case $c_m = m+1$ bits (there are other ways of encoding which Turing machine to select). This is known as the Invariance Theorem (Li and Vitanyi 1993, page 90).

Similarly, for each pair of universal Turing machines that satisfy the Invariance Theorem, U and U′, the complexities are equal up to a fixed constant, for all x:

$$|K_U(x) - K_{U'}(x)| \leq c_{U,U'}.$$

Since the complexity is equal to within a constant, it is customary to fix a single reference machine and define Kolmogorov complexity K(x) with respect to it.

This definition of Kolmogorov complexity should properly be called *prefix Kolmogorov complexity*, due to the requirement that programs executed by the

Turing machines be prefix-free. Non-prefix Kolmogorov complexity $C(x)$ has some properties that make it unsuitable for inductive inference, the major one is that the series $\sum 2^{-C(x)}$ diverges (more on this later). Both functions are asymptotically equal, differing by at most an additive term of $2\log(C(x))$ (Li and Vitanyi 1993, page 173).

Unfortunately, Kolmogorov complexity is incomputable due to the halting problem. The halting problem is that no Turing machine can tell whether any arbitrary Turing machine will halt execution. It may finish tomorrow or it may continue executing forever. As it applies to Kolmogorov complexity, our Turing machine can never tell if it has found the shortest program that computes x (because there might be a shorter program which is still running that may or may not compute x).

2.2.3 The Solomonoff-Levin Distribution—A Universal Prior

Algorithmic complexity was discovered almost as a side issue by Solomonoff in the search for a single universal prior distribution (Solomonoff, 1964). Solomonoff viewed induction as finding a compact description of past observations and predicting future observations in the context of Turing machines. Solomonoff argued that observations past and future can be encoded as a binary sequence, and theories are equated to Turing machines that compute binary sequences starting with the segment which corresponds to past observations. Solomonoff's induction theory is as follows.

Assume the existence of a prior probability distribution described by the probability function P over the space of all binary strings $B=\{0,1\}^*$. Define the function $\mu(x)$ over B by

$$\mu(x)=\sum_{y\in B}\mathrm{P}(xy).$$

Thus, $\mu(x)$ is the probability of a sequence starting with x. Given a data string S representing the past observed data, the task is to predict the next symbol in the sequence. This is expressed in terms of Bayes' formula, where the data D is the initial sequence S, and the hypothesis H_a is that the sequence starts with Sa, that is, H_a explains the past observations S and predicts the next symbol a.

$$\mu(Sa \mid S)=\frac{\mu(S \mid Sa)\mu(Sa)}{\mu(S)}.$$

However, $\mu(S \mid Sa)=1$ for any a, since Sa completely specifies S, giving

$$\mu(Sa \mid S)=\frac{\mu(Sa)}{\mu(S)}.$$

If the prior probability distribution $\mu(x)$ is known, the induction problem is solved; however, the actual prior probability is unknown. To solve this problem, Solomonoff proposed the idea of a *universal prior distribution* that could be used to give results almost as good as if the actual probability distribution were used. Solomonoff succeeded in finding a universal prior, but unfortunately it is incomputable because it uses Kolmogorov complexity. All is not lost because Kolmogorov complexity (and the universal prior) is semi-computable, meaning that there are approximations that are computable.

Solomonoff's original suggestion was that the *a priori* probability P(x) of a binary string x should be the probability that a randomly generated program p generates the string x. The probability of a random program p of length $l(p)$ is $2^{-l(p)}$ making Solomonoff's original prior

$$\mathrm{P}(x)=\sum_{\mathrm{U}(p)=x}2^{-l(p)},$$

where U is a reference universal Turing machine.

The problem with this formulation is that for standard Turing machines the resulting distribution is not a probability distribution, that is, $\sum_x \mathrm{P}(x)$ diverges (i.e., the sum does not converge to 1). Even if one considers only the shortest program computing x rather than all programs, the series diverges. To counteract this problem Solomonoff had to employ normalising terms.

L.A. Levin employed prefix Turing machines to remove the normalising terms (Levin, 1974). The Kraft inequality ensures that for all prefix-free programs $\sum_p 2^{-l(p)} \leq 1$. The *Solomonoff-Levin distribution* is then given by

$$\mathrm{P}(x) = \sum_{\mathrm{U}(p)=x} 2^{-l(p)},$$

where U is the reference prefix-machine. The sum of all P(x) is actually less than 1 since not all programs halt and produce output. A surprising result known as the coding theorem states that K(x) and $-\log(\mathrm{P}(x))$ are equal up to an additive constant (Li and Vitanyi 1993, page 223).

We are interested in how well the universal prior performs in relation to the actual prior. Let M(x) be the universal prior (either taken as P(x) above or $2^{-\mathrm{K}(x)}$) and $\mu(x)$ denote the actual prior. Let S_n denote the expected squared difference between the probability that the (n+1)th symbol is a 0 as given by the universal prior and the actual prior

$$S_n = \sum_{l(x)=n} \mu(x)\left(\frac{\mathrm{M}(x0)}{\mathrm{M}(x)} - \frac{\mu(x0)}{\mu(x)}\right)^2.$$

It can be shown (after a lengthy proof) that the expected squared error at the nth prediction converges to zero faster than $1/n$, so basically the universal prior is a very good approximation to any actual prior (Li and Vitanyi 1993, page 285).

The combination of Bayes' rule and the universal prior allows us to satisfy the dictums of both Occam's razor and Epicurus' principle of multiple explanations. Following Occam's razor, if several programs could generate the string $S0$, the shortest one is preferred (that is, accorded the highest prior probability). Similarly, if the program that generates $S0$ is shorter than the program that generates $S1$, the first would be preferred (that is, predict the next symbol is 0 with higher probability than for the symbol 1). Solomonoff's induction procedure is also in line with the principle of multiple explanations because all hypotheses compatible with the evidence are retained, with the probability distribution over hypotheses modified according to the simplicity of each.

2.2.4 Minimum Information Encoding

The ideas from Kolmogorov complexity and Solomonoff's inference procedure served as inspiration in the development of two related principles for inference; Rissanen's minimum description length principle (MDL), and Wallace's minimum message length principle (MML). Both of these principles are similar; Baxter and Oliver (1994) provide a description of their similarities and differences. The approach described here is closest to MDL and can be considered a computable approximation to Solomonoff's induction procedure. An intuitive explanation of the principle is as follows.

> Minimum Description Length Principle: *the best hypothesis to explain a set of data is the one that minimises the sum of*
>
> 1) *the length in bits of the description of the theory; and*

2) *the length in bits of the data when encoded with the help of the theory.*

There are problems to be addressed in the selection of competing hypotheses. In general, the more complex a hypothesis, the better it fits the observed data. At one extreme, a complex description of the hypothesis H may describe the data completely. However, this hypothesis is vulnerable to errors in data measurement and statistical irregularities of the observed data, meaning it is unlikely to predict new data well—this is known as *overfitting*. At the other extreme is a trivial hypothesis which does not describe the data at all (and offers no predictions). The MDL principle provides a way to find a balance between the simplicity of the hypothesis and its accuracy in describing and predicting the data.

Like Solomonoff's procedure, the MDL principle can be derived from Bayes' rule with the help of Kolmogorov complexity. When using Bayes' rule we are interested in maximising the probability of the hypothesis H given the data D. First we take the negative log of Bayes' rule

$$-\log_2 P(H \mid D) = -\log_2 P(D \mid H) - \log_2 P(H) + \log_2 P(D).$$

When comparing competing hypotheses, the data will not change, making $\log_2 P(D)$ a constant factor that can be ignored. We are therefore concerned with minimising the term $-\log_2 P(H \mid D)$, which is equivalent to minimising

$$-\log_2 P(D \mid H) - \log_2 P(H).$$

Substituting in the universal prior $2^{-K(x)}$, we obtain

$$K(H \mid D) = K(H) + K(D \mid H).$$

We therefore seek to minimise the length of the shortest encoding of the hypothesis H and of the data D with the help of hypothesis H.

Because Kolmogorov complexity is incomputable, the K function must be replaced with some computable approximation for use in practical applications. A common method is to use a standard encoding such as the log* code to provide a simple upper approximation.

To outline a simple example of the MDL principle, consider inferring the distribution of the heights of a group of people. The data consists of some number of height measurements. A hypothesis consists of a specification of the type of distribution (such as normal or bimodal) along with any required parameters (such as mean and standard deviation stated to some precision). The data can be encoded by assigning short codewords to height measurements with a high probability (according to the hypothesis), and longer codewords to height measurements with a low probability. A good hypothesis will result in a short encoding of the data. Stating the hypothesis parameters to a higher level of precision may result in a shorter encoding of the data but a longer encoding of the hypothesis. Since the average heights of women and men are different, a hypothesis representing a bimodal distribution may result in a shorter encoding of the data, but the encoding of the hypothesis requires more parameters than a hypothesis utilising a unimodal distribution. The MDL method will find a trade-off between these factors. The hypothesis that results in the shortest total encoding of both the hypothesis and the data is defined as the best.

The MDL principle has been used successfully in many applications, particularly in the fields of machine learning and data modelling. Quinlan and Rivest (1989) describe the use of MDL in constructing decision trees. The task is to take a training dataset and infer a set of questions that will yield each

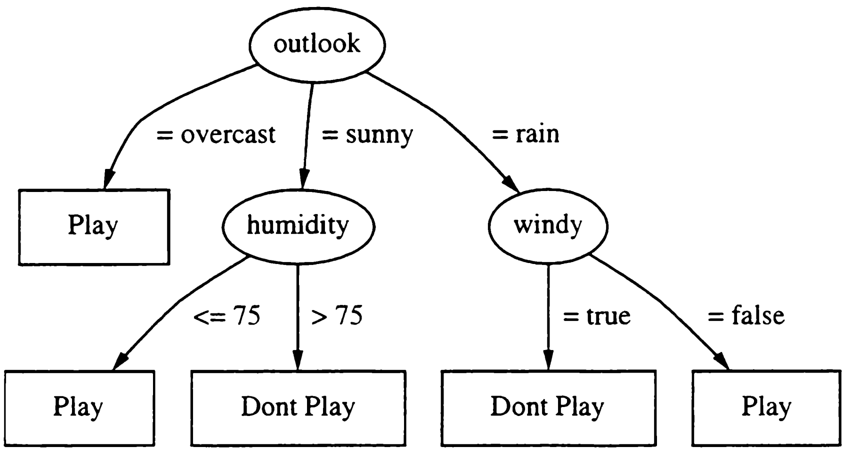

Figure 2.7: An example decision tree

instance's classification. These questions can be represented as a tree. Figure 2.7 shows a decision tree for deciding whether to play golf based on the weather. Quinlan and Rivest describe a method for encoding decision trees, and for producing good trees guided by the MDL principle. The MDL inspired trees were compared to decision trees produced by one of the best alternative methods and found to be smaller on average with roughly the same classification accuracy.

Another example application of MDL is in the estimation of linear regression models. The task is to fit a polynomial to a set of n data points. Although is it always possible to fit a $n-1$ degree polynomial exactly to the data, such a solution provides little general information about the data. Polynomials of lower degree may provide more information about general features of the data but fit the data values themselves less accurately. This sort of trade-off is characteristic of the domains where MDL has found successful application. Legg (1995) describes a system called LME that uses the MDL principles to infer linear regression models that best fit the data. Encoding the data with respect to the model is achieved by encoding the error in the model's prediction for each data value. The errors are encoded to a set accuracy level that determines the accuracy of the model. (A model that makes predictions more accurate than the error resolution will be longer to encode but will not give a reduction in the data encoding.) Encoding the model consists of specifying the number of coefficients, then the coefficients themselves. Each coefficient has an accuracy level (which determines the number of significant digits), followed by the coefficient digits. LME and other minimum information based methods have been evaluated on several regression problems and have excellent performance (Legg, 1995).

2.3 Conclusions

A major difficulty with applying Occam's razor to the task of induction is in determining the simplicity of competing hypotheses. Algorithmic complexity defines the simplicity of any object as the length of the shortest program describing that object. The description of any type of object can be expressed within this framework. The defined complexity gives a measure of the information present in an object independent of description language. While the complexity of an object is incomputable, there are computable approximations. The ideas from algorithmic complexity have been used successfully in inductive inference. The MDL principle is a direct result of interpreting Bayes' rule within algorithmic complexity, and has been applied in a range of applications with good results. The success of the MDL method suggests that a complexity-based approach may also be useful in solving problems in the design of similarity functions.

Chapter 3

Issues and Current Treatment

This chapter examines issues related to designing similarity functions and describes current treatment of these issues by the machine learning community. Similarity functions form the core of instance-based machine learning algorithms. This chapter provides an introduction to instance-based learning and discusses in detail how the issues of similarity function design are currently dealt with. Because geometric domains provided initial motivation for this thesis, an example of this type of application is presented to give a concrete feel for the types of problems facing designers of similarity functions. This example also highlights the *ad hoc* treatment these problems currently receive.

3.1 Weather Case Retrieval

Large repositories of historical weather data have the potential to assist weather forecasters and other meteorologists in their decision making. This weather data is practically unusable without a mechanism for retrieving past situations that are relevant to the current problem. A weather forecaster is unlikely to recall the exact date and time of relevant weather situations. Rather, the forecaster could specify meteorological features of interest and then have similar situations retrieved from the database. Such a system acts as a "memory amplifier", which offers fast access to historical situations of interest. Jones and Roydhouse (1993) describe such a system called MetVUW.

MetVUW combines data such as laser disc video, text descriptions, satellite images, and numeric information on temperature, humidity, and wind speed. As well as allowing retrieval by date and time, MetVUW provides a case retrieval facility that searches for matches based on high-level meteorological content. Users of MetVUW enter a graphical query that represents the primary meteorological features of interest, such as high and low-pressure systems. Queries are compared to *annotations*[1] of the historical situations using a conventional relational database to quickly build a list of potentially relevant cases. Each retrieved case then undergoes a knowledge intensive similarity assessment. Cases that are sufficiently similar are presented to the user.

The following components are considered when comparing cases:

- The shape of high and low-pressure systems are compared in different ways. For high-pressure systems the orientation of ridges is important.

[1]Annotations have the same representation as queries, and are semi-automatically extracted from the raw data offline.

The shape is represented in a conventional manner as a number of convergent axes. For low-pressure systems only the size of the system is relevant.

- Clusters of pressure systems are created and evolve by the splitting and joining of separate small systems. It is important to match these clusters in an intelligent manner. Clusters are represented as hierarchies of enclosure. Query retrieval is based only on the outermost cluster, while detailed examination considers the internal structure of the tree.
- The change of weather patterns over time is often important. Assuming changes between sequential images are small enough, it is easy to identify matching features in successive images. From this information a causal graph is created, which is used to compare movement of features over time.
- Spatial relationships between features are another important component, particularly when one feature has a causal influence over others. For example, a *blocking high* can impede eastward progress of a low-pressure feature to its west.
- There are other important components such as differences in pressure minima and maxima, and the difference in pressure at the centre and boundary of a system.

During detailed matching the average positional displacement between features in the query and corresponding features in the retrieved example is calculated, along with deviations of individual features from the average displacement. The average displacement is used to penalise large north/south translations of the image as a whole, while individual displacements measure the degree of match in the arrangement of features.

It is not immediately obvious how to evaluate weather map similarity based on any of the components above, neither is it clear how to combine the results of multiple components. Jones and Roydhouse (1993) do not describe how the individual components of similarity are combined to arrive at an overall similarity, but that they are experimenting with different methods. The types of information being considered are very different so the lack of an obvious overall function is understandable. A general procedure for deriving similarity measures for these components and combining them could simplify the design of this and other applications that involve measuring similarity between complex examples.

3.2 Instance-based Learning

Instance-based learners classify by comparing the unclassified instance to a database of preclassified instances. The similarity between the new instance and those in the database is used to predict the new instance's class. The assumption is that similar instances will have similar classifications. The important issue is how to define "similar instance" and "similar classification." Typically an instance-based learner has a similarity function which determines how alike instances are, and a classification function which specifies how instance similarities yield a final classification for the new instance. For example, a simple classification function returns the class of the single closest training instance. Once an instance has been classified, it is moved into the database along with its correct classification (for a simple instance-based learner, this is all the "learning" that occurs). Thus, if an incorrectly classified instance (or more importantly a similar instance) is re-presented it is more likely to be classified correctly.

Instance-based learning has been applied successfully to many domains such as letter recognition (Fogarty, 1992), identifying zone types in document images (Inglis and Witten, 1995), predicting word pronunciation (Stanfill and Waltz, 1986; Cost and Salzberg, 1993; Lowe, 1993), and predicting the folded structure of proteins (Cost and Salzberg, 1993). Although rarer in the literature, instance-based learners are also employed in fielded applications. Jabbour *et al.* (1988), describe a system called ALFA, the Automated Load Forecasting Assistant, which predicts the short-term demand for electricity at Niagara Mohawk Power Corporation. Each instance consists of 12 weather-related measurements, the time and date measurements were made, and power demand at that time. The instance database contains hourly entries for the last 15 years. ALFA retrieves the most similar instances to the current situation and uses these along with information about special events such as public holidays to predict the electric load. This system was so successful that a similar system was implemented to predict natural gas demand (Jabbour and Meyer, 1989).

A typical approach when dealing with geometric domains (such as comparing weather maps) is to reduce the image to some number of attribute values and use these as the basis for comparison. For example, Hastie and Tibshirani (1995) report success using their instance-based learner to identify land usage from attributes automatically extracted from satellite photographs. Bankert and Aha (1995) experiment with several algorithms for automatically identifying cloud patterns in satellite images. Initially, values for 98 different attributes are automatically computed from the images. A simple instance-based algorithm is used to search for the best performing subset of these attributes, to eliminate redundant or irrelevant attributes. A subset of 9 attributes was found to perform particularly well. With these attributes the instance-based algorithm gave better accuracy than a decision tree algorithm and a probabilistic neural network classifier. Good performance can be achieved on this type of problem as long as suitable attributes are calculated. It is often difficult to determine what attributes should be considered, which is why Bankert and Aha take the approach of computing many different attributes and determining the irrelevant ones later.

The simplest instance-based learners are nearest neighbour (NN) algorithms (Fix and Hodges, 1951; 1952). They use a domain-specific distance function to retrieve the most similar instance from a database and present the class of this instance as the classification for the new instance. Cover and Hart (1967) found that in the large sample case, the probability of error of the nearest neighbour rule is at most twice the Bayes probability of error. This can be interpreted as meaning that half the classification information in an infinite sample set is contained in the nearest neighbour.

Standard nearest neighbour algorithms can be generalised to k-nearest neighbour (k-NN) algorithms (Fix and Hodges 1951; 1952). The k closest neighbours are retrieved, and whichever class is predominant among them is given as the class of the new instance. Thus, a standard nearest neighbour classifier is a k-NN classifier for $k = 1$. If the number of instances in the database is large, it makes sense to use more than the single nearest neighbour, but k should be small enough that the chosen instances are close enough to the unclassified instance to give an accurate estimate of its class. In the degenerate case where k equals the number of instances in the database, the same class will be predicted regardless of where the unclassified instance is in the feature space. Choosing a suitable value of k for a particular dataset must be done carefully. For instance, k should be much smaller than the number of training instances in the smallest class. Dasarathy (1990) suggests developing an explicit method for selecting the optimum neighbourhood size for a particular training set. Fix and Hodges (1951) showed that when k and the number of instances n tend to infinity such that $k/n \to 0$, the probability of error approaches the Bayes probability of error.

While the only requirement of Fix and Hodges (1951; 1952) was that the distance function be able to specify which of two points is closer to a third, the particular function can have a large effect on the learning rate. For a simple example several functions were examined, such as ordinary Euclidean distance, choosing the maximum difference over all attributes, and attribute weighted combinations. They conclude that a distance function must be chosen carefully for the particular domain, to minimise the number of instances needed to reach the desired performance level.

When Biberman (1994) studied similarity functions in relation to psychological notions of similarity, he found that many similarity functions are not sensitive to the domain at all. The similarity between two objects is usually defined by the objects rather than the domain, and hence is the same for all domains. In many domains this is not the case. For example, the similarity of "red" and "green" will be different when classifying apples than when classifying cucumbers, even though the colour attribute may be of equal importance overall in both domains. Biberman also notes that many similarity functions assume the similarity of two equal values is the same for all values. Humans intuitively take two instances that share some uncommon value as more similar than two instances that share a common value. For example, we perceive two identical twins as more similar to each other than two identical cars. This is because we know that there are many identical cars of a particular model, but only two identical twins of a particular "model."

Designing a similarity function that is appropriate for a particular domain is not easy. The issues that must be addressed by the similarity function of instance-based learning algorithms are:

- Measures for different attribute types must be combined to determine overall similarity. Early use of NN algorithms involved numeric attributes only; however in practical applications attributes may be of different types. The most common other attribute type in current use is symbolic.
- Some attributes may be more relevant to discrimination than others, and these attributes should be given more importance in the distance function. The weighting scheme must be applied consistently to different attribute data types to avoid biasing the measure in favour of particular data types.
- Some instances may be better discriminators than others, and it makes sense to prefer the prediction of a good discriminator over that of a poor discriminator. Typically, instances that are near class boundaries are good discriminators. An instance may be a poor discriminator because it is well away from a class boundary, or it may be the result of erroneous data.
- Many real world datasets contain instances with missing attribute values, and these must be handled sensibly, to allow the most use of the information that is present.

The following sections discuss these issues in detail and outline the approaches currently employed in dealing with them.

3.3 Different Attribute Types

In early applications of NN discrimination, datasets consisted primarily of numeric attributes. Research during this period focused upon how to select the number of neighbours for comparison, how to deal with missing values, and how to avoid problems of attribute scaling. Machine learning as a field began to develop when researchers wanted to apply this technology to a wider range of real world problems. It soon became apparent that the representation of many problems requires more than numeric data. Today machine learning classification schemes typically deal with attributes of two basic types: numeric and symbolic; other attribute types are often transformed into these basic types (Almuallim *et. al.*, 1995). This raises the problems of how to measure similarity over a different type of attribute (such as symbolic), and how to meaningfully combine the similarity of several different attributes.

Very little classification research has been directed towards making use of domain specific knowledge. There may be a few explanations for this. In some cases there is little or no domain knowledge available. A common approach when evaluating a machine learning algorithm is to simply run it on some standard datasets (many of which have little documentation on the meaning of their attributes). While a good classification algorithm might perform well with minimum domain knowledge, it is also sensible to utilise domain information when it is available. Cleary *et al.* (1996a) promote the storage of "metadata" alongside datasets to encourage automated domain customisation, although as yet no machine learning schemes are able to make use of this metadata. Another explanation for the rarity of classifiers using domain specific similarity functions is that there is no general method for tailoring functions to the domain. When a domain specific classifier is developed it is (by definition) rarely suitable for use on other domains.

This section presents similarity functions from the literature. Often these functions handle numeric or symbolic attributes only. The following section discusses research on functions that combine measures for different attribute types.

3.3.1 Numeric Distance Functions

Comparing numeric values is relatively easy since there exists a simple operation that takes two values and returns a distance between them: subtraction. A variety of functions have been proposed when multiple numeric attributes are to be considered. Fix and Hodges (1952) experiment with the following two general distance functions:

$$\text{Euclidean}(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

$$\text{Maximum}(x, y) = \max_{i=1}^{n} |x_i - y_i|$$

where an instance x is a vector of n attribute values $x_1, \ldots, x_n$.

The Euclidean distance function measures the distance between points in a straight line. A similar distance function measures distance between points as the sum of distances along each axis (e.g., travelling between two addresses along city streets):

$$\text{CityBlock}(x, y) = \sum_{i=1}^{n} |x_i - y_i| \, .$$

A problem common to all these functions is that they are dependent on the attribute scale—attributes that have a large range tend to outweigh attributes that have a small range. An early attempt to reduce the sensitivity to attribute scales is reported by Devroye (1978). This research develops the notion of empirical distance, and describes a k-NN algorithm that is independent of monotone transformations of the domain attributes. Essentially, continuous attributes are transformed into ordered attributes. The distance is

$$\text{Empirical}(x, y) = \max_{i=1}^{n} \{ f(x_i, y_i) \},$$

where $f(x_i, y_i)$ is the absolute difference in the order of x_i and y_i.

An alternative approach to the problem of attribute scaling is to normalise attribute values. Aha, Kibler and Albert (1991) describe an instance based learner called IB1 which uses a general purpose distance function to avoid the requirement to custom-derive a function for each domain, allowing the nearest neighbour technique to be used in practical applications. The similarity function used by IB1 is a Euclidean distance measure, with numeric attributes normalised to ensure all attributes have equal relevance.

The problem of measuring similarity between ordinary numerical attributes appears to be fairly well solved, although there are a few different methods used to combine measures of multiple attributes. Euclidean distance is the most popular method in this respect. However, similarity measures have not been reported for more specialised numeric attributes, such as those that are modulo (such as the distance around the perimeter of a circle, or between times of the day).

3.3.2 Symbolic Distance Functions

Symbolic attributes are more difficult to deal with than numeric attributes because there is no natural notion of distance. IB1 (Aha, Kibler and Albert, 1991) uses a simple function, called the *overlap metric*, where the similarity between values is 1 if they are the same, and 0 otherwise. For multiple attributes, the value returned is a count of which attributes have matching values.

Stanfill and Waltz (1986) describe a measure for symbolic attributes that is considerably more sophisticated. Their Value Difference Metric (VDM) assumes that symbols that predict the same class are more similar than symbols that predict different classes. This allows a numerical distance between two symbols to be calculated. Their metric for the similarity of two instances is

$$\mathrm{VDM}(x, y) = \sum_{i=1}^{n} weight(x_i) f(x_i, y_i).$$

Here $f(x_i, y_i)$ is a numerical measure of the difference between the predictions of two symbols, given by

$$f(x_i, y_i) = \sum_{c \in C} \left(\mathrm{P}(c \mid x_i) - \mathrm{P}(c \mid y_i)\right)^2,$$

where C is the set of classes and $\mathrm{P}(c \mid x_i)$ is the probability of class c occurring in the subset of training data where the symbol x_i occurs. $weight(x_i)$ is a measure of how predictive symbol x_i is;

$$weight(x_i) = \sqrt{\sum_{c \in C} \mathrm{P}(c \mid x_i)^2}.$$

Thus, the more skewed a symbol's prediction distribution is, the higher its weighting. Uniformly distributed symbols with tell us little about the class, so have low weighting. This weighting is based on individual symbols as opposed to the more commonly used instance weighting and attribute weighting. Symbol weighting is often avoided as it makes the similarity function non-symmetric (Cost and Salzberg 1993). However, Tversky (1977) believes that in some domains asymmetry is justified. Weighting schemes are discussed in later sections.

Lee (1994) describes a related similarity function for symbolic attributes. Where the VDM uses probabilities, Lee's function uses a measure of information content. The similarity between two symbols is

$$f(x_i, y_i) = 1 - \left| \sum_{c \in C} \frac{\mathrm{P}(c \mid x_i)\log\left(\frac{\mathrm{P}(c \mid x_i)}{\mathrm{P}(c)}\right) - \mathrm{P}(c \mid y_i)\log\left(\frac{\mathrm{P}(c \mid y_i)}{\mathrm{P}(c)}\right)}{\sum_{z_i} \mathrm{P}(z_i)\mathrm{P}(c \mid z_i)\log\left(\frac{\mathrm{P}(c \mid z_i)}{\mathrm{P}(c)}\right)} \right| .$$

In spite of its apparent complexity, the heart of this similarity function is similar to the VDM. Normalising factors are employed to keep the measure in the range 0 to 1, and the central value is subtracted from 1 to convert from dissimilarity to similarity (both of these modifications are for compatibility with Lee's functions for other attribute types).

Biberman (1994) proposes an alternative measure for symbolic attributes, called *context similarity*. This measure is based on the assumption that similarity is dependent on context, which consists of the set of instances given and the domain being considered. Biberman notes that a common failing of simple symbolic measures such as the overlap metric is that they do not allow different degrees of similarity between different values. For example, the colour red is considered equally dissimilar to the colour orange as it is to the colour blue. Between each pair of values there should be a unique parameter representing their similarity. How to obtain a value for these parameters is not discussed. The primary improvement of the VDM beyond the overlap metric is that it incorporates this idea of variable differences between symbolic values, their differences calculated from the symbol associations with each class.

Biberman first defines a similarity measure for equal symbolic values. This will vary from value to value, which implies that the triangle inequality will not hold. The similarity between two identical infrequent values is taken as higher than between two identical frequent values. To ensure that the similarity measure is sensitive to the classification concept, the measure is scaled by the variance of the value's occurrence rate in the different classifications. If the value is irrelevant the occurrence rate should be equal across classifications. The similarity of two equal attribute values x_i, y_i is

$$\mathrm{s}_{eq}(x_i, y_i) = (1 - \mathrm{P}(x_i))\mathrm{var}(x_i, C),$$

where $\mathrm{P}(x_i)$ is the frequency of occurrence of x_i, and $\mathrm{var}(x_i, C)$ is the variance of $\mathrm{P}(x_i)$ over all classifications C.

Having defined a measure for equal values, Biberman defines the *effect* that two non-equal values have, using the similarity of matching values in other attributes:

$$effect(x_i, y_i) = \sum_{x_j = y_j} \frac{s_{eq}(x_j, y_j)}{n}.$$

The *effect* function gives an estimate of how well two instances match based on the number of values that match as well as the degree of similarity of the matching values themselves. This is incorporated into a measure for non-equal values with respect to a classification c:

$$\begin{aligned} \mathrm{s}_{dif}(x_i, y_i, c) = & \sum_{w \in S^{x_i}, w \in c, z \in S^{y_i}, z \in c} \frac{effect(w, z)}{n_1} \\ & - \sum_{w \in S^{x_i}, w \notin c, z \in S^{y_i}, z \in c} \frac{effect(w, z)}{n_2} \\ & - \sum_{w \in S^{x_i}, w \in c, z \in S^{y_i}, z \notin c} \frac{effect(w, z)}{n_3}, \end{aligned}$$

where S^{x_i} is the set of instances that have the value x_i, and n_1, n_2, n_3 are the number of instances in each of the summations. The first summation encodes the intuition that if two instances belonging to the same class share many values, the values in which they differ are also similar with respect to the classification. The second and third summations encode the intuition that if two instances belong to different classes and share many values, the difference in classification must be due to the values in which they differ.

Using the similarity measure defined for equal and non-equal values, the similarity between two instances over all attributes is defined as

$$\text{ContextSimilarity}(x, y) = \left(\sum_{i=1}^{n} \frac{s_{val}(x_i, y_i)}{n} \right)^r,$$

where r is an odd integer greater than one. Raising the sum to a power effectively increases the similarity of similar instances, and pushes dissimilar instances further away. Biberman states that the exact value of r does not affect his results. Biberman found that a nearest neighbour classifier using his similarity function performed better than when using a number of other measures, although not significantly better than Stanfill and Waltz's VDM.

These approaches highlight the inherent difficulty of measuring similarity between symbolic attributes. The overlap metric is essentially the simplest method that could be devised without thought as to what similarity means in symbolic attributes. Stanfill and Waltz and Biberman have constructed functions that incorporate some intuitive ideas about how similarity between symbols should be measured. That both Biberman's context similarity and Stanfill and Waltz's VDM perform quite well suggests that even though their approaches are quite different, they are capturing some of the right kinds of information.

3.3.3 Coherent Treatment

Little research has addressed the problems caused by combining several fundamentally different similarity measures. Wilson and Martinez (1997) and Ting (1995) describe methods to treat continuous attributes and symbolic attributes using a single uniform metric. Ting notes that the current practice of simply combining different attribute measures is analogous to summing numbers of different units—the result may be a bigger number but it is almost meaningless. When additional inconsistent processes are incorporated into the measure, undesirable effects increase. A similarity measure that combines incompatible components for continuous and symbolic attributes is bad enough. Adding in different methods for dealing with missing values for each will make matters worse.

Ting's (1995) solution is to measure both continuous and symbolic attributes using the same function. To achieve this, continuous data is transformed to symbolic form. Several methods for discretisation are considered. The method employed is Fayyad and Irani's (1993) discretisation, which is based on the MDL principle. In this method, a cut point is chosen which maximises the information gain (i.e. the entropy of the resulting sets is minimised). Cut points are chosen recursively until there is no information gain.

Discretisation turns out to have a couple of benefits. First, if a continuous attribute is irrelevant or contains high levels of noisy data, discretisation will place all values into a single set. These attributes are effectively ignored during classification, and the discretisation process can be viewed as a type of attribute weighting. This in itself can account for a large increase in classification accuracy. Second, in domains with a high level of noisy data, discretisation results in noise reduction. This is because discretisation increases the granularity of the instance space. Without discretisation, if a noisy instance is closest to the test instance it will be used in the classification. After discretisation all instances within the same hypercube as the noisy instance will be the same distance, and classification is effectively *k*-NN (giving fewer noise-induced misclassifications). In Ting's experiments, discretisation more than doubled the number of cases where the test instance class was determined by more than one training instance. Third, discretisation provides a type of local adaptation. In areas of the instance space where there

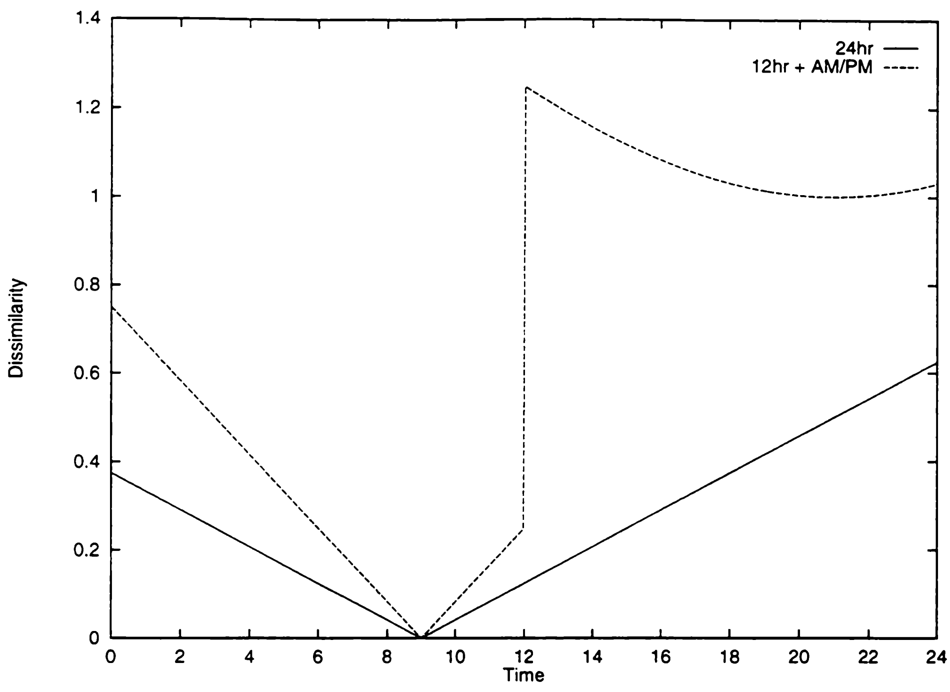

Figure 3.1: IB1 dissimilarity for two representations of time of day

are small clusters of instances, discretisation will create small partitions. In areas of low instance density the partitions will be larger. This variable partition sizing also aids classification, particularly in areas of higher instance density.

The side effects introduced by discretisation during preprocessing result in increased classification accuracy, but obscure the issue of whether a uniform metric for continuous and symbolic attributes is beneficial in its own right. Ting illuminates the issue by examining classification accuracy on two artificial domains.

Ting's first domain models ultra-violet radiation (UV) as a function of time of the day. If the time of the day is between 11AM and 3:30PM the level of UV is high, and at other times it is low. This domain is represented by two datasets: UV1 represents the time of the day as one symbolic attribute (AM or PM) and one continuous attribute (the hour ranging from zero to twelve); UV2 represents the time as a single continuous attribute ranging from zero to twenty four.

Consider the behaviour of IB1's distance function for the time of day representations used by the UV1 and UV2 datasets. Figure 3.1 depicts the dissimilarity between the time 0900 (9AM) and other times of the day. As expected, dissimilarity is at a minimum when the time is 0900. For the 24-hour representation, dissimilarity varies linearly as the difference in time increases. Note however that 2100 is regarded as closer to 0900 than 2300, even though the time interval between 2300 and 0900 is 2 hours smaller than that from 0900 to 2100. Measuring dissimilarity using the 12-hour representation is more interesting. During the AM period the function behaves similarly to the function using 24-hour representation. When the PM period begins there is a sudden jump in dissimilarity, then a decrease until 9PM when it rises again. This function would certainly yield an interesting ordering of neighbours during classification. However these effects will not cause a difference in classification (for our hypothetical test instance at 9AM) unless there are no training instances in the interval between 9AM and 12PM.

Figure 3.2 shows IB1's error rate for the UV1 and UV2 datasets and for a modified IB1 on the UV2 dataset (labelled UV2*). Each data point is the average error over 100 randomly generated training sets. The same test set of 1000 instances was used in all cases. Two-tailed, paired *t*-tests were carried out at the 95% confidence level. When the number of training instances is between 10 and 40 the 24-hour time representation (UV2) gives a significantly lower error rate than the 12-hour representation (UV1). As predicted, the artefacts introduced by the 12-hour representation are detrimental to the classification accuracy when the number of training

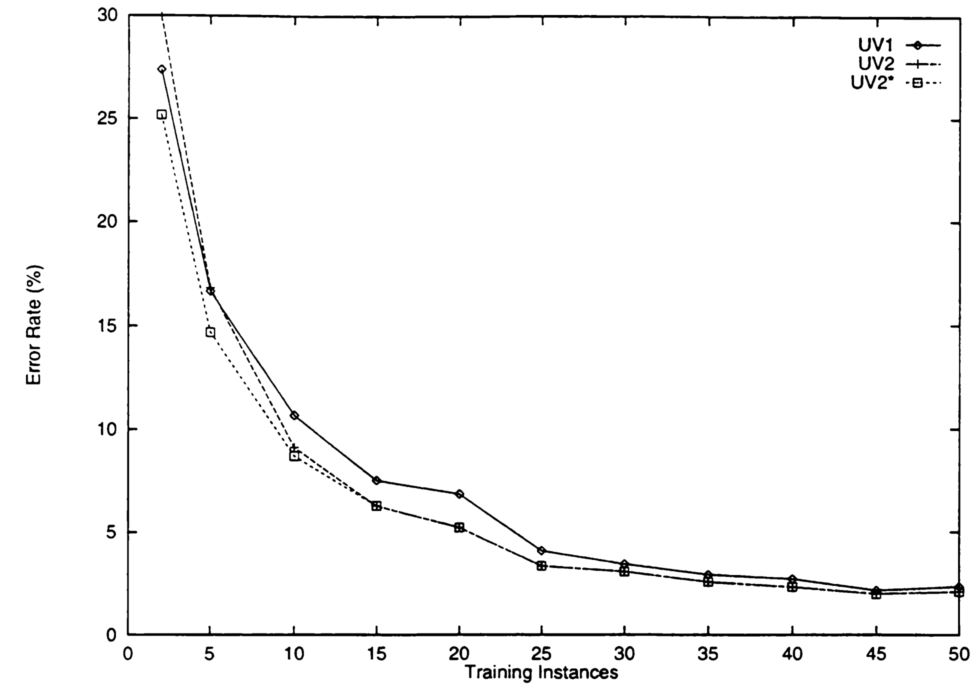

Figure 3.2: IB1 error rate for UV1 and UV2 datasets, and modified IB1 error rate for UV2 dataset

instances is low. Interestingly the error rates are about equal for 5 training instances, primarily because by chance the class boundary lies near the 12-hour border, and also because the UV-HIGH class is small (IB1 tends to overestimate its size). The curve labelled UV2* shows the error rate for a version of IB1 which assumes the time of day is modulo in nature (i.e. the interval between two times x and y is taken as $\min\{|x-y|, 24-|x-y|\}$), on the UV2 dataset. The UV2* error rate remains significantly lower than the UV1 error rate even when there are only two training instances. This simple experiment suggests that data representation should match the underlying domain as much as possible, and that the similarity function should also make use of specific knowledge about the domain.

Ting compares IB1, IB1*, and IB1-MVDM* on the UV1 dataset, and reports that automatic discretisation significantly degrades classification performance[2]. When the data is discretised using the actual cut-points in the domain, IB1* and IB1-MVDM* perform significantly better than IB1. Ting claims this shows that using a single uniform metric is better than using two separate metrics, but this does not follow. IB1* and IB1-MVDM* perform better when actual cut-points are given because this is providing information that can not be discovered from the training data alone. In any case where two training instances of different class are either side of a concept boundary, the only conclusion to be drawn is that the boundary lies somewhere between the two instances—there is no way of knowing exactly where the boundary lies. In addition, automatic discretisation performs more poorly than IB1 because time ordering information is lost in the conversion from a continuous to symbolic attribute.

In the UV datasets, concept boundaries are orthogonal with respect to the axes, and an ideal discretisation can exactly represent the concept with no loss of information. The second artificial domain Ting describes has a non-orthogonal class boundary. The data consists of two continuous attributes named X and Y. Where X+Y is greater than 1, instances fall in one class, otherwise the second class. Ting found that discretisation in this type of domain will result in inaccurately described concept boundaries. IB1 performed significantly better than IB1* and IB1-MVDM* in this domain (except when noise was introduced).

In summary, discretisation at the outset provides a method for treating symbolic and continuous attributes with a single uniform metric. However,

[2] Ting defines IB1* as the standard IB1 classifier (using the overlap metric) which discretises continuous data to symbolic. IB1-MVDM* uses the MVDM metric (Cost and Salzberg 1993) for symbolic attributes.

the conversion from continuous to symbolic data results in a loss of information. In noise-free domains this reduces classification accuracy. In noisy domains accuracy improves because multiple neighbours are considered more often, allowing noisy instances to be outvoted. This suggests that a better solution is to improve the noise tolerance by means that do not degrade performance in noise-free situations.

Wilson and Martinez (1997) extend Stanfill and Waltz's VDM to handle numeric attributes. The difficulty with applying the VDM to numeric attributes is that it requires an estimate of the probability of a class given a particular attribute value. For symbolic attributes, this probability can be estimated with simple counts. Wilson and Martinez examine two methods for estimating class probabilities given a numeric attribute value. The first method, IVDM, discretises a copy of the numeric attribute into equal-width bins. Class probabilities for each bin are calculated, and these probabilities are interpolated when a probability for a specific value are required. The second method, WVDM, slides a "window" along the range of each attribute, calculating class probabilities at each point from the class frequencies in the training instances that fall within the window. Wilson and Martinez found that both methods perform considerably better than Euclidean distance; in addition IVDM performed better than Ting's (1995) IB1-MVDM*, although WVDM performed slightly worse than IB1-MVDM*. Wilson and Martinez' work is a principled approach to treating numeric and symbolic attributes coherently with a common similarity function, however there is no provision for extending the similarity measure to other attribute types.

3.4 Attribute Importance

When a dataset is created, it is seldom known exactly which attributes are relevant with respect to the class; indeed some attributes may be irrelevant altogether. Schemes that attempt to discover the relevance of different attributes and weight them accordingly are called attribute-weighting algorithms.

The attribute-weighting scheme employed by Salzberg's (1991) nested generalised exemplar learner called EACH is simple. Each attribute is assigned a scale factor. When an incorrect prediction is made, the weights for all features that match are decreased by an (arbitrary) factor making these features less important, and those for features that do not match are increased by the same factor making them more important. Unfortunately, this scheme suffers from two main problems. In tasks where one class occurs more frequently than others, attribute weights are biased towards that class to the exclusion of others. A second problem (which applies specifically to EACH) is that the distance to EACH's generalised instances is often zero, which renders the attribute weighting ineffective.

Aha (1992) describes an addition to the IBn series which handles irrelevant and novel (that is, the introduction of previously unseen) attributes. IB4 maintains a set of attribute weights for each class. The similarity function used is:

$$\text{IB4Similarity}(c, x, y) = -\sqrt{\sum_{i=1}^{n} weight_{c_i}^2 \, f(x_i, y_i)},$$

where $weight_{c_i}$ is the weight of attribute i for class c, given by:

$$weight_{c_i} = \max\left(\frac{CumulativeWeight_{c_i}}{WeightNormalizer_{c_i}} - 0.5, 0 \right).$$

After an instance has been classified the attribute weights for the class c are adjusted in the following manner: For each attribute i, if the predicted classification was correct, increase $CumulativeWeight_{c_i}$ proportional to $1 - f(x_i, y_i)$, otherwise increase $CumulativeWeight_{c_i}$ proportional to $f(x_i, y_i)$. Increment $WeightNormalizer_{c_i}$ by the maximum $CumulativeWeight$ increment. The proportion is set lower for frequently occurring classes than for less frequently occurring classes to ensure that all classes have their weight settings adjusted at the same overall rate.

The above attribute-weighting schemes assign global weights to the attributes—that is, an attribute cannot be regarded as relevant only in some areas of the domain space. Hastie and Tibshirani (1995) note that the idea of local adaptation of the distance function has received little research attention. They describe a method called discriminant adaptive nearest neighbour (DANN) classification that determines local decision boundaries and then shrinks the space orthogonal to the boundaries, and elongates the space parallel to the boundaries. This procedure can be thought of as determining the relative attribute importances on a local scale. A further enhancement combines local discriminant information to perform global dimension reduction.

Wettschereck and Aha (1995) present a comparison of several attribute-weighting methods. Attribute-weighting schemes are categorised with respect to the following five dimensions:

Feedback: Weighting methods that use classification results to continually adjust attribute weights are called *feedback* methods. Those schemes that calculate attribute weights in a one-off procedure are called *ignorant* methods. The weighting schemes used by EACH and IB4 are feedback methods, while the weighting used by DANN is an ignorant method.

Weight Space: Some weighting methods perform feature selection, where attributes are either used or completely ignored. Other weighting methods allow continuous attribute weights to determine relative attribute relevance. Weighting methods are classified as either *binary* or *continuous* respectively.

Representation: This dimension refers to whether the attribute set is transformed to another representation. The majority of weighting methods work with the attribute set as given. The DANN classifier transforms the problem space to remove globally irrelevant attributes.

Generality: Many weighting schemes assume the relevance of attributes is constant globally. This assumption is not always valid. The relevance of attributes may vary with the position in the problem space. DANN classification adapts locally to the position of the decision boundaries.

Knowledge: Domain specific knowledge can be used to determine attribute weights, suggest attribute transformations, and assign instance-specific weights. Few instance-based learners employ domain specific knowledge. Domain specific knowledge is often used in case-based reasoning (e.g., PROTOS, Porter *et al.*, 1990), both in determining the relevance of particular features and in defining the case representation itself.

Wettschereck and Aha compare five instance-based algorithms with respect to the feedback dimension. Two of the algorithms incorporated feedback weighting; one that is a slight variant of the IB4 weighting scheme, and a *k*-NN classifier using the weight optimising method from VSM (Lowe 1995). The other three algorithms incorporated ignorant weighting schemes. The first of these algorithms used the *cross-category feature importance*, defined as

$$weight_i = \sum_{c \in C} \mathrm{P}(c \mid i)^2 .$$

Thus, the weight of attribute *i* is based on the skew in the conditional probabilities.

The second algorithm used the VDM metric of Stanfill and Waltz (1986), described previously. The third ignorant weighting method used the mutual information between the class and feature values. The mutual information is the decrease in uncertainty about one variable's value given the value of the second. The weights are calculated as

$$weight_i = \sum_{v \in V_i} \sum_{c \in C} P(x_c = c \wedge x_i = v) \log \frac{P(x_c = c \wedge x_i = v)}{P(x_c = c) P(x_i = v)},$$

where V_i is the set of values that attribute *i* may take, $\mathrm{P}(x_c=c)$ is the probability that the class of some training instance is *c*, and $\mathrm{P}(x_i=v)$ is the probability that its value for attribute *i* is *v*.

The five algorithms were tested on several datasets chosen to test an algorithm's ability to handle attributes that were irrelevant, noisy, or interacted with other attributes. Wettschereck and Aha concluded that: all methods can tolerate irrelevant attributes unless there are many interacting attributes; feedback methods are better at isolating a few interacting attributes; feedback methods appear to learn faster than ignorant methods; and ignorant methods are sensitive to the data preprocessing.

These conclusions state more about the particular weighting methods used in the experiment than the general benefits of feedback versus ignorant weighting methods. For example, the ignorant methods were found to be sensitive to preprocessing. All the ignorant methods used in the experiment were designed only for assigning weights to symbolic attributes, and datasets involving numeric attributes had to be discretised. It is only to be expected that these methods do not perform well on this type of problem. However, one could design an ignorant weighting scheme that does handle numeric attributes without discretisation (such as the method used by DANN). The ignorant methods selected by Wettschereck and Aha do not appear to be well selected. Feedback methods do have an advantage; since they automatically adjust to maximise their classification performance, they can overcome initially inaccurate weight estimates and adapt to statistical differences between the training and test data. An obvious extension would be to use an ignorant method to estimate initial weights, and allow a feedback method to continually refine them.

3.5 Instance Importance

Some instances in the database may be better predictors than others, and instance-weighting algorithms seek to capitalise on this. One reason for the difference in predictive accuracy is that some instances may be the result of erroneous data, or may be atypical of their class. This issue is intertwined with reducing the memory requirements of instance-based learners, because if a training instance is sufficiently unimportant it may be removed from the database altogether.

With the aim of decreasing storage requirements, Aha, Kibler and Albert (1991) introduce a system called IB2 that saves only misclassified instances.

The idea is that only instances near concept boundaries are required to produce correct classifications, and that (since the exact concept is not known) misclassified instances provide a good indication of where concept boundaries lie. In a noise-free example domain, IB2 required that only 5% of training instances be stored with a minimal drop in accuracy. Unfortunately IB2 is sensitive to noisy data—these are often misclassified and are therefore added to the instance database, causing further misclassifications. Obviously the correctness of initial classification alone is not always a good indicator of instance importance. Another indicator of the predictive ability of an instance is its performance in the past. IB3 (Aha, Kibler and Albert, 1991) can be thought of as a simple instance-weighting algorithm that uses past performance statistics to divide instances into three pools: those that may currently be used in giving predictions; those that may not be used (but we are still gathering statistics on); and those that are so poor that they are removed from the database. For noisy datasets this resulted in a further reduction in storage requirements, and increased overall classification accuracy.

A similar instance-weighting scheme is used in Cost and Salzberg's (1993) instance-based learner PEBLS. They introduce the Modified Value Difference Metric (MVDM), which replaces the symbol weight term of the Stanfill and Waltz VDM with an instance weighting mechanism:

$$\mathrm{MVDM}(x, y) = weight_x weight_y \sum_{i=1}^{n} f(x_i, y_i),$$

where $f(x_i, y_i)$ is identical to that of the VDM, and $weight_x$ and $weight_y$ are the instance weights for instances x and y based on their past performance. When a new instance is added to the database, it is given an initial weight the same as its nearest neighbour. Each time an instance gives an incorrect prediction its weight is increased, making its area of influence smaller. Thus, the weighting system does not remove the instance altogether (as IB3 does); rather the instance signifies a small exception in the concept space.

Another indicator of a training instance's predictive value is its distance from the test instance (this is the fundamental assumption of instance-based learners). It is only a small extension to assume that in the case of *k*-NN algorithms each of the *k* neighbours should have its vote weighted proportional to its distance from the current instance. Dudani (1976) proposed a weighting function where the weight assigned to an instance is proportional to its distance relative to the nearest and furthest of the *k* neighbours:

$$weight_j = \begin{cases} \dfrac{d_k - d_j}{d_k - d_1}, & d_k \neq d_1 \\ 1, & d_k = d_1 \end{cases}$$

where d_j, $1 \leq j \leq k$, are the distances of each instance. Dudani also suggested two other potential weighting functions; one where the weight of an instance is given by the reciprocal of its distance, $weight_j = {}^1\!/_{d_j}$ ($d_j \neq 0$), and one based on the rank of the neighbour, $weight_j = k - j + 1$. Stanfill and Waltz's (1986) system MBRtalk used the reciprocal distance weighting function applied to the ten nearest neighbours retrieved by their VDM metric, and produced a prediction based on this weighted vote.

Macleod, Luk, and Titterington (1987) pointed out that in Dudani's weighting function, the *k*th neighbour is effectively removed from participating in the classification, and suggested a generalisation of Dudani's function:

$$weight_j = \begin{cases} \dfrac{(d_s - d_j) + \alpha(d_s - d_1)}{(1+\alpha)(d_s - d_1)}, & d_s \neq d_1 \\ 1, & d_s = d_1 \end{cases}$$

where α is a positive constant and $s = k, k+1, \ldots$ (choosing $\alpha = 0$ and $s = k$ gives Dudani's original function). Their experiments indicated an improvement in performance for several alternative values of α and s, and conclude that in some cases a carefully chosen weighted k-NN function can outperform an unweighted k-NN.

Keller, Gray, and Givens (1985) developed a "fuzzy k-NN" algorithm. Each training instance is assigned a class-membership vector which may either give complete membership to their known class and non-membership to all other classes (the "crisp" method), or partial membership in each class may be assigned according to the distances from the training instance to the class means (a "fuzzy" method). In classifying a test instance, a distance weighted average of the class-membership vectors of the k nearest neighbouring instances yields a class-membership vector for the test instance. The membership of the test instance x in class c_i is given by

$$c_i(x) = \frac{\sum_{j=1}^{k} c_{ij} weight_j}{\sum_{j=1}^{k} weight_j},$$

where c_{ij} is the membership the jth neighbour in class i, and $weight_j = d_j^{\frac{-2}{m-1}}$. For most of their experiments a value of m=2 was used, although it was reported that similar performance was obtained with other values. While the classification for the test instance is taken as that with the highest entry in the class-membership vector, it is noted that misclassifications are most likely when this entry is not significantly larger than other elements of the vector.

Lowe (1993) reports a method called variable-kernel similarity metric (VSM) learning which combines the benefits of looking at the k nearest neighbours with the smooth weighting decline of a Gaussian kernel. The VSM learner combines the contribution of the neighbours in the same fashion as the crisp method above, with the weighting function determined by a Gaussian kernel centred at the current instance:

$$weight_j = e^{\frac{-d_j^2}{2\sigma^2}},$$

where d_j is the attribute-weighted Euclidean distance to the *j*th neighbour. The width of the kernel is determined by σ, which is a multiple of the average distance to the *M* nearest neighbours (e.g., $M = k/2$)

$$\sigma = \frac{r}{M}\sum_{m=1}^{M} d_m .$$

Both *r* and the attribute weights are determined by optimisation using a cross-validation method (conjugate gradient descent).

The instance weighting methods presented fall in two categories, those that use past performance to assign an overall importance to an instance's predictions (regardless of the current test instance), and those that determine the importance of each training instance with respect to the instance currently being classified. These methods are not mutually exclusive—for example, the VSM learner and PEBLS use weighting schemes of both types. As with feedback attribute weighting methods, performance-history instance weighting can adapt to statistical differences between the training and test data.

3.6 Missing Information

In practical datasets, missing values may occur for a number of reasons, such as malfunctioning measurement equipment, change in experimental design during data collection, and merging of several similar but not identical datasets. In some cases the presence of missing values is in itself informative (for example, the omission of a value may indicate the attribute is not even meaningful for the current instance.) Rather than remove attributes or features that contain missing values, the best should be done with the data that is available. Dixon (1979) presents the earliest research that attempts to compare several methods for dealing with missing values.

Dixon first performed the following *gedanken experiment* to get an intuitive feeling for how missing data should be handled. Consider the following two 5-dimensional vectors A and B:

A = (1.2	3.7	10.9	6.3	5.9)
B = (1.1	3.5	blank	6.2	5.7)

Our intuition would suggest that the blank value is close to 10.9, since the other values are close. In addition, we may guess that the blank value is slightly lower than 10.9, since all the other B components are lower than their A counterparts. Dixon interprets this in the form of three assumptions:

1) that the data is clustered;
2) that the features are correlated so that one can carry out linear interpolation across features;
3) that if distances are small along the four dimensions, they will be small in the fifth.

Aided by these assumptions, Dixon suggested the following six methods of handling missing values:

1NNFILL: This method is used as a preprocessing stage to fill any missing values in the training data. If an instance has a missing value for some attribute, the blank is filled with the corresponding attribute of the instance's nearest neighbour.

4NNFILL: Similar to 1NNFILL—missing values are replaced with the average value of the instance's 4 nearest neighbours.

DELETE: This is a preprocessing method that deletes any instances or attributes that contain blanks, attempting to minimise the amount of good data thrown away. The instance or attribute that contains the highest percentage of missing values is deleted. This process is repeated until all missing values are eliminated.

NORMAL: This method is applied when the distance between two instances is calculated. The distance to a missing value is assumed to be the same as the average distance between the non-missing values in other attributes; the distance between two instances is calculated using the attributes with values given, and then scaled proportional to the number of attributes with missing values.

AVERAGE: Similar to NORMAL, but the average is calculated over the particular attribute with the missing value (that is, calculate the average distance between all pairs of values for that attribute).

ZERO: The distance to any missing value is zero.

Dixon analysed the performance of these methods in a number of example domains, and found that DELETE and ZERO are both bad strategies. The other methods performed equally, although Dixon concluded that NORMAL and AVERAGE appeared to be more consistent than 1NNFILL and 4NNFILL. Dixon does not point out that in his experiments NORMAL performs consistently better than AVERAGE for high levels of missing values, but the situation is reversed for low levels of missing values. The methods as described by Dixon apply only to numeric attributes. Aha (1990) examines the behaviour of three methods for treating missing values that are defined for symbolic attributes also.

MAXDIFF: This method treats missing values as maximally different from the value given. Therefore the similarity of a symbolic value to a missing value is always 0. The similarity of a (normalised) numeric value x to a missing value is $\min\{x, 1-x\}$. The motivation here is that for all possible values y, the similarity of x and y should be higher than the similarity of x and a missing value.

MODEMEAN: The missing value is replaced with the most probable value from previously observed values. This method is similar to Dixon's AVERAGE above.

IGNORE: The attribute containing the missing value is ignored, and the measure normalised. This method is the same as NORMAL above.

Aha discovered that none of the three strategies is consistently better than the other two, although in a particular domain there can be a wide difference in performance. Aha concluded that further investigation is required into characterising the conditions under which each scheme performs well.

One method often seen in the statistics literature for estimating values for missing data is the expectation maximisation (EM) algorithm. Dempster, Laird, and Rubin (1977) first formalised the algorithm, although applications had been described earlier. The EM algorithm consists of two steps: the *expectation* step, where the missing values are estimated using initial guesses as to the parameters of the distribution of values; and the *maximisation* step, where the data (including the estimated values) are used to find maximal-likelihood estimates of the distribution parameters. The expectation and maximisation steps are iterated until no significant variations in the parameters occur. The missing values may then be replaced with the estimates from the final iteration.

It is interesting to note the wide range of methods proposed for dealing with missing values. For example, Dixon's ZERO approach is the opposite of Aha's MAXDIFF method. It is also interesting that no method appears consistently better than the others in spite of the differences in approach.

3.7 Robustness

An area that has received little attention in past research, particularly in machine learning, concerns the robustness of similarity functions and the algorithms employing them. We define robustness to mean that the similarity function should be sensitive to the domain only—the function should not register differences in similarity where they do not exist in the domain. For example, consider evaluating the similarity of weather maps based on a single correspondence of features between the two maps. The similarity can be plotted as one image undergoes some continuous change in its features. At some point the feature correspondence considered best may change and this will cause a sudden change in measured similarity, even though there is no sudden change in the images themselves. Reasonable behaviour in the case of smoothly changing objects is for the similarity to also change smoothly.

These ideas extend further than the similarity measure itself. For example, an application may adapt parameters to the similarity function on the basis of objects seen so far. In this case, small changes in the attributes of training objects or their presentation order should not have large effects on the refinement of the measure. Sensitivity to presentation order is one of the problems Wettschereck and Dietterich (1995) found when analysing EACH's performance.

Both Ting (1995) and Wilson and Martinez (1997) work with coherent treatment of different attribute types described in Section 3.3.3 are concerned with improving the robustness of the learning algorithms. Aha (1992) describes another case where robustness of an instance-based learner was improved. IB5 is a modification to IB4 that adapts to the introduction of novel attributes during training. The introduction of novel attributes may be simulated in IB4 by presenting the attribute values as missing until actual values appear, but the problem is that during this time IB4 sets the weighting for this attribute to be very low. However, IB5 only performs weight modifications when both values are not missing, allowing it to reach a correct weighting much faster once values begin to appear.

One way to increase robustness is to ensure that domain knowledge is employed whenever available. The following studies suggest other methods to improve robustness in a similarity measure.

3.7.1 Smoothness

It is suggested above that smoothness is one ingredient of a robust similarity function. The following research provides evidence for the importance of smoothness in a game-playing evaluation function, and the work is relevant to similarity functions in general. Berliner (1980) describes the development of an evaluation function for the game of backgammon. Using a single evaluation function for the entire game did not produce good play because different tactics are required for various stages of the game. As a solution Berliner tried having several evaluation functions, each for different stages of the game. One function would be used for the initial stages, and at some point the next evaluation function would take over. However, the discontinuities between the separate functions sometimes caused the program to avoid

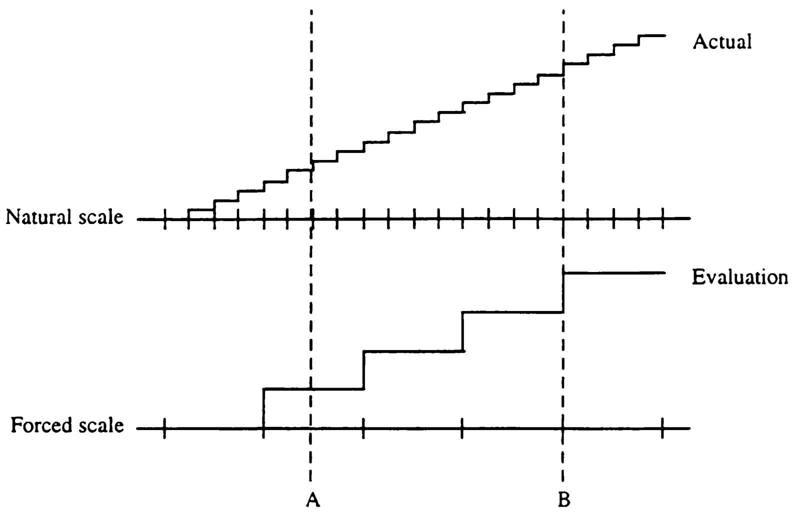

Figure 3.3: Non-smooth evaluation function

making the transition between game phases, essentially attempting to delay the inevitable.

The primary conclusion drawn by Berliner is the importance of smoothness in an evaluation function. An evaluation function defines a surface in the feature hyperspace. If a surface is not smooth it may have a ridge, discontinuity or sudden step in the surface. Values on either side of such a blemish may be quantitatively very different, and therefore a small change in the value of one feature may produce a large change in the value of the evaluation function. Berliner found that when a program has the ability to manipulate such a feature, often it does so to its detriment. For example, if the change in evaluation signals an improvement, the program will attempt to enter that region of the hyperspace. If the evaluation represents an unfavourable change, the program will try to avoid crossing the blemish. The program is making decisions based on characteristics that are not really there.

To illustrate the problems caused by non-smooth evaluation functions, consider the evaluation functions depicted in Figure 3.3. The upper function makes use of the natural scale of the attribute, and the lower evaluation function internally simplifies this scale (this example could represent the conversion of some floating-point attribute to integer). On either side of point A the lower evaluation function sees no difference in goodness, and on either side of point B the lower evaluation function sees a large difference. The reality is that for both points there is a small difference either side (as seen in the upper function). Berliner found that these problems were the cause of poor performance in his early backgammon programs. By ensuring a smooth evaluation function, Berliner was able to achieve a significant improvement in his program's performance (his program BKG9.8 defeated the World Backgammon Champion in 1979).

Berliner's findings for evaluation functions also apply to similarity measures. If the similarity measure is not smooth it will suffer problems. Between some points in the feature space the similarity measure will return a disproportionately higher similarity than it should, and in others the measure will return a lower similarity than it should. When asked to judge which of two objects is most similar to some object, the similarity measure may result in an incorrect choice being made. An illustration of this exact problem was given in the discussion of the UV datasets in Section 3.3.3.

3.7.2 Multiple Paths

Yee and Alison's (1993) research into the comparison of DNA sequences presents another idea for increasing similarity function robustness. Under one evolutionary model, DNA sequences may undergo any of a set of genetic mutations, such as insertion or deletion of nucleotides. A number of successive mutations may completely transform one DNA sequence to another. Depending on the probabilities for each of the basic mutation operators, some possible transformation paths are more likely than others. The

1. Start with some reasonable values for the probabilities.
2. Calculate the single optimum alignment using the dynamic programming algorithm.
3. Recalculate the mutation probabilities from their observed frequencies in the optimum alignment.
4. Repeat steps 2–3 until the solution converges (typically after 4–8 iterations).

Figure 3.4: Algorithm to discover mutation probabilities

objective was to discover the parameters (that is, probabilities for each of the basic mutations) for the evolutionary model that produced two strings.

Yee and Alison initially used the iterative approach shown in Figure 3.4. Although convergence is guaranteed, the method may get trapped in a local optimum. The dynamic programming algorithm (DPA) is a well-known method for finding an optimal alignment between two strings for given instruction costs. The DPA uses a matrix D, where D_{ij} is the minimum cost to edit the first i symbols of string A into the first j symbols of string B. Alison and Yee formulate the DPA in terms of minimum description length, where the increment in cost for each edit operation is the length to encode the relevant operation. The DPA result is the message length to encode the optimum alignment. In MDL terms, this algorithm searches for a hypothesis that specifies the mutation probabilities along with a specific DNA alignment that can be used to encode the strings. The obvious null-hypothesis is that the strings are unrelated (in that case it is best to encode the DNA strings literally at 2 bits per symbol).

Yee and Alison define a hypothesis called the r-theory, that the two DNA sequences are related but in an unspecified manner. The strings are encoded based on all possible alignments between the two strings. In order to estimate the mutation probabilities the algorithm in Figure 3.4 is modified. The central step of the DPA is adapted to store the message length resulting from encoding all possible paths rather than the shortest message length from a single path encoding, as follows:

$$D_{ij} = \text{logplus}\begin{pmatrix} D_{i-1,j-1} + \text{if } A[i] = B[j] \text{ then } \mathrm{ML}(match) \text{ else } \mathrm{ML}(change) \\ D_{i,j-1} + \mathrm{ML}(ins_B) \\ D_{i-1,j} + \mathrm{ML}(ins_A) \end{pmatrix},$$

where $\text{logplus}(\log(P), \log(Q), \ldots) = \log(P + Q + \ldots)$. The new DPA calculates the message length to generate strings A and B in an unspecified manner.

The iterative method to estimate probabilities for the change, match and insert operators is also modified, since the DPA no longer tries to find a single alignment. From the DPA matrix, weighted averages of the instruction frequencies are calculated to obtain parameters for the next iteration.

To evaluate the r-theory (the method incorporating all possible transformation paths) against the single alignment method, Yee and Alison generated many strings using known parameters and measured the ability of the two methods to discover the actual parameters. Their results indicate that the r-theory performs better, particularly at higher mutation levels. The r-theory gives unbiased estimates of the parameters of the evolutionary process that gave rise to the two strings A and B. Parameter estimates based on a single alignment are biased. A possible explanation for the performance difference is that evolution is a random process, and the probability that it follows the optimum path may be small. Considering all possible edit paths appears to be a useful method for removing biases in the comparison.

3.8 Other Issues

The previous sections discuss issues particularly relevant to this thesis. There are other points regarding instance-based learners that are worthy of a brief discussion, although they are not directly related to the problem of measuring instance similarity. In particular, the issues of storage requirements and interpretability have not been discussed. These issues are presented in the following sections for completeness, not as issues to be addressed by this thesis.

3.8.1 Memory Requirements

One problem with the basic nearest neighbour algorithm is the large storage requirements—all instances are maintained in the database. Aside from the disadvantage in its own right, this increases the time taken to find a nearest neighbour if the distance to every instance in the database is calculated. Edited nearest neighbour algorithms are selective in which instances are stored in the database and used in classification with the objective of reducing storage requirements. In probably the earliest work in this area, Hart (1968) describes the condensed nearest neighbour rule (CNN). The set of instances kept in the database are defined to be some subset of the instances such that those left out would be correctly classified by those in the subset. This subset is generated by an iterative procedure: instances are either placed in the database if their class was not predicted by the instances already in the database, or they are placed in a *grab-bag*. Repeated passes are made through the grab-bag attempting to classify each instance using those in the database. Any incorrectly classified instances are moved to the database, and when an entire pass is made through the grab-bag without any transfers the process ends. CNN trades a possible drop in classification performance on new instances for reduced storage requirements (and the corresponding reduction in computation time to reach a decision). In a simple letter recognition experiment with a large training set, the CNN rule required only 5% of the instances be retained.

Gates (1972) further modifies CNN to produce the reduced nearest neighbour rule (RNN). The idea here is that each instance in the CNN database is tested to see whether its removal results in incorrect classifications of those instances not in the database. In their experiments, the CNN rule required on average 16% of the training instances be retained, and the RNN gave a further reduction to 12%.

A reduction in storage requirements can also be a secondary achievement during instance weighting. We have seen previously that IB3 is able to remove sufficiently unimportant instances from the database as part of its instance weighting procedure. A similar reduction is often achieved by methods aimed at increasing the interpretability of an instance-based learner's results.

3.8.2 Interpretability

Instance-based learners often perform very well at classification tasks—answering the question "what class does this instance belong to?". They are not good at answering the question "why does this instance belong to that class?" (for which decision trees or rules are ideal). Many machine learning algorithms generalise training data to form simple classification rules. These systems require more training data in order to learn than instance-based learners; however, they have the advantage of being faster during classification, as well as providing the user with an easily understood representation of the concept learned (i.e. the rules themselves). It is difficult for a human to look at an instance database and gain an understanding of the domain. Instance generalisation is a way of providing more human interpretable output.

Instance-averaging algorithms perform a simple form of generalisation. Rather than forming rules or decision trees, multiple instances are combined together in the database, retaining the primitive form of instances. These instances may be considered prototypes for their class.

Bradshaw (1986) describes an early instance-averaging system called Nexus. When a training instance correctly classifies a new instance, a weighted average is carried out with the new instance. Weights of 1 are initially assigned to all training instances, and whenever an instance is averaged the weights increase by 1 (the weights are a therefore measure of how many instances are represented by the training instance). If the new instance is incorrectly classified, it is simply added to the database. Instances that are not useful are removed. NEXUS was applied to speech recognition and obtained an accuracy about 13% higher than a traditional speech recognition program. Instance-averaging algorithms such as NEXUS have lower storage requirements for the instance database and fewer classification errors caused by noisy data, but it is not clear how they can be extended to handle symbolic attributes (which cannot be averaged easily).

An alternative approach to providing interpretable output is to combine rules and instances. A rule of the form "IF (value1 $\leq$ attribute A $\leq$ value2) AND (value3 $\leq$ attribute B $\leq$ value4) THEN classification X" defines a hyperrectangle in the attribute hyperspace, and any new instance which falls inside the hyperrectangle is given the same classification.

Salzberg (1991) introduces a system capable of generalising instances to form hyperrectangles while maintaining the nearest neighbour approach. In the nested generalised exemplar (NGE) theory, the memory is "seeded" with a small set of training instances which can be considered point hyperrectangles. A new example is matched to the nearest neighbouring hyperrectangle in memory using an instance and attribute weighted Euclidean distance function. In the case of a tie between hyperrectangles, the smallest is preferred (as it is the most specific). If the new instance is correctly classified, the hyperrectangle is extended (generalised) to cover the new instance. When an instance is incorrectly classified, the second closest hyperrectangle is examined to see whether it would have given the correct classification, had it been used. If so, it is generalised to cover the new instance, otherwise the new instance is added to the database. One benefit of storing generalised instances as hyperrectangles is that hyperrectangles can be examined by a human to give an intuitive idea of the concept description (which is hard with traditional IBL algorithms). In experiments, it was found that their implementation, called EACH, performed as well as other algorithms, although not markedly better. EACH required approximately 10% of the instances be stored.

Wettschereck and Dietterich (1995) conducted an analysis of EACH's poor performance in comparison with *k*-NN and suggested three improvements: first, that overlapping hyperrectangles be avoided; second, that if possible the classifier should be trained on the entire training set in batch mode to avoid problems caused by presentation order (which indicates the original algorithm for growing the hyperrectangles was not robust); third, that the attribute weighting scheme implemented in NGE was poor. They conclude that their modified algorithm (BNGE) gives superior performance in domains where an axis-parallel hyperrectangle bias is appropriate, or interpretability and memory efficiency are important, but in other domains *k*-NN is a better choice.

Wettschereck (1994) further examined the performance of BNGE and discovered that most misclassification occurs when an instance is not covered by a hyperrectangle. A hybrid classifier that uses *k*-NN whenever an instance does not fall inside a hyperrectangle gave a substantial improvement in performance while retaining much of the speed and understandable concept representation of BNGE.

3.9 Conclusions

Specific solutions to many issues pertaining to similarity functions in instance-based learning have been described. There are several functions for measuring the similarity between instances represented as numeric attributes. Other functions have been developed for instances represented as symbolic attributes. There has been little work on dealing directly with attributes other than plain numeric and symbolic. Combining functions designed for numeric attributes with those designed for symbolic attributes causes unwanted biases. The addition of attribute weighting schemes is intended to counteract these biases as well as to adapt similarity functions to the particular problem domain. With the exception of Ting and Wilson and Martinez' work, there appears to be no research directed at combining measures from multiple sources coherently, or at allowing domain information to be captured within the similarity function in a general way.

There are problems with the current treatment of other issues. Dealing with instances that have information missing is a common requirement for an instance-based learning algorithm. Again, several different methods have been successfully used in the literature; however, not all of these are applicable for different attribute types. There is a general *ad hoc* approach to dealing with this and other issues. Solutions to different problems are combined without thought to whether they are compatible. The resulting instance-based learners often exhibit non-robust behaviour (the EACH learner is a good example of this). Some research has attempted to address specific cases of non-robustness, but none have been concerned with producing a robust and general similarity function from the outset.

These issues are vitally important in fielded applications, where a similarity function must accurately reflect the characteristics of the domain to give optimum performance. These problems are also interesting from an academic point of view—how can these problems be solved within a general framework. Many of the instance-based algorithms described perform well in practical and artificial domains. However, comparison is usually made with other similar algorithms rather than domain specific algorithms. Few of these solutions have more than empirical justification. Certainly none of the solutions are related by a common framework. Discovering better ways of dealing with these issues is an area ripe for research.

Chapter 4

Similarity Function Design

This chapter presents a framework for the design of similarity functions that attempts to address the issues identified in the previous chapter. Similarity is interpreted as the likelihood of transforming one instance to another. Similarity is determined by calculating the probability of a sequence of basic transformations; to improve similarity function robustness, all possible transformation sequences between instances are considered. Algorithmic complexity theory provides the basis for this approach to measuring similarity. This chapter presents several examples that illustrate how to construct similarity functions for simple domains.

4.1 Complexity as Distance—Probability as Similarity

The approach we take to computing the distance between two instances is inspired by algorithmic complexity theory. The central idea is that the distance between instances can be defined as the complexity of transforming one instance to another. Kolmogorov complexity gives us a measure of the amount of information in an object. If the object is a description of a transformation between two instances, a low Kolmogorov complexity would imply that the instances are similar. The similarity between instances decreases as the information needed to describe the transformation increases. This raises the question of how to describe the transformation between instances. First, a finite set of transformations that map instances to instances is defined. A "program" to transform one instance (a) to another (b) is a finite sequence of transformations starting at a and terminating at b. This procedure will not enable us to calculate the Kolmogorov complexity (since there may be more efficient encodings for the transformation), but it will allow us to calculate an upper bound to it. The set of transformations should be chosen carefully as this will affect how close an approximation the upper bound is.

Following the usual development in complexity theory, transformation programs are made prefix-free by appending a termination symbol to each string. Recall from Chapter 2 that the Kolmogorov complexity of an object is the length of the shortest string describing it. A Kolmogorov distance between two instances can be defined as the length of the shortest transformation program connecting them. This method focuses on a single transformation (the shortest one), out of many possible transformations. Chapter 3 presented evidence that this type of approach is likely to make the resulting distance measure overly sensitive to small changes in the instance space, or to the set of basic transformations chosen, and therefore does not solve the robustness problem well. The distance measure defined below attempts to deal with this problem by considering all possible transformations between two instances (and hence we name our distance measure K*).

It is not immediately obvious how the transformation paths should be combined—adding the lengths of the different transformations is clearly not correct. The solution is to associate a probability with each sequence. If the complexity (length) of a program measured in bits is c, the corresponding probability is 2^{-c}. In particular, it is true that in any well-defined distance measure based on Kolmogorov complexity, the sum of this probability over all transformations will satisfy the Kraft inequality. This sum can be interpreted as the probability that a program will be generated by a random selection of transformations. In terms of the similarity between instances, it is the probability of reaching an instance after executing a random walk from the original instance and stopping. After summing over all paths this probability (similarity) may be transformed into units of complexity (distance) by taking the logarithm; however in most cases it is convenient to deal directly with probabilities.

This approach is somewhat different from traditional random walk treatment (Wax, 1954; Spitzer, 1975). In these models a particle moves stochastically without stopping. The type of question asked in random walk problems is "what is the probability of the particle being at point x at time t?" or "what is the probability that the first visit to point x occurs at time t?", whereas the question we ask is "what is the probability of stopping at point x, regardless of the time taken?". As we shall see though, our theory is general enough to encompass ordinary random walk models.

This approach may be likened to the Solomonoff-Levin universal prior. However, the Solomonoff-Levin prior employs a universal Turing machine, whereas our similarity measure uses a simplified machine that processes a defined set of basic transformations. In our formulation of transformation programs it is possible to ensure the transformation machine halts for all input, and therefore the similarity is computable.

The beauty of this interpretation of similarity is that it is applicable to many types of instances, be they one-dimensional numbers, points on a plane, images, or high-level feature descriptions. Because similarities are expressed in common units of probability, we may use standard methods for manipulating them.

There are two issues that require further explanation. The first is how to define a good set of basic transformations. Clearly the set of basic transformations must be expressive enough to allow any two instances to be connected by some program. In many cases a reasonable set will be apparent by looking at the underlying domain. For example, if the instances are integer numbers the most obvious transformations are to add one and subtract one. To define a set of basic transformations for other instance types may require more creative modelling. Changing the set of basic transformations allows the similarity function to be customised to different types of instance, whether they are simple numbers, DNA sequences, or weather maps.

The second issue is how to determine the actual length of a transformation program in bits (or its corresponding probability). One method is to assign reasonable probabilities to each basic transformation. The actual probabilities chosen will depend on the domain, however they must sum to less than 1 for the resulting code to satisfy the Kraft inequality. A special case arises when they sum to exactly 1—the resulting code is complete and the transformation machine will halt for all randomly chosen programs. The probability of an entire transformation program is the product of the probabilities of each transformation in the sequence. This method is used in most of the worked examples, although other methods could be envisaged. For example, the probability of each basic transformation could be adaptively altered on the basis of its context. This approach might be employed in DNA sequence comparison by allowing the probability of a "delete nucleotide" instruction to be higher if the immediately preceding instruction is also a delete instruction, effectively giving block deletions a higher probability than many single deletions. The probabilities assigned to the basic transformations permit customisation of the similarity measure to the actual domain. For example, weather map instances may be used in predicting both temperature and pressure. While the set of basic transformations will be the same for both domains, the assigned probabilities may well be different.

4.2 Specification of K*

This section presents a formal specification of the transformation-based similarity measure, before proceeding to worked examples of constructed similarity functions.

Let **I** be a (possibly infinite) set of instances and **T** a set of transformations that operate on I. Each $\mathrm{t} \in \mathbf{T}$ maps instances to instances: $\mathrm{t}: \mathbf{I} \rightarrow \mathbf{I}$. **T** contains a distinguished member σ (the stop symbol), which for completeness maps instances to themselves ($\sigma(a) = a$). Let **P** be the set of all prefix codes from **T*** that are terminated by σ. Members of **T*** (and so of **P**) uniquely define a transformation on **I**:

$$\bar{\mathrm{t}}(a) = \mathrm{t}_n(\mathrm{t}_{n-1}(\ldots\mathrm{t}_1(a)\ldots)) \text{ where } \bar{\mathrm{t}} = \mathrm{t}_1,\ldots\mathrm{t}_n.$$

(We denote members of **T*** with an overbar, and members of **T** without. Concatenation of $\bar{t}_1, \bar{t}_2 \in \mathbf{T}^*$ is denoted by $\bar{t}_1\bar{t}_2$. Concatenation of $\bar{t}_1 \in \mathbf{T}^*$ with $t_2 \in \mathbf{T}$ is denoted by $\bar{t}_1 t_2$. The length of $\bar{t} \in \mathbf{T}^*$ is denoted by $l(\bar{t})$.)

A probability function p is defined on **T*** which satisfies the following properties:

$$0 \leq \frac{p(\bar{t}u)}{p(\bar{t})} \leq 1. \tag{4.1}$$

That is, $0 \leq p(u \mid \bar{t}) \leq 1$. The probability of transformation u occurring is not necessarily independent of the previous transformations $\bar{t}$.

$$\sum_{u \in \mathbf{T}} p(\bar{t}u) = p(\bar{t}). \tag{4.2}$$

Thus, from any point in a transformation sequence, the probabilities of the next possible transformations sum to 1.

$$p(\Lambda) = 1. \tag{4.3}$$

That is, the initial probability (before any transformations take place) is 1.

Theorem 4.4: $\Omega = \sum_{\bar{t} \in \mathbf{P}} p(\bar{t}) \leq 1.$

That is, the sum of all prefix transformation programs is no greater than 1. We call Ω the halting probability, because it may be thought of as the probability that the transformation-processing machine halts when its program is supplied randomly.

Proof: Let $\mathbf{Q}_k = \mathbf{P}_k \cup \mathbf{R}_k$, where $\mathbf{P}_k = \{\bar{t} : \bar{t} \in \mathbf{P}, l(\bar{t}) \le k\}$, the set of all prefix codes with length k or less, and $\mathbf{R}_k = \{\bar{t} : \bar{t}\sigma \in \mathbf{P}, l(\bar{t}) = k\}$, the set of not-yet terminated codes of length k. Note that $\mathbf{P}_k$ and $\mathbf{R}_k$ are disjoint. Then

$$\begin{aligned} \mathbf{Q}_{k+1} &= \mathbf{P}_{k+1} \cup \mathbf{R}_{k+1} \\ &= \left(\mathbf{P}_k \cup \{\bar{t}\sigma : \bar{t} \in \mathbf{R}_k\}\right) \cup \{\bar{t}u : \bar{t} \in \mathbf{R}_k, u \in \mathbf{T}, u \ne \sigma\} \end{aligned}$$

So we see that $\lim_{k \to \infty} \mathbf{Q}_k \supseteq \mathbf{P}$.

First, we show by induction that $\sum_{\bar{t} \in \mathbf{Q}_k} p(\bar{t}) = 1$ for all natural numbers k.

Base step: $\sum_{\bar{t} \in \mathbf{Q}_0} p(\bar{t}) = p(\Lambda) = 1$.

Induction step: Assume $\sum_{\bar{t} \in \mathbf{Q}_k} p(\bar{t}) = 1$ for some value k. Then,

$$\begin{aligned} \sum_{\bar{t} \in \mathbf{Q}_{k+1}} p(\bar{t}) &= \sum_{\bar{t} \in \mathbf{P}_{k+1}} p(\bar{t}) + \sum_{\bar{t} \in \mathbf{R}_{k+1}} p(\bar{t}) \\ &= \left(\sum_{\bar{t} \in P_k} p(\bar{t}) + \sum_{\bar{t} \in R_k} p(\bar{t}\sigma) \right) + \sum_{\bar{t} \in R_k} \sum_{\substack{u \in \mathbf{T} \\ u \ne \sigma}} p(\bar{t}u) \\ &= \sum_{\bar{t} \in P_k} p(\bar{t}) + \sum_{\bar{t} \in R_k} \sum_{u \in \mathbf{T}} p(\bar{t}u) \\ &= \sum_{\bar{t} \in P_k} p(\bar{t}) + \sum_{\bar{t} \in R_k} p(\bar{t}) \\ &= \sum_{\bar{t} \in \mathbf{Q}_k} p(\bar{t}) \\ &= 1. \end{aligned}$$

Now, since

$$\begin{aligned} \sum_{\bar{t} \in \mathbf{P}_k} p(\bar{t}) &= \sum_{\bar{t} \in \mathbf{Q}_k} p(\bar{t}) - \sum_{\bar{t} \in \mathbf{R}_k} p(\bar{t}) \\ &= 1 - \sum_{\bar{t} \in \mathbf{R}_k} p(\bar{t}), \end{aligned}$$

the inequality holds.

◊

The case where $\Omega = 1$ requires that $\lim_{k\to\infty} \sum_{\bar{t}\in \mathbf{R}_k} p(\bar{t}) = 0$, that is, the probability that a randomly chosen program has not been terminated approaches zero as the length of the program increases. One way to ensure this is to impose an extra condition on *p*, for example

$$0 < c < \frac{p(\bar{t}\sigma)}{p(\bar{t})} \leq 1, \qquad (4.5)$$

where *c* is some small constant. Note that it is not sufficient to simply have $0 < \frac{p(\bar{t}\sigma)}{p(\bar{t})} \leq 1$. For example, if $\frac{p(\bar{t}\sigma)}{p(\bar{t})} = \frac{1}{l(\bar{t})}$, we find that $\lim_{k\to\infty} \sum_{\bar{t}\in \mathbf{R}_k} p(\bar{t}) = \frac{1}{e}$.

The probability function P* is defined as the probability of all programs transforming instance *a* to instance *b*:

$$\mathrm{P}^*(b \mid a) = \sum_{\substack{\bar{t}\in \mathbf{P} \\ \bar{t}(a)=b}} p(\bar{t}). \qquad (4.6)$$

P* satisfies the following properties:

Theorem 4.7: $0 \leq \mathrm{P}^*(b \mid a) \leq \Omega \leq 1$.

That is, the P* transformation probability (or similarity) from instance *a* to instance *b* is between zero and the halting probability. When no programs transform *a* to *b*, $\mathrm{P}^*(b \mid a) = 0$, and when all (halting) programs transform *a* to *b*, $\mathrm{P}^*(b \mid a) = \Omega$.

Proof: Theorem 4.4 states that the probability of all possible programs is Ω. Since $\{\bar{t} : \bar{t} \in \mathbf{P}, \bar{t}(a) = b\} \subseteq \mathbf{P}$, the probability P* must be less than or equal to Ω.

◊

Theorem 4.8: $\sum_{b} \mathrm{P}^*(b \mid a) = \Omega$.

That is, from any initial instance *a*, the sum of transformation probabilities to all instances equals the halting probability.

Proof: From Theorem 4.7 we see that $\sum_{b} \mathrm{P}^*(b \mid a) = \sum_{b} \sum_{\bar{t} \in \mathbf{P}, \bar{t}(a) = b} p(\bar{t})$. This is equivalent to $\sum_{\bar{t} \in \mathbf{P}} p(\bar{t})$ which by Theorem 4.4 is Ω.

◊

The K* distance function is defined as:

$$\mathrm{K}^*(b \mid a) = -\log_2(\mathrm{P}^*(b \mid a)). \tag{4.9}$$

K* is not strictly a distance metric. For example, $\mathrm{K}^*(a \mid a)$ is typically non-zero, and the function (as emphasised by the | notation) is not necessarily symmetric.

The following properties are provable:

Theorem 4.10: $\mathrm{K}^*(b \mid a) \geq 0$.

That is, the K* transformation complexity (or distance) between two instances is greater than or equal to 0.

Proof: Directly from Theorem 4.7.

◊

Theorem 4.11: If we make it a condition that the basic transformations are independent, that is, $p(\overleftarrow{\mathrm{t}}\mathrm{u}) = p(\overleftarrow{\mathrm{t}})p(\mathrm{u})$ (which is also sufficient to ensure that $\Omega = 1$), then

$$\mathrm{P}^*(c \mid b)\mathrm{P}^*(b \mid a) \leq \mathrm{P}^*(c \mid a).$$

After converting from similarity to distance we see that this is actually the triangle inequality:

$$\begin{aligned}
&\mathrm{P}^*(c \mid b)\mathrm{P}^*(b \mid a) \leq \mathrm{P}^*(c \mid a) \\
&\Leftrightarrow -\log_2(\mathrm{P}^*(c \mid b)) - \log_2(\mathrm{P}^*(b \mid a)) \geq -\log_2(\mathrm{P}^*(c \mid a)) \\
&\Leftrightarrow \mathrm{K}^*(c \mid b) + \mathrm{K}^*(b \mid a) \geq \mathrm{K}^*(c \mid a).
\end{aligned}$$

Proof:

$$\begin{aligned}
\mathrm{P}^*(c \mid b)\mathrm{P}^*(b \mid a) &= \sum_{\substack{\overleftarrow{\mathrm{t}}\sigma \in \mathbf{P} \\ \overleftarrow{\mathrm{t}}(a) \models b}} p(\overleftarrow{\mathrm{t}}\sigma) \sum_{\substack{\bar{\mathrm{u}} \in \mathbf{P} \\ \bar{\mathrm{u}}(b) \models c}} p(\bar{\mathrm{u}}) \\
&\leq \sum_{\substack{\overleftarrow{\mathrm{t}}\sigma \in \mathbf{P} \\ \overleftarrow{\mathrm{t}}(a) \models b}} \sum_{\substack{\bar{\mathrm{u}} \in \mathbf{P} \\ \bar{\mathrm{u}}(b) \models c}} p(\overleftarrow{\mathrm{t}})p(\bar{\mathrm{u}}) \\
&\leq \sum_{\substack{\overleftarrow{\mathrm{t}}\sigma \in \mathbf{P} \\ \overleftarrow{\mathrm{t}}(a) \models b}} \sum_{\substack{\bar{\mathrm{u}} \in \mathbf{P} \\ \bar{\mathrm{u}}(b) \models c}} p(\overleftarrow{\mathrm{t}}\bar{\mathrm{u}}) \\
&= \sum_{\substack{\overleftarrow{\mathrm{t}}\bar{\mathrm{u}} \in \mathbf{P} \\ \overleftarrow{\mathrm{t}}(a) \models b \\ \bar{\mathrm{u}}(b) \models c}} p(\overleftarrow{\mathrm{t}}\bar{\mathrm{u}}) \\
&\leq \sum_{\substack{\overleftarrow{\mathrm{t}}\bar{\mathrm{u}} \in \mathbf{P} \\ \overleftarrow{\mathrm{t}}\bar{\mathrm{u}}(a) \models c}} p(\overleftarrow{\mathrm{t}}\bar{\mathrm{u}}) \\
&= \mathrm{P}^*(c \mid a).
\end{aligned}$$

That is, programs that transform a to b can be concatenated with programs that transform b to c, to give all programs that transform a to c by including b as an intermediate stage. All of these new programs are already included in the calculation of $\mathrm{P}^*(c \mid a)$, so it cannot be lower than the product of $\mathrm{P}^*(c \mid b)$ and $\mathrm{P}^*(b \mid a)$. If there are no programs that transform a to c that bypass b

(and the stop probability is 1), the two sides are equal, otherwise the inequality holds.

◊

Theorem 4.12: If, (in addition to the condition required for Theorem 4.11) each basic transformation $\mathrm{t} \in \mathbf{T}$ has an *inverse* $\mathrm{t}^{-1} \in \mathbf{T}$ such that $\mathrm{t}(a) = b \leftrightarrow \mathrm{t}^{-1}(b) = a$, and $p(\mathrm{t}) = p(\mathrm{t}^{-1})$, then

$$\mathrm{P}^*(a \mid b) = \mathrm{P}^*(b \mid a).$$

That is, the similarity function is symmetric.

Proof: First we show by induction that each $\bar{\mathrm{t}} \in \mathbf{T}^*$ has an *inverse sequence* $\bar{\mathrm{t}}^{-1} \in \mathbf{T}^*$ such that $\bar{\mathrm{t}}(a) = b \leftrightarrow \bar{\mathrm{t}}^{-1}(b) = a$ and $p(\bar{\mathrm{t}}) = p(\bar{\mathrm{t}}^{-1})$.

Base step: Let $\bar{\mathrm{t}} \in \mathbf{T}^*$ such that $l(\bar{\mathrm{t}}) = 1$. Then $\bar{\mathrm{t}}$ consists of a single basic transformation, for which an inverse exists as given.

Induction step: Assume an inverse sequence exists for all $\bar{\mathrm{u}} \in \mathbf{T}^*$ with $l(\bar{\mathrm{u}}) = k$. Let $\bar{\mathrm{t}} \in \mathbf{T}^*$ such that $l(\bar{\mathrm{t}}) = k+1$. Then $\bar{\mathrm{t}} = \bar{\mathrm{u}}\mathrm{v}$ with $\bar{\mathrm{u}} \in \mathbf{T}^*$, $l(\bar{\mathrm{u}}) = k$, $\mathrm{v} \in \mathbf{T}$. Let $\bar{\mathrm{t}}^{-1} = \mathrm{v}^{-1}\bar{\mathrm{u}}^{-1}$. Then

$$\begin{aligned} p(\bar{\mathrm{t}}) &= p(\bar{\mathrm{u}})p(\mathrm{v}) \\ &= p(\bar{\mathrm{u}}^{-1})p(\mathrm{v}^{-1}) \\ &= p(\bar{\mathrm{t}}^{-1}). \end{aligned}$$

Let $a = \bar{\mathrm{t}}(b) = \mathrm{v}(\bar{\mathrm{u}}(b))$ and $c = \bar{\mathrm{u}}(b)$ (and so $a = \mathrm{v}(c)$). Then, since $b = \bar{\mathrm{u}}^{-1}(c)$ and $c = \mathrm{v}^{-1}(a)$, $b = \bar{\mathrm{u}}^{-1}(\mathrm{v}^{-1}(a)) = \bar{\mathrm{t}}^{-1}(a)$. Thus, $\bar{\mathrm{t}}^{-1}$ is an inverse sequence to $\bar{\mathrm{t}}$.

If $\bar{t}\sigma \in \mathbf{P}$, its *prefix inverse sequence* is defined as $\bar{t}^{-1}\sigma$ (and since σ maps instances to themselves, $\bar{t}^{-1}\sigma$ is also an inverse sequence of $\bar{t}\sigma$). Then

$$\begin{aligned} P^*(a \mid b) &= \sum_{\substack{\bar{t} \in \mathbf{P} \\ \bar{t}(b)=a}} p(\bar{t}) \\ &= \sum_{\substack{\bar{t}^{-1} \in \mathbf{P} \\ \bar{t}^{-1}(a)=b}} p(\bar{t}^{-1}) \\ &= P^*(b \mid a). \end{aligned}$$

Effectively, each possible transformation program may be replaced by its prefix inverse sequence with equal probability, and so the similarity function is symmetric.

◊

Having defined the K* theory and its properties, the following section describes its application to some example domains.

4.3 Applications of K* Theory

The examples presented in this section illustrate potential approaches to developing K* distance functions, and provide a foundation for the K* instance-based classifier described in Chapter 5. Many of these examples illustrate methods for dealing with problems identified in the weather domain discussed in Chapter 1 and Chapter 3. It is important to note that the particular models for instance transformations used in these examples are by no means the only (or best) models. These transformation models may be useful in some domains, while in other domains different models may be more applicable. In many of the following examples multiple transformation models are discussed.

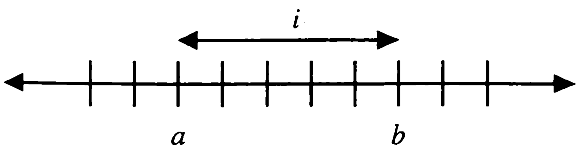

Figure 4.1: Discrete instance positions

4.3.1 Discrete Infinite Space (Integers)

In this example we are interested in determining the similarity between two integers. Let the set of instances **I** be the integers (positive and negative). There are three transformations in the set **T**: σ the end of string marker; and **left** and **right**, which respectively subtract one and add one. The probability of a string of transformations is the product of the probability of the individual transformations,

$$p(\bar{\mathrm{t}}) = \prod_i p(\mathrm{t}_i), \text{ where } \bar{\mathrm{t}} = \mathrm{t}_1, \ldots \mathrm{t}_n .$$

The probability of the stop symbol σ is set to the (arbitrary) value s and $p(\textbf{left}) = p(\textbf{right}) = \frac{1-s}{2}$. This probability assignment satisfies the preconditions for Theorem 4.11 and Theorem 4.12.

The shortest transformation program that transforms a to b and terminates consists of i **right** symbols (or **left** symbols if $a>b$) followed by the stop symbol σ, where $i = |b-a|$ (see Figure 4.1). The probability of the shortest transformation from a to b is therefore

$$\mathrm{P}(b \mid a) = \left(\frac{1-s}{2}\right)^i s .$$

As P(b | a) depends only on the absolute difference between a and b, we can abuse our notation slightly and write P(i).

To generate alternative programs, additional **left** symbols may be inserted anywhere in the shortest transformation string, and provided that each has a corresponding **right** symbol added, the new transformation string will still map a to b. Adding k symbol pairs yields $\binom{2k+i}{k}$ valid transformation strings. Considering all possible mutations to the shortest transformation gives

$$P_{3_\infty}{}^*(i) = \left(\frac{1-s}{2}\right)^i s \sum_{k\geq 0} \binom{2k+i}{k} \left(\frac{1-s}{2}\right)^{2k}$$

(P_{3_∞} denotes that the domain of P is all integers.)

The sum on the right hand side has a closed form using the following generating function identities (from Graham, Knuth and Patashnik, page 203)

$$\sum_{k\geq 0} \binom{2k+i}{k} z^k = \frac{\beta_2(z)^i}{\sqrt{1-4z}}, \text{ where } \beta_2(z) = \frac{1-\sqrt{1-4z}}{2z}.$$

Thus,

$$\sum_{k\geq 0} \binom{2k+i}{k} \left(\frac{1-s}{2}\right)^{2k} = \sum_{k\geq 0} \binom{2k+i}{k} \left(\frac{(1-s)^2}{4}\right)^k$$

$$= \frac{1}{\sqrt{1-4\left(\frac{1-2s+s^2}{4}\right)}} \left(\frac{1-\sqrt{1-4\left(\frac{1-2s+s^2}{4}\right)}}{2\left(\frac{(1-s)^2}{4}\right)}\right)^i$$

$$= \frac{1}{\sqrt{2s-s^2}} \left(\frac{2\left(1-\sqrt{2s-s^2}\right)}{(1-s)^2}\right)^i.$$

Returning to P*:

$$\mathrm{P}_{3\infty}{}^*(i)=\left(\frac{1-s}{2}\right)^i s\frac{1}{\sqrt{2s-s^2}}\left(\frac{2\left(1-\sqrt{2s-s^2}\right)}{(1-s)^2}\right)^i$$
$$=\frac{s}{\sqrt{2s-s^2}}\left(\frac{1-\sqrt{2s-s^2}}{1-s}\right)^i$$

It will prove helpful for later developments to assign $c=\frac{s}{\sqrt{2s-s^2}}$ and $m=\ln(1-s)-\ln\left(1-\sqrt{2s-s^2}\right)$, enabling the probability to be re-expressed as

$$\mathrm{P}_{3\infty}{}^*(i)=ce^{-mi}. \tag{4.13}$$

Figure 4.2 shows the probability of an instance at position 0 transforming to other positions on the line. When the probability of the stop symbol is set to $\frac{1}{2}$, there is a high probability of finishing at position 0. The three most probable transformation programs are "σ" with probability $\frac{1}{2}$, "**left right** σ" and "**right left** σ" both with probability $\frac{1}{32}$; the contribution from longer programs drops off rapidly. Similarly, the probability of finishing at positions further away from the start position also decreases rapidly. With $p(\sigma)=\frac{1}{100}$ the distribution over the final positions becomes more uniform because long programs are more likely.

The K* distance function is obtained by taking the log of the P* probability function,

$$\mathrm{K}_{3\infty}{}^*(i)=\tfrac{1}{2}\log_2\left(2s-s^2\right)-\log_2(s)+i\left(\log_2(1-s)-\log_2\left(1-\sqrt{2s-s^2}\right)\right).$$

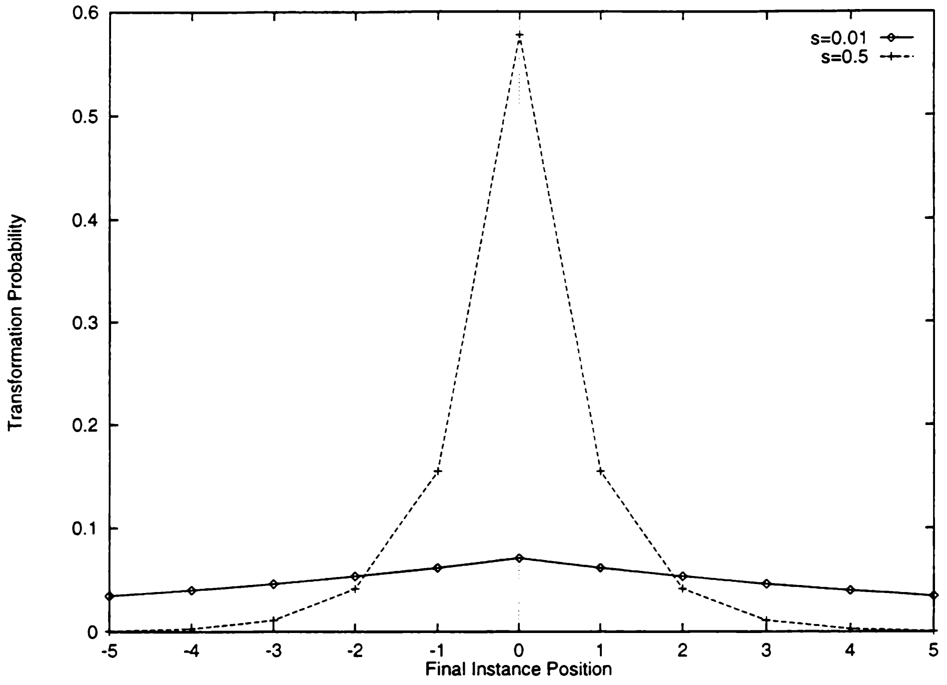

Figure 4.2: P* probabilities for the discrete line

That is, the distance (in bits) is proportional to the absolute difference between two instances. Figure 4.3 plots the distance functions corresponding to the probability functions in Figure 4.2. As mentioned previously, the distance from position 0 to itself is non-zero.

The set of transformations chosen above is not determined by the theory. Other transformation sets (with different distance functions) could be envisaged, but this formulation seems to capture the idea that all points are equivalent, and that space is "invariant" under left and right shifts. It is easy to imagine a situation where the probability of the **left** and **right** transformations should be different, perhaps to provide a distance measure between points in a slowly flowing river. The probability of randomly arriving at a point downstream would be higher than arriving at a point upstream. This

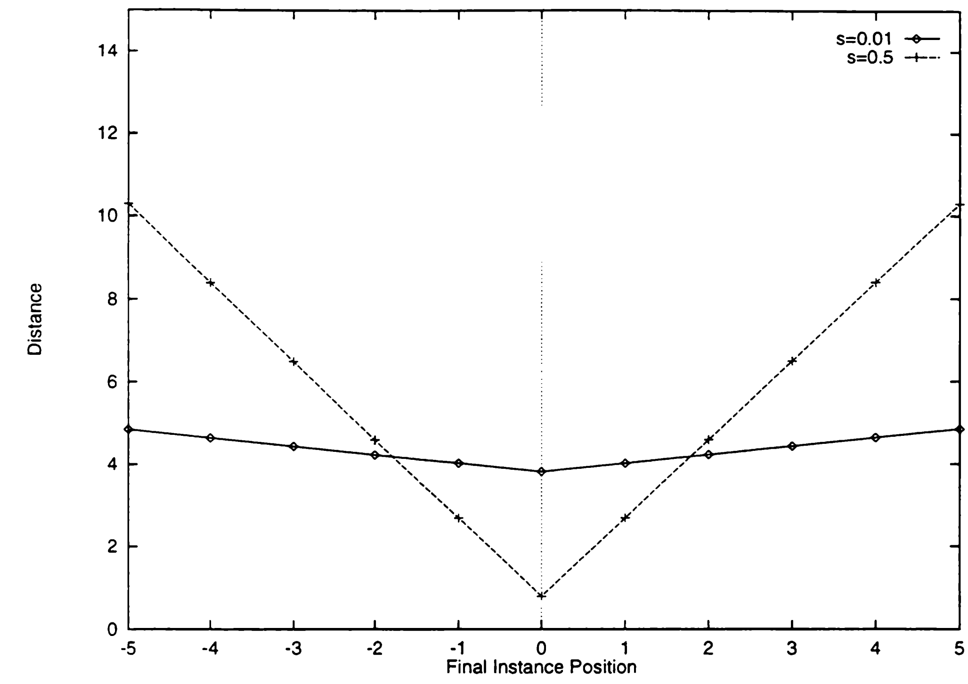

Figure 4.3: K* distance for the discrete line

asymmetry may be useful for the weather domain, where weather pressure systems generally move from west to east and not from east to west.

Following the same general development as above, let $p(\textbf{left}) = l$ and $p(\textbf{right}) = r$, so that $s + l + r = 1$. Since the measure will be asymmetric, we return to the P* $(b \mid a)$ notation.

$$\begin{aligned}
P_{\mathfrak{I}_\infty}{}^*(b \mid a) &= ((b-a>0)?r:l)^{|b-a|} s \sum_{k \ge 0} \binom{2k+|b-a|}{k} r^k l^k \\
&= ((b-a>0)?r:l)^{|b-a|} s \frac{1}{\sqrt{1-4rl}} \left(\frac{1-\sqrt{1-4rl}}{2rl} \right)^{|b-a|} \\
&= \frac{s}{\sqrt{1-4rl}} \left(\frac{1-\sqrt{1-4rl}}{2((b-a>0)?l:r)} \right)^{|b-a|},
\end{aligned}$$

where E?A:B is a ternary operator that takes the value A if expression E is true and B otherwise.

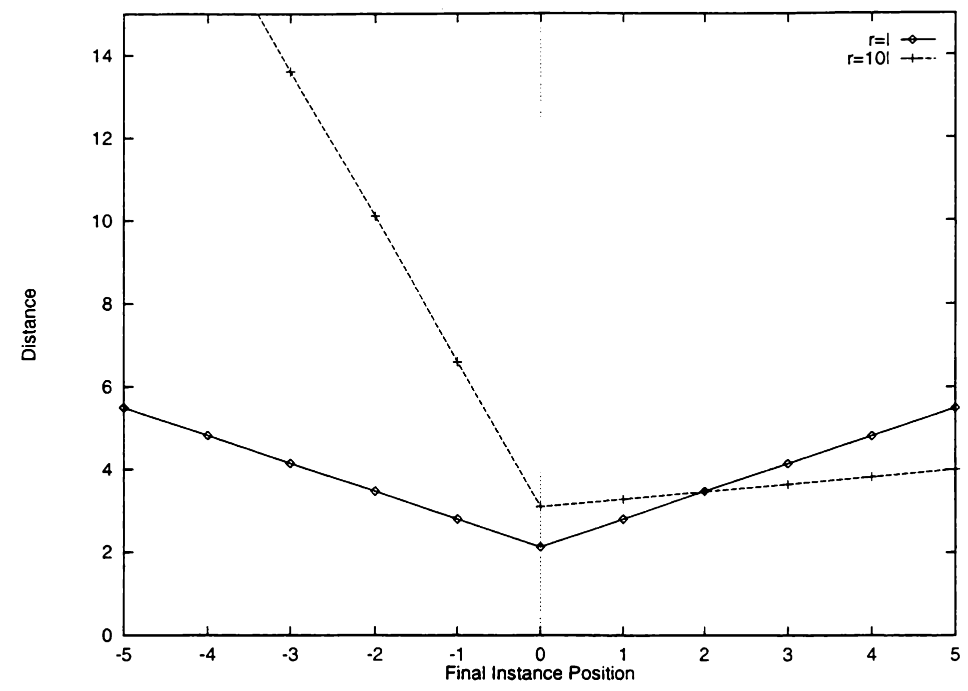

Figure 4.4: Asymmetric K* distance function

Figure 4.4 graphs the resulting distance function when the probability of the **right** transformation is ten times as high as the **left** transformation, in comparison to when the probabilities for these transformations are equal. As expected, the distance from the starting point to a position to the left is greater than to the same distance to the right.

In this development it was assumed the transformation probabilities were constant throughout the length of each program. By setting the probability of the stop symbol to 0 for the first *N* program instructions, and 1 thereafter, it is possible to obtain the probability of transforming from instance *a* to instance *b* by any program consisting of *N* displacements. This is the problem commonly found in random walk texts (Wax, 1954; Spitzer, 1975).

Let the **left** and **right** instructions each have probability of $\frac{1}{2}$ for the first N instructions and 0 after. The probability of the stop symbol is 0 for the first N instructions and 1 after that. N must be odd or even as i is odd or even. The probability of all programs of length N transforming a to b is

$$\mathrm{P}_{3_\infty}*(i,N)=\binom{N}{\frac{1}{2}(N+i)}\left(\frac{1}{2}\right)^N .$$

Since the most interesting case is when N is large and $i<<N$, the formula can be simplified somewhat. Using Stirling's approximation for log$n!$,

$$\begin{aligned}\log(\mathrm{P}_{3_\infty}*(i,N)) \approx{} & (N+\tfrac{1}{2})\log N \\ & -\tfrac{1}{2}(N+i+1)\log\left(\frac{N}{2}\left(1+\frac{i}{N}\right)\right) \\ & -\tfrac{1}{2}(N-i+1)\log\left(\frac{N}{2}\left(1-\frac{i}{N}\right)\right) \\ & -\tfrac{1}{2}\log 2\pi - N\log 2.\end{aligned}$$

Since $i<<N$, the series expansion for log(1+x) may be applied, to obtain

$$\begin{aligned}\log(\mathrm{P}_{3_\infty}*(i,N)) \approx{} & (N+\tfrac{1}{2})\log N-\tfrac{1}{2}\log 2\pi - N\log 2 \\ & -\tfrac{1}{2}(N+i+1)\left(\log N-\log 2+\frac{i}{N}-\frac{i^2}{2N^2}\right) \\ & -\tfrac{1}{2}(N-i+1)\left(\log N-\log 2-\frac{i}{N}-\frac{i^2}{2N^2}\right).\end{aligned}$$

Simplifying this further,

$$\log(\mathrm{P}_{3_\infty}*(i,N)) \approx -\tfrac{1}{2}\log N+\log 2-\tfrac{1}{2}\log 2\pi-\frac{i^2}{2N} .$$

In other words, for large N, the transformation probability is given by the asymptotic formula

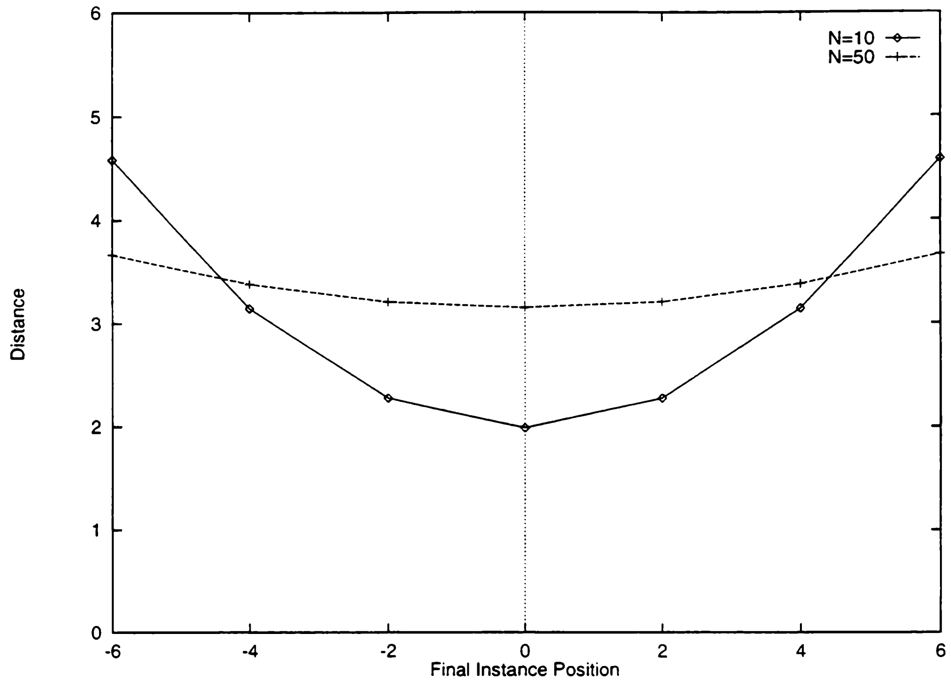

Figure 4.5: K* distance for set-length programs on the discrete line

$$P_{\mathfrak{S}_\infty}{}^*(i,N)=\sqrt{\frac{2}{\pi N}}e^{-\frac{i^2}{2N}}. \qquad (4.14)$$

The resulting K* distance function is shown in Figure 4.5 for $N = 10$ and $N = 50$. In comparison with Figure 4.3, the function is no longer linear but is instead proportional to i^2. Only even values are sampled because, for the values of N chosen, finishing an odd number of positions away is impossible.

The similarity function given in Equation 4.13 assumes that programs may be any length, in contrast to the assumption in Equation 4.14 that programs be the same length. It is important to choose the appropriate function for the domain. One can imagine cases where the above function would be more suitable than Equation 4.13, such as when a constant time interval separates the instances being compared.

Take, for example, a domain where plant growth measurements are compared for fertiliser effectiveness. If the plants are the same age, it is sensible to use Equation 4.14, whereas if plant ages vary, Equation 4.13 is more appropriate. In the first case, one expects many measurements in the same region, so a large difference between two plant measurements is significant. In the second case, one would not expect the measurements to be similar, due to the varying ages of the plants. A large measurement difference is not as significant; it may be due to an age difference between the plants, rather than due to the fertiliser.

4.3.2 Discrete Finite Space

Consider the case where the set of instances is restricted to n integers in the range $0,\ldots,n-1$. The same basic transformations may be used; however some reasonable behaviour must be proposed for transformations at the two edge positions. Two obvious possibilities come to mind: instances may wrap around, so an attempt to move past one edge transforms the instance to the other edge, or instances may "reflect" off an imaginary border. The first case may be useful when comparing modulo instances, such as the days of the year. The second case is less useful, because it turns out the resulting measure is not significantly different from the infinite case. It is also more difficult to imagine a practical situation where this type of model would apply, but for the purpose of comparison both possibilities are examined.

4.3.2.1 Wraparound

Take n positions labelled $0,\ldots,n-1$. Assume that at position 0, possible transformations are to transform left to position $n-1$ and right to position 1, and at position $n-1$ the possible transformations are left to position $n-2$

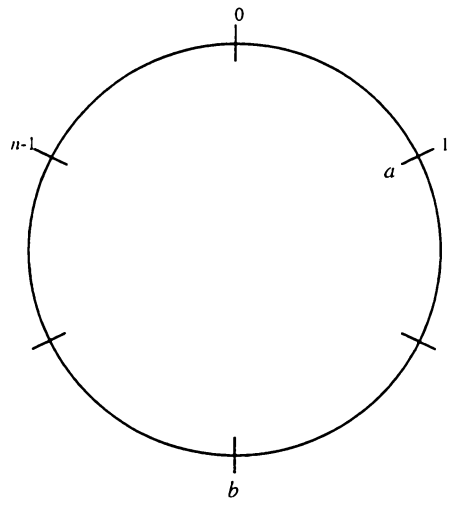

Figure 4.6: Finite positions using wraparound

and right to position 0, as depicted in Figure 4.6. This situation could be useful in obtaining a similarity measure between days of the year. The first day of the year should be equally similar (perhaps with respect to the probability of rain) to the last day of the year as to the second day of the year.

The possible transformations from instance a to instance b can be mapped onto the integers by placing "images" of b on the line wherever integer $i \bmod n = b$ (see Figure 4.7). Any transformation sequence on the integers that transform a to either b or one of the images b' also map a to b in Figure 4.6.

To calculate the probability of transforming from position a to position b, first assume $b \geq a$. Then

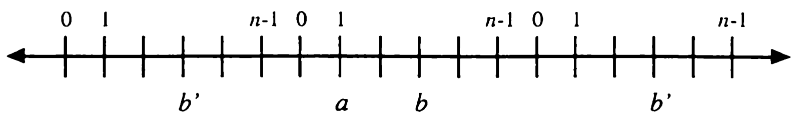

Figure 4.7: Mapping wraparound onto the integers

$$\mathrm{P}_{\mathfrak{Z}_n}{*}(b \mid a) = \sum_{k \geq 0} \mathrm{P}_{\mathfrak{Z}_\infty}{*}(nk + (b-a)) + \sum_{k \geq 0} \mathrm{P}_{\mathfrak{Z}_\infty}{*}(n(k+1) - (b-a)). \quad (4.15)$$

In this equation, the first sum corresponds to the probability of *a* transforming to *b*, as well as to all pseudo-*b*s to the right. The second sum calculates the probability of *a* transforming to all pseudo-*b*s to the left. $\mathrm{P}_{\mathfrak{Z}_\infty}{*}(i) = ce^{-mi}$ is the probability function over all integers from Equation 4.13.

$$\begin{aligned}
\mathrm{P}_{\mathfrak{Z}_n}{*}(b \mid a) &= \sum_{k \geq 0} ce^{-m(nk+(b-a))} + \sum_{k \geq 0} ce^{-m(n(k+1)-(b-a))} \\
&= ce^{-m(b-a)} \sum_{k \geq 0} e^{-mnk} + ce^{-m(n-(b-a))} \sum_{k \geq 0} e^{-mnk} \\
&= c\frac{e^{-m(b-a)} + e^{-m(n-(b-a))}}{1 - e^{-mn}} \\
&= c\frac{e^{m(\frac{n}{2}-(b-a))} + e^{-m(\frac{n}{2}-(b-a))}}{e^{m\frac{n}{2}} - e^{-m\frac{n}{2}}} \\
&= c\frac{\cosh(m(\frac{n}{2} - (b-a)))}{\sinh(m\frac{n}{2})}.
\end{aligned}$$

The development is similar for the case where *b*<*a*. In the following equation the transformation from *a* to *b* itself is moved from the first sum to the second.

$$\mathrm{P}_{\mathfrak{Z}_n}{*}(b \mid a) = \sum_{k \geq 0} \mathrm{P}_{\mathfrak{Z}_\infty}{*}(n(k+1) + (b-a)) + \sum_{k \geq 0} \mathrm{P}_{\mathfrak{Z}_\infty}{*}(nk - (b-a)).$$

Rearranging this to the same form as Equation 4.15 we find

$$\begin{aligned}
\mathrm{P}_{\mathfrak{Z}_n}{*}(b \mid a) &= \sum_{k \geq 0} \mathrm{P}_{\mathfrak{Z}_\infty}{*}(kn + (a-b)) + \sum_{k \geq 0} \mathrm{P}_{\mathfrak{Z}_\infty}{*}((k+1)n - (a-b)) \\
&= c\frac{\cosh(m(\frac{n}{2} - (a-b)))}{\sinh(m\frac{n}{2})}.
\end{aligned}$$

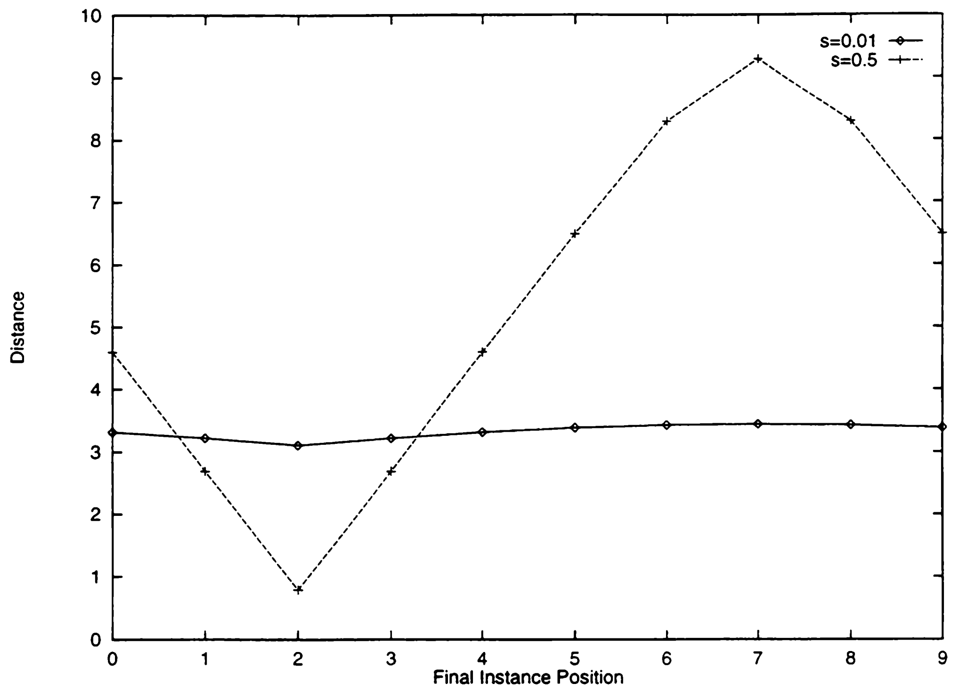

Figure 4.8: K* distance for ten positions using wraparound

For both cases, we can write

$$P_{S_n}^*(b \mid a) = c\frac{\cosh\left(m\left(\frac{n}{2} - |b-a|\right)\right)}{\sinh\left(m\frac{n}{2}\right)}.$$

As with the infinite discrete example, this function depends only on the relative instance positions. Figure 4.8 shows the distance function for an example with ten positions, with the instance initially at position 2. The first point to note is that the distance drops as the final position increases above position 7. This is due to transformation paths that wrap left, past the edge position 0. This effect is more prominent when the stop probability is high ($s = 0.5$) because the distance to positions near the original position is primarily determined by the shortest transformation path. The distance to "opposite" positions, such as 6, 7, and 8, is slightly reduced due to transformation paths from both directions contributing almost equally. When the stop probability is low ($s = 0.01$), this smoothing effect is greater because

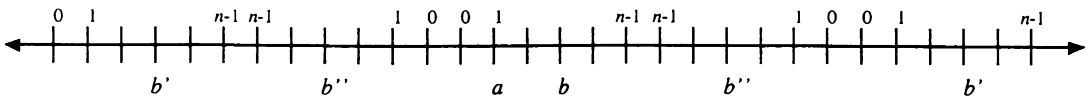

Figure 4.9: Mapping finite positions, with reflection past end points

the bias towards the shortest path is lowered—transformation paths that circle multiple times have more weight in the distance function.

4.3.2.2 Reflecting

Consider the case where an imaginary reflective boundary is placed at each edge of the range of positions. The boundary may be placed either halfway between an end position and where the next position would be, or exactly at the edge positions. The first of these alternatives is examined in detail, the second alternative is treated in Appendix A.

Take n positions labelled $0,\ldots,n-1$. Assume that at position 0, possible transformations are **right** to position 1, and **left** to position 0 (after reflecting off the boundary mid-way through the transformation). At position $n-1$, possible transformations are **left** to position $n-2$, and **right** to position $n-1$ (after reflecting mid-way through the transformation).

Figure 4.9 shows a mapping onto the integers (similar to the previous example) that is valid if $p(\textbf{left}) = p(\textbf{right})$. This condition is required because the behaviour of the **left** and **right** instructions must swap at each reflection. For example, the shortest program transforming instance a to the closest image of b to the left in Figure 4.9 is "**left left left left left** σ", while the corresponding program for Figure 4.6 is "**left left right right right** σ" (a reflection occurs mid-way through the second instruction).

To calculate the probability of transforming from position a to position b $\mathrm{P}_{\mathfrak{S}_n}{}^*(b \mid a)$, first assume $b \geq a$. Then,

$$\mathrm{P}_{\mathfrak{S}_n}{}^*(b \mid a) = \sum_{k\geq 0} \mathrm{P}_{\mathfrak{S}_\infty}{}^*(2nk + (b-a)) + \sum_{k\geq 0} \mathrm{P}_{\mathfrak{S}_\infty}{}^*(2n(k+1) - (b-a))$$
$$+ \sum_{k\geq 0} \mathrm{P}_{\mathfrak{S}_\infty}{}^*(2nk + (b+a+1)) + \sum_{k\geq 0} \mathrm{P}_{\mathfrak{S}_\infty}{}^*(2n(k+1) - (b+a+1)),$$

where $\mathrm{P}_{\mathfrak{S}_\infty}{}^*(i) = ce^{-mi}$ is the probability function over the integers defined in Equation 4.13.

The first sum incorporates the transformation from a to b and all b's to the right. The second sum includes the transformations to all b's to the left. The third sum takes the transformations to all b''s to the left of a, and the fourth sum takes the transformations to all b''s to the right. Following the equation through,

$$\begin{aligned}
\mathrm{P}_{\mathfrak{S}_n}{}^*(b \mid a) &= \sum_{k\geq 0} ce^{-m(2nk+(b-a))} + \sum_{k\geq 0} ce^{-m(2n(k+1)-(b-a))} \\
&+ \sum_{k\geq 0} ce^{-m(2nk+(b+a+1))} + \sum_{k\geq 0} ce^{-m(2n(k+1)-(b+a+1))} \\
&= ce^{-m(b-a)} \sum_{k\geq 0} e^{-m2nk} + ce^{-m(2n-(b-a))} \sum_{k\geq 0} e^{-m2nk} \\
&+ ce^{-m(b+a+1)} \sum_{k\geq 0} e^{-m2nk} + ce^{-m(2n-(b+a+1))} \sum_{k\geq 0} e^{-m2nk} \\
&= c\frac{e^{-m(b-a)} + e^{-m(2n-(b-a))} + e^{-m(b+a+1)} + e^{-m(2n-(b+a+1))}}{1-e^{-m2n}} \\
&= c\frac{e^{m(n-(b-a))} + e^{-m(n-(b-a))} + e^{m(n-(b+a+1))} + e^{-m(n-(b+a+1))}}{e^{mn}-e^{-mn}} \\
&= c\frac{\cosh(m(n-(b-a))) + \cosh(m(n-(b+a+1)))}{\sinh(mn)}.
\end{aligned}$$

The case when $b < a$ is similar, and the following expression holds for both cases:

$$P_{\mathfrak{S}n}^{*}(b \mid a) = c \frac{\cosh(m(n - |b - a|)) + \cosh(m(n - (b + a + 1)))}{\sinh(mn)}.$$

Figure 4.10 plots the resulting distance function for a case with ten discrete positions and stop probability $s = 0.1$. This distance function is dependent on the absolute instance positions, so plots are shown for instances starting at positions 0, 2, and 4. Under this transformation model we expect some effects caused by the edge positions. What we find is that the distance function is almost linear except near the edge positions. For final positions near edges, transformation programs that go a few steps further and reflect off the boundary contribute to the distance measure. For final positions away from edges more transformations are needed to reflect off a boundary and return so the contribution from such programs is negligible.

The second alternative has the reflecting boundary placed exactly on the edge positions. The derivation is fairly straightforward, and is presented in Appendix A. However, it turns out that a special-case function is required when the destination position is exactly at one of the edges. Figure 4.11 illustrates the behaviour of this distance function. Other than at edge positions, the function is similar to that shown in Figure 4.10. However, the distance to actual edge positions is much higher. Consider a billiard table as a physical analogy. Assume we can measure the position of a ball to the nearest centimetre. On average (for randomly sized tables) the size of the interval containing the edge of the table will be half a centimetre in size. If a ball is given an initial push of random strength, the ball is approximately half as likely to stop in the region against the edge of the table as in the adjacent full-centimetre wide interval.

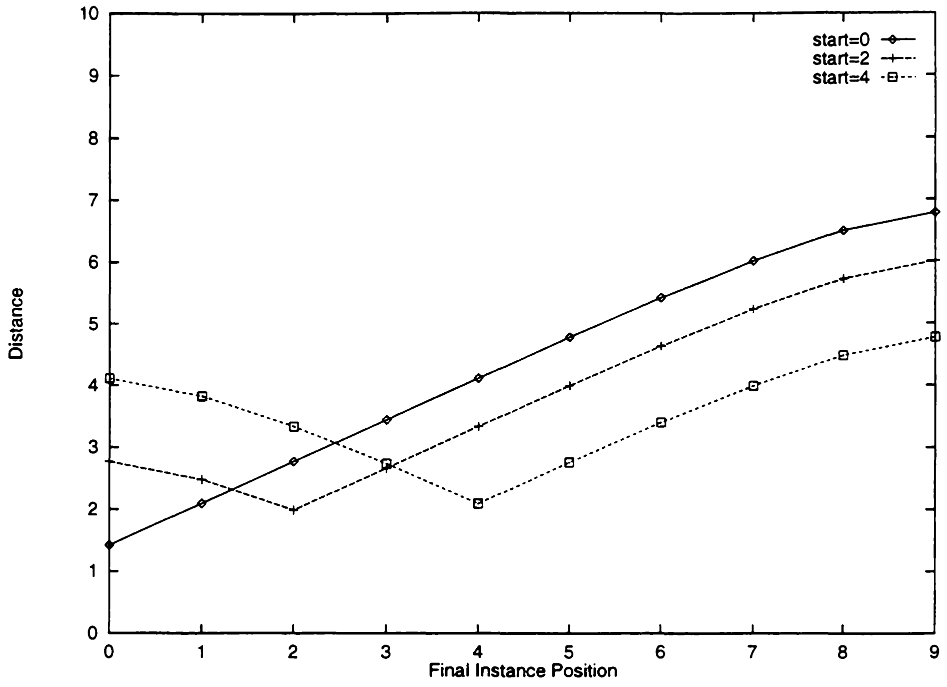

Figure 4.10: K* distance for ten positions using reflection past end points

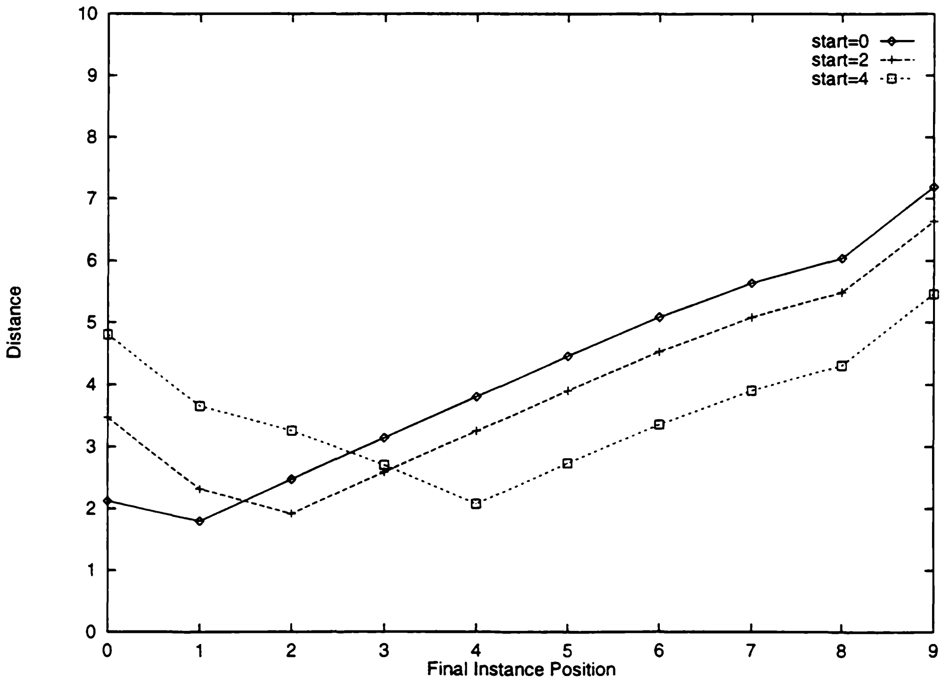

Figure 4.11: K* distance for ten positions using reflection at end points

4.3.3 Continuous Space (Reals)

The function developed for discrete space may be reformulated for real numbers by making the assumption that underlying the real space is a discrete space with very small distances between the discrete instances. Transformation strings between two real numbers will then be very long, so the first step is to examine the expressions *c* and *m* from Equation 4.13 in the limit as *s* approaches 0. This gives

$$\begin{aligned}
P_{\mathfrak{I}\infty}*(i) &= \frac{s}{\sqrt{2s-s^2}}e^{-i\left(\ln(1-s)-\ln\left(1-\sqrt{2s-s^2}\right)\right)} \\
&= \sqrt{\frac{s}{2-s}}e^{-i\left(\left[-s-\frac{s^2}{2}-\frac{s^3}{3}-\dots\right]-\left[-\sqrt{2s-s^2}-\frac{2s-s^2}{2}-\frac{\sqrt{2s-s^2}^3}{3}-\dots\right]\right)} \\
&\approx \sqrt{s/2}\,e^{-i\sqrt{2s}}.
\end{aligned}$$

Note: The above function can also be derived from Equation 4.14 as follows. Equation 4.14 is the probability of transforming from *a* to *b* (*i* positions apart) when all programs involve *N* transformations. This can be extended to include programs of all lengths by multiplying Equation 4.14 by the probability of a program stopping after *N* transformations, and summing over all lengths.

$$\begin{aligned}
P_{\mathfrak{I}\infty}*(i) &= \sum_N P_{\mathfrak{I}\infty}*(i,N)(1-s)^N s \\
&= \sum_N \sqrt{\frac{2}{\pi N}}e^{-\frac{i^2}{2N}}(1-s)^N s,
\end{aligned}$$

where the sum is over all positive integers that are either odd or even as *i* is odd or even. This can be approximated by dividing by 2, and taking the integral from 0 to infinity.

$$\begin{aligned}
\mathrm{P}_{\mathfrak{I}_\infty}*(i) &\approx \tfrac{1}{2}\sum_N \sqrt{\frac{2}{\pi N}}\, e^{-\frac{i^2}{2N}} (1-s)^N s \\
&\approx \frac{s}{\sqrt{2\pi}} \int_0^\infty \sqrt{\tfrac{1}{N}}\, e^{-\frac{i^2}{2N}} (1-s)^N\, dN \\
&= \frac{s}{\sqrt{2\pi}} \int_0^\infty \sqrt{\tfrac{1}{N}}\, e^{N\ln(1-s) - \frac{i^2}{2N}}\, dN \\
&= \frac{s}{\sqrt{-2\ln(1-s)}}\, e^{-i\sqrt{-2\ln(1-s)}} \\
&\approx \frac{s}{\sqrt{2s-s^2}}\, e^{-i\sqrt{2s-s^2}} \\
&\approx \sqrt{s/2}\, e^{-i\sqrt{2s}}\,.
\end{aligned}$$

◊

This can be reformulated as a probability density function where the probability that an integer between i and $i+\Delta i$ will be generated is

$$\mathrm{P}_{\mathfrak{I}_\infty}*(i) = \sqrt{s/2}\, e^{-i\sqrt{2s}}\, \Delta i\,, \qquad (4.16)$$

which can be re-scaled in terms of a real value x where $\frac{x}{x_0} = i\sqrt{2s}$, resulting in the probability density function P* over the reals

$$\mathrm{P}_{\mathfrak{R}_\infty}*(x) = \tfrac{1}{2x_0}\, e^{-x/x_0}\, dx\,. \qquad (4.17)$$

In this formulation, x_0 functions as a scale length; for example, it is the mean expected value for x over the distribution P*. For different applications it is necessary to choose a reasonable value for x_0. There are some rough guidelines about how to do this. For example, if the instances have a measurement error, x_0 should probably not be smaller than the standard deviation of the errors on the measurement. The next chapter specifies a technique for choosing x_0 values.

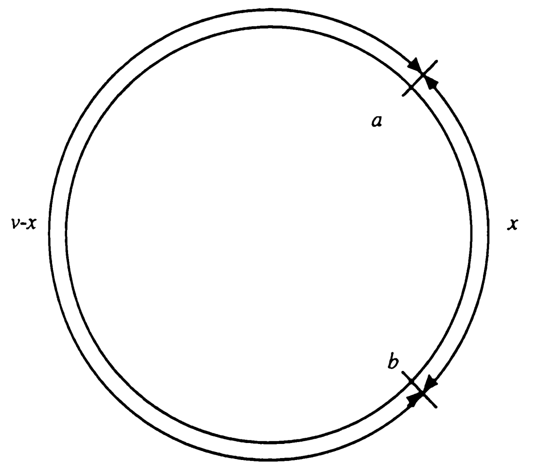

Figure 4.12: Continuous space wrapped

4.3.4 Continuous Space, Wrapped

The similarity function developed in Section 4.3.2.1 for modulo attributes would prove useful for comparing days of the year, because these are essentially discrete. For continuous modulo attributes, such as time of day or longitude (which could be stated to any accuracy) we should use a measure designed for continuous attributes. Consider the space represented by the perimeter of a circle; this may be treated as a case of one-dimensional continuous space wrapped. Let the circumference be of length v. Let a and b be two points on the perimeter of the circle (separated by a distance x in one direction, and $v - x$ in the other). This situation is represented in Figure 4.12.

Let $P_{\Re\infty}*$ be a probability function for one-dimensional infinite continuous space (Equation 4.16). The probability of the shortest path between a and b is

$$P_{\Re v}(x) = \max\left(P_{\Re\infty}*(x), P_{\Re\infty}*(v - x)\right).$$

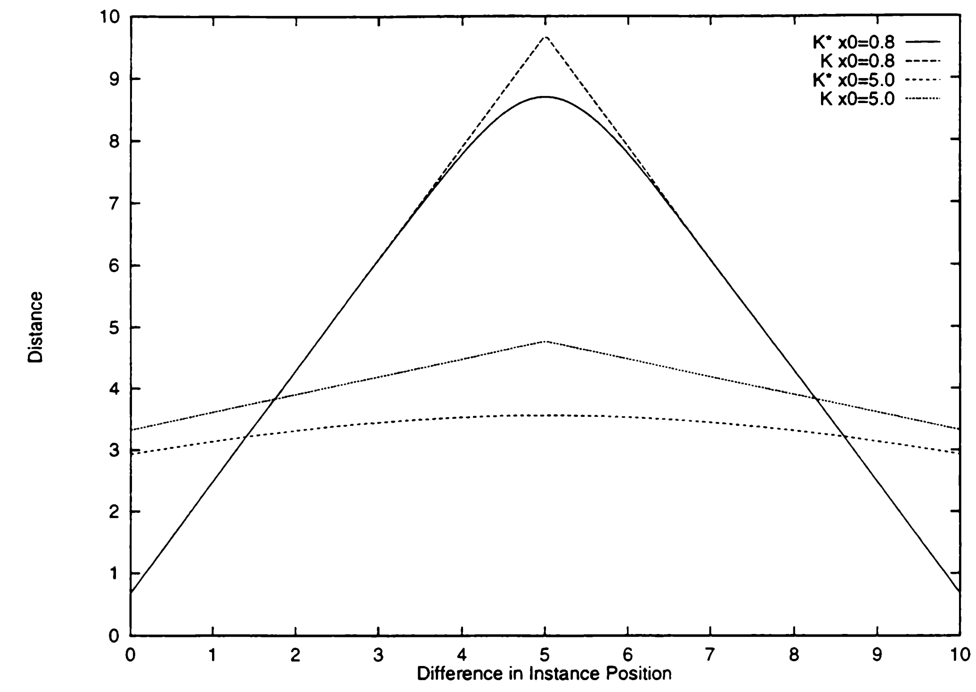

Figure 4.13: Distance functions for continuous space wrapped

The probability of all paths between a and b, allowing any number of trips around the entire circumference in either direction is

$$P_{\Re_v}{}^*(x) = \sum_{k\geq 0} P_{\Re_\infty}{}^*(vk + x) + \sum_{k\geq 0} P_{\Re_\infty}{}^*(v(k+1) - x).$$

Substituting $P_{\Re_\infty}{}^*(x) = ce^{-mx}$, where $c = \dfrac{1}{2x_0}$ and $m = \dfrac{1}{x_0}$ from the one-dimensional continuous probability function of Equation 4.16, the development is similar to that in Section 4.3.2.1:

$$\begin{aligned}
P_{\Re_v}{}^*(x) &= \sum_{k\ge 0} ce^{-m(vk+x)} + \sum_{k\ge 0} ce^{-m(v(k+1)-x)} \\
&= ce^{-mx}\sum_{k\ge 0} e^{-mvk} + ce^{-m(v-x)}\sum_{k\ge 0} e^{-mvk} \\
&= c\frac{e^{-mx} + e^{-m(v-x)}}{1-e^{-mv}} \\
&= c\frac{e^{m(\frac{v}{2}-x)} + e^{-m(\frac{v}{2}-x)}}{e^{m\frac{v}{2}} - e^{-m\frac{v}{2}}} \\
&= c\frac{\cosh\left(m\left(\frac{v}{2}-x\right)\right)}{\sinh\left(m\frac{v}{2}\right)}.
\end{aligned}$$

Taking the log gives the corresponding K* distance function

$$\begin{aligned}
K_{\Re_v}{}^*(x) &= \log_2\left(\sinh\left(\tfrac{mv}{2}\right)\right) - \log_2\left(\cosh\left(m\left(\tfrac{v}{2}-x\right)\right)\right) - \log_2(c) \\
&= \log_2\left(\sinh\left(\tfrac{v}{2x_0}\right)\right) - \log_2\left(\cosh\left(\tfrac{v}{2x_0} - \tfrac{x}{x_0}\right)\right) + \log_2(2x_0).
\end{aligned}$$

Figure 4.13 shows a comparison between the K* distance function and the "shortest path" distance function, K. The circumference of the circle is ten. When the scale length x_0 is set to 0.8, the K* function is determined almost entirely by the shortest programs, and hence behaves like the K function. For small differences in instance positions the function is almost linear. When the final instance position is opposite the initial position, both the shortest "anticlockwise" program and the shortest "clockwise" program have similar contributions, resulting in a smoothing of the curve. When x_0 is set to 5, programs that transform the instance around the entire circumference are more likely and the distance function makes little distinction between the possible final positions.

4.3.5 Continuous Space, Clipped

Clipping boundaries are another interesting problem. We define a clipping boundary as a point on the line beyond which we are uncertain as to an

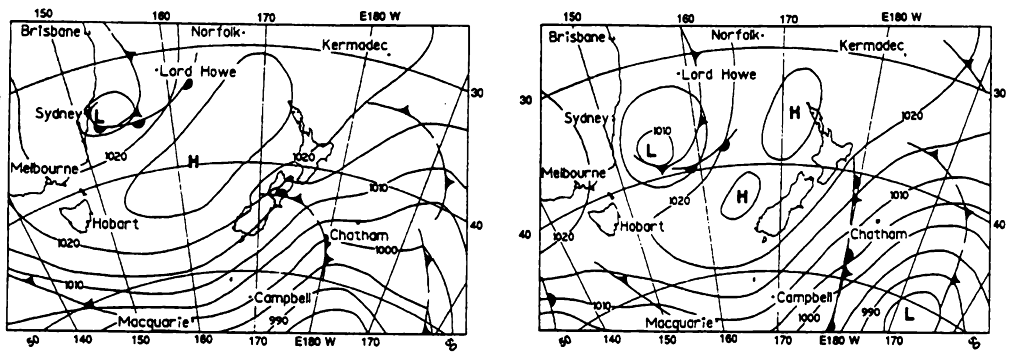

Figure 4.14: Two weather maps taken 24 hours apart

instance's exact position. This type of situation might occur when a measurement for some feature is beyond the capabilities of the measuring apparatus (such as an off-scale pressure reading). In the case of weather maps, clipping occurs as features move off the image borders. For example, the lower right of the second map of Figure 4.14 contains a low-pressure system not visible in the first map; the low-pressure system is still present in the first map, but it has been clipped so its position cannot be determined exactly.

Although the precise position of an instance beyond the clipping boundary is unknown, we can ask questions like how likely the instance is to move from a known position to somewhere past the clipping boundary, and where past the boundary the instance is most likely to be. This problem is also related to that of missing values as discussed in Chapter 3—missing values could be viewed in this context as cases where the clipping region covers the entire instance space. Section 4.3.10 deals with the problem of missing values further.

As a simple example, consider the probability that a point to one side of a clipping boundary will transform to the other side of the clipping boundary. This situation is represented in Figure 4.15. Let instance a be at distance d_0

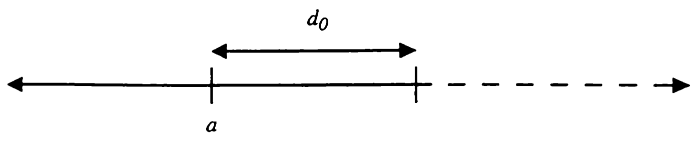

Figure 4.15: A clipping boundary on the line

from the clipping boundary. The probability of a transforming to any point beyond the clipping boundary is

$$P^*(?\,|\,a) = \int_{d_0}^{\infty} P_{\Re_\infty}{}^*(x)\,dx,$$

where '?' denotes the clipping region, and $P_{\Re_\infty}{}^*$ is a probability function for one-dimensional infinite continuous space. Substituting in the function from Equation 4.16,

$$\begin{aligned} P^*(?\,|\,a) &= \int_{d_0}^{\infty} \tfrac{1}{2x_0} e^{-x/x_0}\,dx \\ &= -\tfrac{1}{2} e^{-x/x_0} \Big]_{d_0}^{\infty} \\ &= \tfrac{1}{2} e^{-d_0/x_0}. \end{aligned}$$

It is possible to calculate the single instance position that corresponds to this probability:

$$\begin{aligned} &\frac{1}{2x_0} e^{-x/x_0} = \frac{1}{2} e^{-d_0/x_0} \\ \Rightarrow\ &\frac{1}{x_0} e^{-x/x_0} = e^{-d_0/x_0} \\ \Rightarrow\ &-x/x_0 = \ln(x_0) - d_0/x_0 \\ \Rightarrow\ &x = d_0 - x_0 \ln(x_0). \end{aligned}$$

That is, the probability of *a* moving to any point beyond the clipping border is the same as the probability of *a* moving to a point *x* away.

If is known that an instance has transformed to beyond the clipping boundary, its expected position past the boundary is

$$\begin{aligned}
&\int_{d_0}^{\infty} \frac{x e^{-x/x_0}}{x_0 e^{-d_0/x_0}} dx \\
&= -\left(x + x_0\right) e^{-x+d_0/x_0} \Big]_{d_0}^{\infty} \\
&= d_0 + x_0 .
\end{aligned}$$

4.3.6 Continuous Space, Splitting Points

A problem often faced in complex domains occurs when there are multiple features of the same basic type that may differ in number between the two objects being compared. One domain where this type of problem arises is weather map comparison; the number of high-pressure and low-pressure systems may be different in the maps being compared. Another domain is when determining likely evolutionary paths of plant populations—a description of the current state would include multiple populations of plant species (each with their own size, locality, and physical characteristics). The task is to determine which initial configurations of parent populations are most likely to give rise to the current distribution. The number of populations in the current state will probably be more than in the proposed initial state due to speciation. A natural way to deal with this type of problem is to introduce a "split" transformation. In this section we consider simple examples where instances consist of points on the real line.

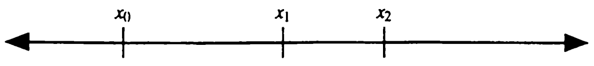

Figure 4.16: One Point Transforming to Two Points

4.3.6.1 Transforming One Instance to Two

As a simplification of this problem, consider the probability of one instance on the real line transforming to two instances at different positions.

Let the initial instance be x, and the two final instances be x_1 and x_2, as shown in Figure 4.16. One way to model this situation is to allow x to split into two instances at some point during the transformation, with one of the new instances transforming to each of x_1 and x_2. This can be treated as three separate sub-transformations, with the behaviour of the basic transformations changing during the course of the transformation:

1) A transformation sequence taking x to some intermediate position y and which is terminated by the stop transformation. This first occurrence of the stop instruction results in x splitting in two, and further transformations affect only one of the parts. (Under this model there will be only one split transformation.) The probability of this transformation is $\mathrm{P}_{\Re_\infty}{*}(|x-y|)$, where $\mathrm{P}_{\Re_\infty}{*}$ is a probability function for one-dimensional infinite continuous space.

2) A transformation sequence taking the first of the instances at y to x_1 and which is terminated by the stop transformation. This second occurrence of the stop instruction results in further transformations affecting the

second instance. The probability of transforming the first instance from y to x_1 is $\mathrm{P}_{\Re\infty}*(|x_1 - y|)$.

3) A transformation sequence taking the other instance to x_2, terminated by the stop transformation (which signals the end of transformations). The probability of transforming the second instance from y to x_2 is $\mathrm{P}_{\Re\infty}*(|x_2 - y|)$.

In order to consider all transformation paths, we must also integrate over all possible intermediate split positions,

$$\mathrm{P}*(x_1, x_2 \mid x) = \int_{-\infty}^{\infty} \mathrm{P}_{\Re\infty}*(|x - y|)\mathrm{P}_{\Re\infty}*(|x_1 - y|)\mathrm{P}_{\Re\infty}*(|x_2 - y|)dy .$$

The first case to consider is where both of the final instances are to one side of the initial instance. To deal with the absolute values above the integral is split into parts; this is simplified by assuming assume that the initial instance x is at 0. Substituting $\mathrm{P}_{\Re\infty}*(x) = ce^{-mx}$, where $c = \dfrac{1}{2x_0}$ and $m = \dfrac{1}{x_0}$ from the one-dimensional continuous probability function of Equation 4.17 gives

$$\begin{aligned}\mathrm{P}*(x_1, x_2 \mid 0) = &\int_{-\infty}^{0} ce^{my} ce^{-m(x_1 - y)} ce^{-m(x_2 - y)} dy \\ &+ \int_{0}^{x_1} ce^{-my} ce^{-m(x_1 - y)} ce^{-m(x_2 - y)} dy \\ &+ \int_{x_1}^{x_2} ce^{-my} ce^{-m(y - x_1)} ce^{-m(x_2 - y)} dy \\ &+ \int_{x_2}^{\infty} ce^{-my} ce^{-m(y - x_1)} ce^{-m(y - x_2)} dy.\end{aligned}$$

Carrying out the integrations yields

$$
\begin{aligned}
P^*(x_1, x_2 \mid 0) &= \frac{c^3}{3m} e^{-m(x_1 - x_2)} + \frac{c^3}{m}\left(e^{-mx_2} - e^{-m(x_1+x_2)}\right) \\
&+ \frac{c^3}{m}\left(e^{-mx_2} - e^{-m(2x_2 - x_1)}\right) + \frac{c^3}{m} e^{-m(2x_2 - x_1)} \\
&= \frac{2c^3}{3m}\left(3e^{-mx_2} - e^{-m(x_1+x_2)} - e^{-m(2x_2 - x_1)}\right) \\
&= \frac{1}{12x_0}\left(3e^{-\frac{x_2}{x_0}} - e^{-\frac{x_1 + x_2}{x_0}} - e^{-\frac{2x_2 - x_1}{x_0}}\right).
\end{aligned}
$$

A similar function is obtained for cases where $x_1 > x_2$. The upper right quadrant of Figure 4.17 shows the contours of equal dissimilarity for the corresponding distance function, with scale factor x_0 set to 1. Notice that dissimilarity is largely dependent upon whichever of the two final instances is furthermost. The function has the same value when $x_1 = x = 0$ as when $x_1 = x_2$. In the former case, significant contributing programs involve x splitting almost immediately, when very little transformation is required for one part to reach x_1. In the latter case, programs with high probability involve x transforming near to x_2 before splitting, leaving the new instances near x_1 and x_2. Both cases involve one instance transforming the distance from x to x_2.

Consider the case where x_1 and x_2 are on opposite sides of x. Again, it is assumed that x is at 0, and that $x_1 > x_2$. Following a similar development to above,

$$\begin{aligned}
P^*(x_1, x_2 \mid 0) = &\int_{-\infty}^{x_1} ce^{my} ce^{-m(x_1-y)} ce^{-m(x_2-y)} dy \\
&+ \int_{x_1}^{0} ce^{my} ce^{-m(y-x_1)} ce^{-m(x_2-y)} dy \\
&+ \int_{0}^{x_2} ce^{-my} ce^{-m(y-x_1)} ce^{-m(x_2-y)} dy \\
&+ \int_{x_2}^{\infty} ce^{-my} ce^{-m(y-x_1)} ce^{-m(y-x_2)} dy.
\end{aligned}$$

Carrying through the integration gives

$$\begin{aligned}
P^*(x_1, x_2 \mid 0) = &\frac{c^3}{3m} e^{-m(x_2-2x_1)} \\
&+ \frac{c^3}{m}\left(e^{-m(x_2-x_1)} - e^{-m(x_2-2x_1)}\right) \\
&+ \frac{c^3}{m}\left(e^{-m(x_2-x_1)} - e^{-m(2x_2-x_1)}\right) \\
&+ \frac{c^3}{m} e^{-m(2x_2-x_1)} \\
= &\frac{2c^3}{3m}\left(3e^{-m(x_2-x_1)} - e^{-m(x_2-2x_1)} - e^{-m(2x_2-x_1)}\right) \\
= &\frac{1}{12x_0}\left(3e^{-\frac{x_2-x_1}{x_0}} - e^{-\frac{x_2-2x_1}{x_0}} - e^{-\frac{2x_2-x_1}{x_0}}\right).
\end{aligned}$$

The upper left quadrant of Figure 4.17 shows contours of equal dissimilarity for the corresponding distance function, with scale factor x_0 set to 1. In contrast to the upper right quadrant, dissimilarity is primarily dependent on the total distance between x_1 and x_2. Major contributing programs in this case involve the original instance x splitting almost immediately, leaving two distances to be transformed over, from x to x_1 and from x to x_2. The total distance is independent of the exact position of x between x_1 and x_2, although

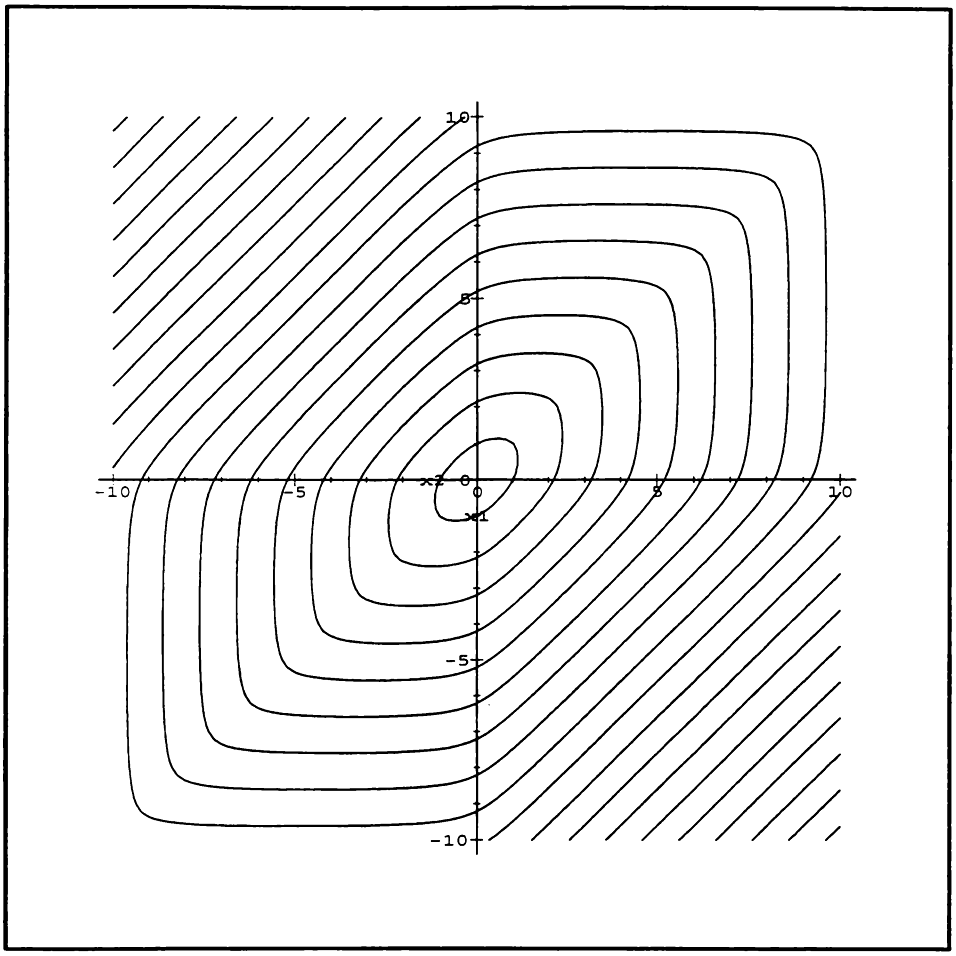

Figure 4.17: Contours of equal dissimilarity (combined function)

if the distances are small in relation to the scale factor x_0, dissimilarity is slightly lower when both distances are approximately equal.

The functions developed above (along with those for the cases when $x_2<0$) may be combined piecewise to give the final similarity function, $P^*(x_1, x_2 \mid x)$. Contours of equal dissimilarity for the corresponding distance function are shown in Figure 4.17. The positions of instances x_1 and x_2 may be swapped without changing the similarity, and their positions may be reflected about x without changing the similarity.

4.3.6.2 Transforming One Instance to Three

The above results are applicable to the problem of where one instance x transforms to three instances, x_1, x_2, and x_3. Instead of allowing only one split transformation to occur, two splits are required (or alternatively, a single ternary split, which we will not consider for simplicity). The original instance x first transforms to an intermediate position y and splits. Now one instance must transform to one of the final positions, and the other instance must transform to the other two final positions. There are three combinations to consider, one for each final destination of the instance that undergoes a single split transformation. The destination in each case is specified by an instruction with probability $\frac{1}{3}$. The transformation probability for two instances at position y transforming to x_1, x_2, and x_3 is

$$\tfrac{1}{3}\mathrm{P}^*(x_1 \mid y)\mathrm{P}^*(x_2,x_3 \mid y)+\tfrac{1}{3}\mathrm{P}^*(x_2 \mid y)\mathrm{P}^*(x_1,x_3 \mid y)+\tfrac{1}{3}\mathrm{P}^*(x_3 \mid y)\mathrm{P}^*(x_2,x_1 \mid y),$$

where $\mathrm{P}^*(x \mid y)$ is a probability function for an instance y transforming to an instance x (from Equation 4.17), and $\mathrm{P}^*(x_1,x_2 \mid y)$ is a probability function for an instance y transforming to two instances x_1 and x_2 (developed in Section 4.3.6.1).

As in Section 4.3.6.1, all possible initial split positions must be considered, so the probability function is

$$\begin{aligned}P^*(x_1,x_2,x_3 \mid x) = &\int_{-\infty}^{\infty} \tfrac{1}{3} P^*(y \mid x) P^*(x_1 \mid y) P^*(x_2,x_3 \mid y)\,dy \\ &+ \int_{-\infty}^{\infty} \tfrac{1}{3} P^*(y \mid x) P^*(x_2 \mid y) P^*(x_1,x_3 \mid y)\,dy \\ &+ \int_{-\infty}^{\infty} \tfrac{1}{3} P^*(y \mid x) P^*(x_3 \mid y) P^*(x_2,x_1 \mid y)\,dy.\end{aligned}$$

We will leave the function at this point, although it is not difficult to carry out the integrations piecewise as in the previous section. In principle this approach can be used to derive similarity functions transforming one instance to higher numbers of instances.

4.3.6.3 Transforming *n* Instances to *m* Instances

Given a series of probability functions for transforming one instance to multiple instances, it is possible to formulate a similarity measure for an arbitrary number of instances to some higher number of instances. As indicated by the developments above, it is impossible to derive a single analytical expression for the similarity function, so instead an algorithm is outlined. Assume n original instances $x_1,\ldots,x_n$, and m final instances $y_1,\ldots,y_m$, where $n < m$. The calculation has two steps.

1) Determine the set of all possible instance mappings, where each of the initial instances maps to at least one final instance (and each final instance has only one initial instance mapping to it). The number of such mappings is given by the recursive function

$$\begin{aligned}&\text{mappings}(1,m) = 1\\ &\text{mappings}(n,m) = \sum_{i=1}^{m-(n-1)} \binom{m}{i} \text{mappings}(n-1,m-i).\end{aligned}$$

2) Sum the transformation probabilities for each possible instance mapping. For a given mapping, let M_i be the number of final instances that initial instance x_i maps to, and $y_1,\ldots,y_{M_i}$ be those final instances. The transformation probability for the mapping is

$$\frac{1}{\mathrm{mappings}(n,m)}\prod_{i=1}^{n}\mathrm{P}^*\left(y_1,\ldots,y_{M_i} \mid x_i\right),$$

where $\mathrm{P}^*\left(y_1,\ldots,y_{M_i} \mid x_i\right)$ is a probability function for one instance transforming to M_i instances as described in the previous section.

This example illustrates one potential method for treating different numbers of instances. These models utilise a new instance transformation—the "split" operation. For domains such as the weather it also makes sense to introduce a "merge" transformation. With both split and merge transformations, the next step is to model multiple merge/split operations. For example, two high-pressure systems may merge as their paths cross, and split as the paths diverge. These types of problems are more complicated, but can be handled within the general framework.

4.3.7 Symbolic Space (Independent Symbols)

One advantage of the K* approach is that both numeric attributes and symbolic attributes can be handled within the same framework. To deal with symbolic attributes consider a set $\mathbf{S}$ of instances that occur with probabilities p_a, $a \in \mathbf{S}$; the transformations allowed on instances are the transmutation of any instance to any other instance. In this example, it is assumed that the probability of transmuting to an instance is independent of the current instance. The probability of not transforming an instance (the end of string instruction) is assigned probability s and the probability of a transformation to instance a to be $(1-s)p_a$ (regardless of the current instance).

The probability of the shortest string that transforms symbol a to symbol b, where $a \neq b$ is

$$\mathrm{P}(b \mid a) = (1-s)p_b s.$$

The probability of all programs allowing one intermediate transformation to another symbol, c, is

$$\begin{aligned}\mathrm{P}'(b \mid a) &= \sum_{c \in \mathrm{S}} (1-s)p_c(1-s)p_b s \\ &= (1-s)(1-s)p_b s.\end{aligned}$$

Summing over all possible intermediate transformations gives

$$\begin{aligned}\mathrm{P_S}^*(b \mid a) &= (1-s)p_b s \sum_{k \geq 0} (1-s)^k \\ &= (1-s)p_b s\left(\frac{1}{1-(1-s)}\right) \\ &= (1-s)p_b.\end{aligned}$$

In the special case where $a = b$, there is also the shorter transformation string which simply consists of the stop symbol. The final symbolic probability function is given by

$$\mathrm{P_S}^*(b \mid a) = \begin{cases} (1-s)p_b & \text{if } a \neq b \\ s+(1-s)p_b & \text{if } a = b. \end{cases} \tag{4.18}$$

The probability s here is analogous to the probability s (and the equivalent x_0) in the developments above. That is, some reasonable value must be chosen for s depending on the data being modelled. When $s = 1$, this function behaves the same as the overlap metric for symbolic values discussed in Chapter 3.

4.3.8 Symbolic Space (Non-independent Symbols)

The following development assumes that the probability of transforming to an instance is not independent of the current instance. This is the case for many real domains; for example, if instances represent traffic light colours, the probability of transforming to red depends on whether the light is currently green or amber. Let S be the set of instances. Define $\mathrm{P}(b \mid a)$ as the probability of instance a transmuting directly to instance b, where $a, b \in \mathrm{S}$ (this is called the *one-step* probability). Basic transformation instructions include the stop symbol σ (with probability s), and the symbols required to specify changes between instances (denoted as "$a \to b$", and assigned probability $(1-s)\mathrm{P}(b \mid a)$).

A transformation program consists of a number of change instructions terminated by the stop symbol. For example, if $\mathrm{S} = \{\alpha, \beta, \chi\}$, the following programs specify a transformation from instance α to instance β:

"$\alpha \to \beta\sigma$",

with probability $(1-s)\mathrm{P}(\beta \mid \alpha)s$;

"$\alpha \to \alpha\alpha \to \chi\chi \to \beta\sigma$",

with probability $(1-s)\mathrm{P}(\alpha \mid \alpha)(1-s)\mathrm{P}(\chi \mid \alpha)(1-s)\mathrm{P}(\beta \mid \chi)s$.

The possible programs can partitioned by their length, and $\mathrm{P}^n(b \mid a)$ defined as the probability of all programs with length n that transform instance a to instance b. Thus,

$$\begin{aligned}
\mathrm{P}^1(b \mid a) &= \begin{cases} 0 & \text{if } a \neq b \\ s & \text{if } a = b \end{cases} \\
\mathrm{P}^2(b \mid a) &= (1-s)\mathrm{P}(b \mid a)s \\
\mathrm{P}^3(b \mid a) &= \sum_{c \in \mathrm{S}} (1-s)\mathrm{P}(c \mid a)\mathrm{P}^2(b \mid c) \\
\mathrm{P}^n(b \mid a) &= \sum_{c \in \mathrm{S}} (1-s)\mathrm{P}(c \mid a)\mathrm{P}^{n-1}(b \mid c).
\end{aligned}$$

The P* function that considers all programs of all lengths is

$$\mathrm{P_S}^*(b \mid a) = \sum_{k \geq 0} \mathrm{P}^k(b \mid a).$$

This can be expressed using matrix notation. Let **I** denote the identity matrix. Let all $\mathrm{P}(b \mid a)$ form the elements of a matrix **P**; that is, element $\mathbf{P}_{ab} = \mathrm{P}(b \mid a)$ (and similarly $\mathbf{P}^*_{ab} = \mathrm{P_S}^*(b \mid a)$), then

$$\begin{aligned}
\mathbf{P}^* &= s\sum_{k \geq 0}(1-s)^k \mathbf{P}^k \\
&= s\mathbf{I} + s\sum_{k \geq 1}(1-s)^k \mathbf{P}^k \\
&= s\mathbf{I} + s(1-s)\mathbf{P}\sum_{k \geq 1}(1-s)^{k-1} \mathbf{P}^{k-1} \\
&= s\mathbf{I} + (1-s)\mathbf{P}\mathbf{P}^* \qquad (4.19) \\
\mathbf{P}^* - (1-s)\mathbf{P}\mathbf{P}^* &= s\mathbf{I} \\
(\mathbf{I} - (1-s)\mathbf{P})\mathbf{P}^* &= s\mathbf{I} \\
\mathbf{P}^* &= (\mathbf{I} - (1-s)\mathbf{P})^{-1} s\mathbf{I}.
\end{aligned}$$

Element $\mathbf{P}^*_{ab}$ is the probability of symbol a transforming to symbol b, considering all possible transformation paths.

It turns out that this result is a good general tool for problems with a finite number of instances. For example, the results obtained in Section 4.3.2 can be duplicated by inserting appropriate transformation probabilities into the initial matrix **P**. For the simple examples in Section 4.3.2, reasonable probabilities are obvious; however, choosing probabilities for other domains may be more difficult. One common situation is where the set of instances is found in association with another set of instances (for example, a "class" attribute that the similarity function should be sensitive to). The following method is one way to choose transformation probabilities.

Assume a set of classes $\mathbf{C}$ is found in association with the instances. It is possible to calculate the frequency of each class $c \in \mathbf{C}$ given instance $s \in \mathbf{S}$, $\mathrm{P}_{\mathrm{CS}}(c \mid s)$, and also the frequency of each instance given a class, $\mathrm{P}_{\mathrm{SC}}(s \mid c)$. One way to define the probability of instance a transmuting to instance b is as the probability of a transforming to some class c and then transforming from c to b, summed over all classes:

$$\mathrm{P}(b \mid a) = \sum_{c \in \mathbf{C}} \mathrm{P}(c \mid a)\mathrm{P}(b \mid c). \tag{4.20}$$

This method is analogous to the method for assigning symbol similarities used by Stanfill and Waltz (1986) in the Value Difference Metric. The benefit of this type of method for assigning basic transformation probabilities is that the resulting function is sensitive to the distribution of the classes. For example, assume the set of instances contains the alphabetic characters and we wish to assign basic transformation probabilities specifying the likelihood of one letter transforming to another in the context of English text. In the Brown corpus (Francis and Kucera, 1982), characters occur with the frequencies shown in Figure 4.18. However, using these frequencies in conjunction with the similarity function of Section 4.3.7 is not a good solution, because the characters are not independent in their usage.

0.178 space	0.103 E	0.076 T	0.066 A	0.062 O
0.060 I	0.058 N	0.054 S	0.050 R	0.045 H
0.034 L	0.033 D	0.026 C	0.022 U	0.021 M
0.019 F	0.017 P	0.016 G	0.015 W	0.014 Y
0.013 B	0.008 V	0.005 K	0.002 X	0.001 Z
0.001 Q	0.001 J			

Figure 4.18: Ranked character frequencies from the Brown corpus

If the "class" instances are taken as the characters that appear immediately following occurrences of the current character, transformation probabilities can be assigned using Equation 4.20. The results for some characters are shown in Table 4.1 (the full table is provided in Appendix B). These are the one-step probabilities that form the matrix **P** in Equation 4.19—that is, they do not consider multiple letter transformations, or the possibility of not transforming at all. The letter 'A' is most likely to transform to (in order), a space character, the vowels 'E', 'A', 'I', 'O', the consonant 'N', and the vowel 'U'. The letter 'B' is most likely to transform to the space character, the consonants 'H', 'R', 'T', and 'L'. It is relatively unlikely that a 'B' will transform to a 'B', primarily because the letter occurs infrequently (as seen in Figure 4.18). Similarly, the space character is often the most likely character to transform to. Interestingly, there are half a dozen characters that the letter 'D' is more likely to transform to than the space character, presumably because the characters that 'D' precede are rarely preceded by the space character (for example, the letter 'D' is often the last letter of a word, whereas the space character never is). In this simple example the assigned probabilities have extracted information about the grouping of vowels and consonants based only on the characters they precede.

A	E	I	B	C	D
0.190 sp	0.178 **E**	0.197 sp	0.133 sp	0.191 sp	0.138 E
0.132 E	0.121 sp	0.142 **I**	0.090 H	0.135 T	0.091 S
0.131 **A**	0.085 A	0.125 A	0.071 R	0.065 H	0.087 T
0.113 I	0.075 O	0.116 E	0.069 T	0.059 R	0.074 **D**
0.093 O	0.070 N	0.094 O	0.066 L	0.057 **C**	0.072 N
0.055 N	0.067 I	0.058 N	0.059 O	0.051 S	0.067 R
0.040 U	0.058 S	0.038 U	0.059 E	0.046 E	0.063 sp
0.037 S	0.052 T	0.037 R	0.052 A	0.041 N	0.061 H
0.034 R	0.044 R	0.031 S	0.043 S	0.041 I	0.042 O
0.030 T	0.044 D	0.026 T	0.037 N	0.039 A	0.041 L
0.030 L	0.030 L	0.021 L	0.037 C	0.038 L	0.036 Y
0.016 D	0.027 U	0.019 H	0.036 **B**	0.036 O	0.033 F
0.015 P	0.023 Y	0.018 C	0.035 I	0.027 W	0.033 A
0.015 C	0.023 H	0.011 P	0.034 M	0.026 M	0.025 M
0.011 H	0.021 F	0.011 M	0.031 D	0.023 P	0.025 G
0.010 G	0.018 G	0.011 D	0.029 P	0.023 D	0.020 I
0.010 F	0.015 P	0.009 F	0.021 F	0.021 F	0.018 C
0.010 B	0.014 M	0.008 W	0.019 V	0.019 G	0.016 W
0.009 M	0.011 C	0.008 G	0.018 W	0.018 B	0.014 P
0.007 Y	0.009 W	0.007 B	0.018 U	0.014 U	0.012 B
0.006 W	0.007 B	0.006 Y	0.018 G	0.012 V	0.011 V
0.004 K	0.005 K	0.004 K	0.007 K	0.008 Y	0.009 U
0.001 X	0.002 V	0.003 V	0.006 Y	0.005 K	0.009 K
0.001 V	0.001 X	0.001 Z	0.004 Q	0.003 J	0.001 Z
0.000 Z	0.000 Z	0.001 X	0.004 J	0.001 Z	0.001 X
0.000 Q	0.000 Q	0.001 J	0.002 Z	0.001 X	0.001 Q
0.000 J	0.000 J	0.000 Q	0.001 X	0.001 Q	0.001 J

Table 4.1: Ranked one-step character transformation probabilities

Table 4.2 shows the corresponding **P*** probabilities obtained from Equation 4.19 when the stop probability is 0.2 (the full matrix is given in Appendix B). Because the **P*** matrix includes programs that stop before making any transformations, the probability of a character transforming to itself is much higher than in Table 4.1. The lower the stop probability, the more the transformation distributions will resemble the distribution in Figure 4.18.

This method for assigning one-step probabilities could also be used in ordered domains (that is, each instance has definite neighbours to which it may transform), but the transformation probabilities are unknown. Given an

A	E	I	B	C	D
0.266 A	0.296 E	0.264 I	0.214 B	0.226 C	0.234 D
0.146 sp	0.131 sp	0.148 sp	0.133 sp	0.144 sp	0.118 sp
0.089 E	0.057 A	0.086 E	0.074 E	0.073 T	0.089 E
0.059 I	0.056 T	0.065 A	0.061 T	0.071 E	0.064 T
0.057 O	0.053 O	0.057 O	0.049 A	0.047 A	0.050 S
0.050 T	0.050 I	0.050 T	0.049 O	0.044 O	0.049 N
0.046 N	0.049 N	0.047 N	0.045 H	0.043 I	0.046 A
0.039 S	0.044 S	0.038 S	0.045 R	0.043 N	0.045 O
0.037 R	0.039 R	0.037 R	0.043 I	0.043 S	0.044 R
0.028 H	0.031 H	0.029 H	0.043 N	0.042 R	0.040 H
0.026 L	0.028 D	0.024 L	0.041 S	0.040 H	0.039 I
0.022 D	0.026 L	0.021 D	0.033 L	0.028 L	0.029 L
0.022 U	0.019 U	0.021 U	0.026 D	0.025 D	0.019 C
0.018 C	0.018 C	0.018 C	0.023 C	0.018 M	0.018 F
0.014 F	0.016 F	0.014 M	0.019 M	0.016 F	0.018 M
0.014 M	0.015 M	0.013 F	0.017 U	0.016 U	0.016 Y
0.013 P	0.013 G	0.012 P	0.016 F	0.015 P	0.015 G
0.011 G	0.013 P	0.011 G	0.016 P	0.015 W	0.015 U
0.010 W	0.013 Y	0.011 W	0.013 G	0.013 G	0.013 P
0.010 Y	0.011 W	0.010 Y	0.013 W	0.011 B	0.013 W
0.009 B	0.009 B	0.009 B	0.010 Y	0.010 Y	0.010 B
0.005 V	0.005 V	0.005 V	0.009 V	0.008 V	0.007 V
0.004 K	0.004 K	0.004 K	0.005 K	0.004 K	0.005 K
0.001 J	0.001 J	0.001 J	0.002 J	0.001 J	0.001 J
0.001 Q	0.001 Q	0.001 Q	0.001 Q	0.001 Q	0.001 Q
0.001 X	0.001 X	0.001 X	0.001 X	0.001 X	0.001 X
0.001 Z	0.001 Z	0.001 Z	0.001 Z	0.001 Z	0.001 Z

Table 4.2: Ranked P* character transformation probabilities

instance s_i with neighbours s_{i-1} and s_{i+1}, the association of classes with s_i, $P_{CS}(c \mid s_i)$ is calculated as above. The association of instances with classes $P_{SC}(s \mid c)$ is calculated only for the neighbouring instances s_{i-1} and s_{i+1}. The transformation probability can then be calculated using Equation 4.20.

4.3.9 Multiple Attributes

To compute a distance between instances with more than one attribute is conceptually straightforward. The set of transformations on the combined attributes can be taken as the union of the transformations for the individual attributes. Two potential methods for modelling transformation strings are immediately obvious.

The first method (called the *additive* method) is to sequentially transform the first attribute, then the second attribute and so on until all attributes are transformed. The resulting probability for the total string is the product of the probabilities of the individual strings, so the distance is the sum of the distances for the individual attributes. Removing the restriction of transforming the attributes in order and considering all attribute orderings produces the same result, because the transformation programs must then be prefixed with an instruction specifying the attribute transformation order. However, the restriction of transforming an entire attribute at one time is arbitrary.

The second method (called the *merge* method) allows transformations on any attribute in any order—the probability of transforming from instance a to instance b is the probability of stopping at b when taking an unconstrained random walk along all attributes, starting at a. The merge method is more difficult to calculate because a single similarity function must be derived for all the attributes, rather than combining the results of the individual attribute similarity functions. The next section considers examples of this nature.

4.3.9.1 Two Symbolic Attributes

This example extends the model used for independent symbols in Section 4.3.7 to two attributes. Consider two sets of symbols, $\mathbf{S}_1$ and $\mathbf{S}_2$ with their associated occurrence frequencies. An instance is represented as a pair of symbols (a,b), with $a \in \mathbf{S}_1$ and $b \in \mathbf{S}_2$.

Additive Method

The following similarity function is obtained directly from Section 4.3.7:

$$\mathrm{P}_{\mathrm{S}^2}*((a,b)|(c,d)) = \begin{cases} (1-s)p_a(1-s)p_b & \text{if } a \neq c, b \neq d \\ ((1-s)p_a + s)(1-s)p_b & \text{if } a = c, b \neq d \\ (1-s)p_a((1-s)p_b + s) & \text{if } a \neq c, b = d \\ ((1-s)p_a + s)((1-s)p_b + s) & \text{if } a = c, b = d. \end{cases}$$

In this example the probability of the stop instruction is the same for both attributes. This need not be the case—if one attribute is more likely to undergo transformations, the probability of its stop instruction could be lowered.

Merge Method

The probability of the end of string instruction is s. Let the probability of a transformation of the first attribute to symbol a equal $c_1 p_a$, and the probability of the second attribute transforming to symbol b equal $c_2 p_b$. In this example, $c_1 = c_2 = \frac{1-s}{2}$ (in any case, c_1 and c_2 should sum to $(1-s)$). The probability of transforming from (c,d) to (a,b) by any program involving n symbol transformations on the first attribute and m symbol transformations on the second attribute is

$$\mathrm{P}_{\mathrm{S}^2}{}^{n,m}((a,b)|(c,d)) = \binom{n+m}{n}\left(\frac{1-s}{2}\right)^n\left(\frac{1-s}{2}\right)^m p_a p_b s \qquad (n>0, m>0)$$

$$\mathrm{P}_{\mathrm{S}^2}{}^{0,m}((a,b)|(c,d)) = \left(\frac{1-s}{2}\right)^m p_b s \qquad (a=c, m>0)$$

$$\mathrm{P}_{\mathrm{S}^2}{}^{n,0}((a,b)|(c,d)) = \left(\frac{1-s}{2}\right)^n p_a s \qquad (n>0, b=d)$$

$$\mathrm{P}_{\mathrm{S}^2}{}^{0,0}((a,b)|(c,d)) = s. \qquad (a=c, b=d)$$

Consider the case where $a \neq c$ and $b \neq d$. The probability of transforming from (c,d) to (a,b) by any program involving N total symbol transformations is

$$
\begin{aligned}
\mathrm{P}_{\mathrm{S}^2}{}^{N}((a,b) \mid (c,d)) &= \sum_{n=1}^{N-1} \mathrm{P}_{\mathrm{S}^2}{}^{n,N-n}((a,b) \mid (c,d)) \\
&= \sum_{n=1}^{N-1} \binom{N}{n} \left(\frac{1-s}{2}\right)^{n} \left(\frac{1-s}{2}\right)^{N-n} p_a p_b s \\
&= \left(\left(\sum_{n=0}^{N} \binom{N}{n} \left(\frac{1-s}{2}\right)^{n} \left(\frac{1-s}{2}\right)^{N-n} \right) - \left(\frac{1-s}{2}\right)^{N} - \left(\frac{1-s}{2}\right)^{N} \right) p_a p_b s \\
&= \left((1-s)^{N} - 2\left(\frac{1-s}{2}\right)^{N} \right) p_a p_b s.
\end{aligned}
$$

The probability when considering programs of all possible lengths (which must include at least two symbol transformations in this case) is

$$
\begin{aligned}
\mathrm{P}_{\mathrm{S}^2}{}^{*}((a,b) \mid (c,d)) &= \sum_{N \geq 2} \mathrm{P}_{\mathrm{S}^2}{}^{N}((a,b) \mid (c,d)) \\
&= \sum_{N \geq 2} \left((1-s)^{N} - 2\left(\frac{1-s}{2}\right)^{N} \right) p_a p_b s \\
&= \frac{(1-s)^2 p_a p_b}{1+s}.
\end{aligned}
$$

The case where $a = c$ and $b \neq d$ requires the addition of transformation programs do not transform the first attribute. In this case the total transformation probability is

$$\begin{aligned}
\mathrm{P}_{\mathrm{s}^2}{}^*((a,b)\,|\,(c,d)) &= \frac{(1-s)^2 p_a p_b}{1+s} + \sum_{m>0} \mathrm{P}_{\mathrm{s}^2}{}^{0,m}((a,b)\,|\,(c,d)) \\
&= \frac{(1-s)^2 p_a p_b}{1+s} + \sum_{m>0}\left(\frac{1-s}{2}\right)^m p_b s \\
&= \frac{(1-s)^2 p_a p_b}{1+s} + \frac{(1-s)p_b s}{1+s} \\
&= \frac{((1-s)p_a + s)(1-s)p_b}{1+s}.
\end{aligned}$$

The development for the case where $a \neq c$ and $b = d$ yields a similar result. When both $a = c$ and $b = d$, the program consisting of only the stop instruction must also be included; in this case the final probability function is

$$\mathrm{P}_{\mathrm{s}^2}{}^*((a,b)\,|\,(c,d)) = \frac{(1-s)^2 p_a p_b + (1-s)p_b s + (1-s)p_a s}{1+s} + s.$$

The probability functions produced by the additive and merge methods are very similar; the functions are identical when s is 0 or 1. For other values of s, the shortest program that transforms between two identical instances is assigned higher probability by the merge method than the additive method (s versus s^2, respectively). The probability of all other programs is reduced accordingly.

4.3.9.2 Integers in Two Dimensions

Let the set of instances have two integer dimensions. Let i be the number of horizontal positions to be transformed through, and j be the number of vertical positions to be transformed through. Assume for simplicity that both of these are positive. In the following sections, similarity functions are derived for both combination methods.

Additive Method

Assuming the probability of the stop instruction is the same for both attributes, the additive similarity function may be obtained directly from Equation 4.13

$$\begin{aligned} \mathrm{P}_{\mathfrak{S}^2_\infty}{}^*(i, j) &= ce^{-mi}ce^{-mj} \\ &= c^2e^{-m(i+j)}. \end{aligned} \tag{4.21}$$

Since the core of this function is simply $i + j$, this function is analogous to the city block metric described in Chapter 3.

Merge Method

The merged set of transformations contains five members: **left** and **right**, which operate on the horizontal dimension, **up** and **down**, which operate on the vertical dimension, and the stop symbol σ. Let k be a number of additional **right** instructions, beyond the minimum needed to get to i—there must also be k additional **left** instructions to ensure we still finish at i. Let l be a number of additional **up** instructions (similar to k above).

Let $numright = i + k$, $numleft = k$, $numup = j + l$, $numdown = l$. For a given number of these instructions, there are

$$\frac{(numleft + numright + numup + numdown)!}{numleft!\,numright!\,numup!\,numdown!} = \frac{(i + 2k + j + 2l)!}{(i + k)!\,k!\,(j + l)!\,l!}$$

possible programs.

If the probability of the stop symbol is s and the probabilities assigned to each of the other symbols is $\frac{1-s}{4}$, the expression for the sum of the probabilities of all programs stopping at (i, j) is

$$\mathrm{P}_{\mathfrak{S}^2_\infty}{}^*(i, j) = \left(\frac{1-s}{4}\right)^{(i+j)} s \sum_{k\geq 0}\sum_{l\geq 0} \frac{(i + 2k + j + 2l)!}{(i + k)!\,k!\,(j + l)!\,l!} \left(\frac{1-s}{4}\right)^{(2k+2l)}$$

The double sum may be reformulated by summing diagonally. Let $t = k + l$, then the expression is

$$\left(\frac{1-s}{4}\right)^{(i+j)} s\sum_{t\geq 0}\sum_{l=0}^{t}\frac{(i+j+2t)!}{(i+(t-l))!(t-l)!(j+l)!l!}\left(\frac{1-s}{4}\right)^{(2t)} \tag{4.22}$$

Another way of interpreting the problem is in terms of the number of displacements N. The only possible program that stops after 0 displacements is "σ." For $N = 1$, one program terminates at each of $(-1,0)$, $(1,0)$, $(0,-1)$, $(0,1)$. The number of programs that terminate at position (i, j) after N displacements is

$$\binom{N}{\frac{N+(i+j)}{2}}\binom{N}{\frac{N+(i-j)}{2}},$$

and the expression for the sum of the probabilities of all programs stopping at (i, j) is

$$s\sum_{N\geq 0}\binom{N}{\frac{N+(i+j)}{2}}\binom{N}{\frac{N+(i-j)}{2}}\left(\frac{1-s}{4}\right)^{N}. \tag{4.23}$$

Note: Equation 4.22 and Equation 4.23 are identical. If $t = \frac{N-(i+j)}{2}$, Equation 4.23 becomes

$$\left(\frac{1-s}{4}\right)^{(i+j)} s\sum_{t\geq 0}\binom{2t+i+j}{t+i+j}\binom{2t+i+j}{t+i}\left(\frac{1-s}{4}\right)^{2t}. \tag{4.24}$$

Considering only the inner sum in Equation 4.22 and the two binomials in Equation 4.24,

$$\begin{aligned}\sum_{l=0}^{t}\frac{(i+j+2t)!}{(i+(t-l))!(t-l)!(j+l)!l!} &= \sum_{l=0}^{t}\binom{(i+(t-l))+(t-l)+(j+l)+l}{(i+(t-l)),(t-l),(j+l),l} \\ &= \sum_{l=0}^{t}\binom{i+j+2t}{i+j+t+l}\binom{i+j+t+l}{i+j+t}\binom{i+j+t}{j+l} \\ &= \binom{i+j+2t}{i+j+t}\sum_{l=0}^{t}\binom{t}{l}\binom{i+j+t}{j+l} \\ &= \binom{i+j+2t}{i+j+t}\binom{i+j+2t}{t+j}.\end{aligned}$$

◊

To proceed with Equation 4.23, we first derive an expression for the probability of transforming to (i,j) when all programs contain N transformations—that is, the probability assigned to the stop symbol is zero for the first N instructions and 1 afterwards. This gives

$$\mathrm{P}_{\mathfrak{I}^2_\infty}*(i,j,N)=\binom{N}{\frac{N+(i+j)}{2}}\binom{N}{\frac{N+(i-j)}{2}}\left(\frac{1}{4}\right)^N.$$

Note that N here must be odd or even as $i+j$ is odd or even.

Using Stirling's approximation for $\log(n!)$, we have

$$\begin{aligned}\log\left(\mathrm{P}_{\mathfrak{I}^2_\infty}*(i,j,N)\right)\approx{}&(2N+1)\log N-\log 2\pi-N\log 4\\ &-\tfrac{1}{2}(N+i+j+1)\log\left(\frac{N}{2}\left(1+\frac{i+j}{N}\right)\right)\\ &-\tfrac{1}{2}(N-i-j+1)\log\left(\frac{N}{2}\left(1-\frac{i+j}{N}\right)\right)\\ &-\tfrac{1}{2}(N+i-j+1)\log\left(\frac{N}{2}\left(1+\frac{i-j}{N}\right)\right)\\ &-\tfrac{1}{2}(N-i+j+1)\log\left(\frac{N}{2}\left(1-\frac{i-j}{N}\right)\right).\end{aligned}$$

Using the series expansion for $\log(1+x)$, we obtain

$$\begin{aligned}\log\left(\mathrm{P}_{3^2_\infty}{}^*(i,j,N)\right) \approx\ & (2N+1)\log N - \log 2\pi - N\log 4 \\ & -\tfrac{1}{2}(N+i+j+1)\left(\log N - \log 2 + \frac{i+j}{N} - \frac{(i+j)^2}{2N^2}\right) \\ & -\tfrac{1}{2}(N-i-j+1)\left(\log N - \log 2 - \frac{i+j}{N} - \frac{(i+j)^2}{2N^2}\right) \\ & -\tfrac{1}{2}(N+i-j+1)\left(\log N - \log 2 + \frac{i-j}{N} - \frac{(i-j)^2}{2N^2}\right) \\ & -\tfrac{1}{2}(N-i+j+1)\left(\log N - \log 2 - \frac{i-j}{N} - \frac{(i-j)^2}{2N^2}\right).\end{aligned}$$

This may be further simplified, to obtain

$$\log\left(\mathrm{P}_{3^2_\infty}{}^*(i,j,N)\right) \approx \log\left(\frac{2}{N\pi}\right) - \frac{i^2+j^2}{N}$$

$$\mathrm{P}_{3^2_\infty}{}^*(i,j,N) \approx \frac{2}{N\pi} e^{-\frac{i^2+j^2}{N}}.$$

This similarity function is the two dimensional equivalent to that of Equation 4.14. To find the similarity function when the probability of the stop instruction is constant, we use the above result in a procedure like that in Section 4.3.3.

$$\begin{aligned}\mathrm{P}_{3^2_\infty}{}^*(i,j) &= \sum_N \mathrm{P}_{3^2_\infty}{}^*(i,j,N)(1-s)^N s \\ &= \sum_N \frac{2}{N\pi} e^{-\frac{i^2+j^2}{N}} (1-s)^N s,\end{aligned}$$

where the sum is over all positive integers that are either odd or even as $i+j$ is odd or even. This can be approximated by dividing the sum over all integers by two and taking the integral from 0 to infinity.

$$\begin{aligned} P_{3^2_\infty}{}^*(i,j) &\approx \tfrac{1}{2}\sum_N \frac{2}{N\pi} e^{-\frac{i^2+j^2}{N}} (1-s)^N s \\ &\approx \tfrac{s}{\pi}\int_0^\infty \tfrac{1}{N} e^{-\frac{i^2+j^2}{N}} (1-s)^N dN \qquad (4.25) \\ &= \tfrac{2s}{\pi} BesselK\left(0, 2\sqrt{i^2+j^2}\sqrt{-\log(1-s)}\right), \end{aligned}$$

where *BesselK* is the modified Bessel function of the second kind of order 0 (Thomas and Finney, 1988). The central term includes the Euclidean distance function, and so the contours of equal distance will have circular symmetry. Tables of Bessel functions can be computed from series, although in practice an approximation based on Euclidean distance may be more appropriate. Carrying out a series expansion, we find that for large x,

$$\begin{aligned} BesselK(0,x) &= \sqrt{\tfrac{\pi}{2x}} e^{-x}\left(1 - \tfrac{1}{8x} + \tfrac{9}{128x^2} - \tfrac{75}{1024x^3} + \tfrac{3675}{32768x^4} + O\left(\tfrac{1}{x^5}\right)\right) \\ &\approx \sqrt{\tfrac{\pi}{2x}} e^{-x}, \end{aligned}$$

and so Equation 4.25 may be approximated as

$$P_{3^2_\infty}{}^*(i,j) \approx s \frac{1}{\sqrt{\pi\sqrt{i^2+j^2}\sqrt{-\log(1-s)}}} e^{-2\sqrt{i^2+j^2}\sqrt{-\log(1-s)}}.$$

Figure 4.19 compares the contours of equal distance for the distance functions obtained from Equations 4.21 and 4.25 (that is, after we take the log) when the stop probability is set to 0.1. The additive method (shown on the left) results in distances similar to the city block distance, while the merge method gives distances similar to Euclidean distance. However, both of these functions are now measured as probabilities (for similarity), or bits (for distance).

4.3.9.3 One Integer Dimension and One Symbolic Dimension

In this example, instances have one symbolic attribute (values of which come from the set **S**) and one integer attribute (values of which are from the set **I**).

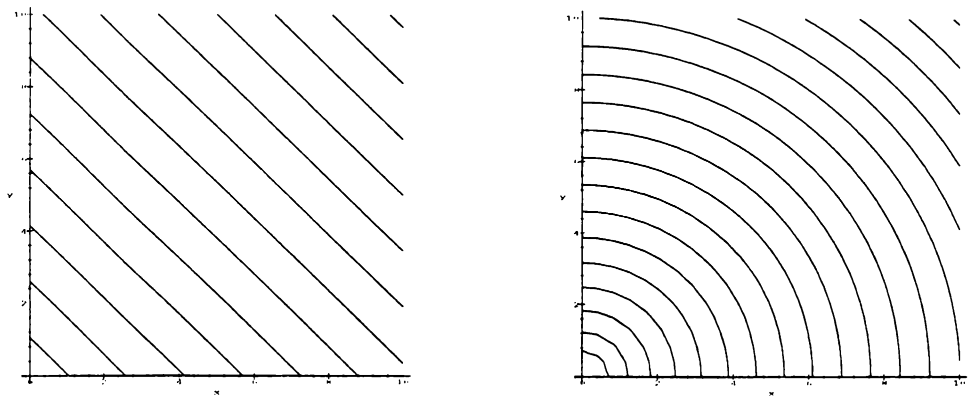
Figure 4.19: Contours of equal distance for additive and merge combination

Instances are represented as an ordered pair (a,b) where $a \in \mathbf{S}$ and $b \in \mathbf{I}$. Let the two instances being compared be (a,b) and (c,d). Let $i = |b - d|$.

Additive Method

Assume the probability of the stop instruction is the same for both attributes. Simple combination of the probability functions from Equation 4.13 and Equation 4.17 gives

$$P_{S\mathfrak{I}_\infty}{}^*((a,b)|(c,d)) = \begin{cases} \dfrac{(1-s)p_a s}{\sqrt{2s-s^2}}\left(\dfrac{1-\sqrt{2s-s^2}}{1-s}\right)^i & \text{if } a \neq c \\ \dfrac{((1-s)p_a + s)s}{\sqrt{2s-s^2}}\left(\dfrac{1-\sqrt{2s-s^2}}{1-s}\right)^i & \text{if } a = c. \end{cases}$$

Merge Method

In the merged transformation set, the **left** and **right** instructions are each assigned probability $\frac{1-s}{4}$. The probability of transforming to a symbol $a \in \mathbf{S}$ is set to $\frac{1-s}{2}p_a$. First we take the case where $a \neq c$. The probability of all programs with k extra **left/right** pairs, and j transformations of the symbolic attribute is

$$\mathrm{P}_{S3_\infty}{}^{k,j}\left((a,b)\,|\,(c,d)\right)=\binom{2k+i+j}{k+i,k,j}\left(\frac{1-s}{4}\right)^{2k+i}\left(\frac{1-s}{2}\right)^{j}p_a s.$$

Summing over all possible lengths of programs gives

$$\begin{aligned}
\mathrm{P}_{S3_\infty}{}^{*}\left((a,b)\,|\,(c,d)\right)&=\sum_{k\ge 0}\sum_{j\ge 1}\binom{2k+i+j}{k+i,k,j}\left(\frac{1-s}{4}\right)^{2k+i}\left(\frac{1-s}{2}\right)^{j}p_a s\\
&=\sum_{k\ge 0}\binom{2k+i}{k}\left(\frac{1-s}{4}\right)^{2k+i}p_a s\sum_{j\ge 1}\binom{2k+i+j}{2k+i}\left(\frac{1-s}{2}\right)^{j}\\
&=\sum_{k\ge 0}\binom{2k+i}{k}\left(\frac{1-s}{4}\right)^{2k+i}p_a s\left(\sum_{j\ge 0}\binom{2k+i+j}{2k+i}\left(\frac{1-s}{2}\right)^{j}-1\right).
\end{aligned}$$

The inner sum can be dealt with by the following generating function identity (from Graham, Knuth and Patashnik, page 199)

$$\sum_{j\ge 0}\binom{n+j}{n}z^{j}=\frac{1}{(1-z)^{n+1}}.$$

Thus

$$\begin{aligned}
\mathrm{P}_{S3_\infty}{}^{*}\left((a,b)\,|\,(c,d)\right)&=\sum_{k\ge 0}\binom{2k+i}{k}\left(\frac{1-s}{4}\right)^{2k+i}p_a s\left(\left(\frac{2}{1+s}\right)^{2k+i+1}-1\right)\\
&=p_a s\sum_{k\ge 0}\binom{2k+i}{k}\left(\frac{1-s}{4}\right)^{2k+i}\left(\frac{2}{1+s}\right)^{2k+i+1}\\
&\quad-p_a s\sum_{k\ge 0}\binom{2k+i}{k}\left(\frac{1-s}{4}\right)^{2k+i}\\
&=p_a s\frac{2}{1+s}\left(\frac{1-s}{2(1+s)}\right)^{i}\sum_{k\ge 0}\binom{2k+i}{k}\left(\frac{1-s}{2(1+s)}\right)^{2k}\\
&\quad-p_a s\left(\frac{1-s}{4}\right)^{i}\sum_{k\ge 0}\binom{2k+i}{k}\left(\frac{1-s}{4}\right)^{2k}.
\end{aligned}$$

Each of the sums can be dealt with in the same manner as the one-dimensional discrete case in Section 4.3.1, to obtain

$$\mathrm{P}_{\mathrm{S3}_\infty}*((a,b)|(c,d)) = p_a\sqrt{s}\left(\frac{1+s-2\sqrt{s}}{1-s}\right)^i - \frac{2p_a s}{\sqrt{3+2s-s^2}}\left(\frac{2-\sqrt{3+2s-s^2}}{1-s}\right)^i.$$

When $a=c$, there are additional programs that do not undergo any transformations on the symbolic attribute. The probability function in this case is

$$\begin{aligned}\mathrm{P}_{\mathrm{S3}_\infty}*((a,b)|(c,d)) &= p_a\sqrt{s}\left(\frac{1+s-2\sqrt{s}}{1-s}\right)^i - \frac{2p_a s}{\sqrt{3+2s-s^2}}\left(\frac{2-\sqrt{3+2s-s^2}}{1-s}\right)^i \\ &\quad + s\left(\frac{1-s}{4}\right)^i \sum_{k\geq 0}\binom{2k+i}{k}\left(\frac{1-s}{4}\right)^{2k} \\ &= p_a\sqrt{s}\left(\frac{1+s-2\sqrt{s}}{1-s}\right)^i - \frac{2p_a s}{\sqrt{3+2s-s^2}}\left(\frac{2-\sqrt{3+2s-s^2}}{1-s}\right)^i \\ &\quad + \frac{2s}{\sqrt{3+2s-s^2}}\left(\frac{2-\sqrt{3+2s-s^2}}{1-s}\right)^i \\ &= p_a\sqrt{s}\left(\frac{1+s-2\sqrt{s}}{1-s}\right)^i + \frac{2(1-p_a)s}{\sqrt{3+2s-s^2}}\left(\frac{2-\sqrt{3+2s-s^2}}{1-s}\right)^i.\end{aligned}$$

These examples illustrate the relative difficulty of developing similarity functions using the merge method. Conceptually, the merge method seems a better way of combining attributes than the additive method, because instances are not restricted to completely transforming along each attribute before beginning transformation on another. However, the choice of method for modelling multiple attributes is also dependent on the domain. As is shown in Chapter 5, there are domains for which the additive method is a better choice.

In addition, unless the instances are simple, the development of a similarity function using the merge method may be difficult.

4.3.10 Missing Values

An issue to be dealt with in many datasets is instances where one or more attribute values are missing. As discussed in Chapter 3, approaches in the literature vary widely on how to deal with this problem. In some cases, the distance to the missing attribute is taken to be the maximum possible, in some it is the minimum possible, and in others the entire instance is ignored (Aha, 1990; Dixon, 1979).

One intuitive way to deal with this is to assume that missing values can be treated as if they were drawn at random from among the instances in the database. This fits within the probability-based similarity method, by setting the probability of transforming to a missing value to be the mean probability of transforming to each (specified) attribute value in the database. That is,

$$\mathrm{P}^*(?\,|\,a) = \sum_b \frac{\mathrm{P}^*(b\,|\,a)}{N},$$

where '?' represents the unknown value, and the sum is over all N specified instances in the database. The effective distance to a missing value is (roughly) the expected distance to a random instance of that attribute.

For instance-based classification, missing values in a test instance can be ignored since the test instance missing value will produce a constant value across all training instances—predictions can be made on just the remaining attributes. With this method the probability for missing values is therefore only needed for missing values in training instances. This method (and others) for treating missing values is evaluated in Chapter 5.

4.4 Conclusions

This chapter presented the idea of interpreting similarity between instances as the probability of transforming between them; transformations are modelled as sequences of smaller basic transformations that can be customised for different instance representations. In order to calculate the probability of a transformation sequence, probabilities are assigned to the basic transformations. The probabilities should be chosen according to what constitutes an important difference in the comparison domain. Adjusting these probabilities is a natural way of setting the relevance of attributes—if a transformation on some dimension is relatively unimportant it should be given a high probability of occurring.

The design framework meets the basic requirements set out in Chapter 1. Different instance types are treated consistently—the only difference between instance types is the set of basic transformations. With appropriate basic transformations we can measure similarity between differing numbers of instances. Missing information may also be treated intuitively within the framework. Domain information is captured within the similarity functions, both in the set of basic transformations and the probabilities assigned to them.

Several similarity functions for simple domains have been developed and discussed. The example similarity functions demonstrate that the difficulties identified in Chapter 1 can be handled within the framework. In the next chapter these examples are combined into a practical implementation that can be applied to real-world problems.

Chapter 5

K* Application: An Instance-based Learner

This chapter deals with the construction of the K* instance-based learner, a machine learning scheme that makes use of K* theory in its distance measure. The K* learner is used to test our claims of coherent attribute treatment and missing value handling, and the ability to capture domain information within the function. We evaluate the K* learner on artificial and real-world datasets, and compare its performance against other machine learning schemes.

5.1 Implementation

The implementation of an instance-based learner that determines similarity based on the K* theory requires solving two specific problems. The first is how to employ the similarity function in making predictions about the test instance. The second is how to select values for the free parameters in the similarity functions of the previous chapter. These can be handled within the K* framework, and are discussed in detail below.

5.1.1 Category Prediction

The usual task for an instance-based learner is to predict the category of test instances. A nearest neighbour learner returns the category of the nearest training instance as its prediction—an analogous treatment in this implementation would be to return the category of the training instance most likely to be transformed to. However, following the K* philosophy of considering all possible paths, we calculate the probability of an instance *a* being in category *c* by summing the probabilities from *a* to each training instance *t* that is a member of *c*:

$$\mathrm{P}^*(c \mid a) = \sum_{t \in c} \mathrm{P}^*(t \mid a) . \qquad (5.2)$$

The probabilities are calculated for each category; the relative probabilities obtained give an estimate of the category distribution in the area of instance space represented by *a*. Most other techniques return a single category as the classification result, so for ease of comparison we choose the category with the highest probability as the classification of the new instance. Alternatives to this include choosing a class at random using the relative probabilities, or returning a normalised probability distribution as the result.

5.1.2 Simple Numeric Prediction

Instance-based learners are frequently used to predict numeric values. The K* learner's prediction of numeric attributes is simply based on the expected value for the attribute after the test instance has transformed to one of the training instances. The predicted value of attribute x for instance a is

$$a_x = \sum_t t_x \, \mathrm{P}^*(t \mid a).$$

Additionally, the predicted variance of the value a_x is

$$Va_x = \sum_t t_x^{\,2} \, \mathrm{P}^*(t \mid a) - \left(\sum_t t_x \, \mathrm{P}^*(t \mid a) \right)^2 . \tag{5.3}$$

This method assumes the predicted attribute is a standard numeric attribute—that is, the attribute is not (for example) modulo in nature. For example, if the predicted attribute is modulo in the range zero to ten, with the bulk of similar training instances having values one and ten, the method above would return an expected value of around five rather than zero. However, custom methods for predicting modulo and other types of attributes could be derived. For example, one method to obtain a predicted distribution over a modulo attribute is to calculate the sum of transformation probabilities to each training instance (including transformation of the predicted attribute).

5.1.3 Choosing Values for the Free Parameters

For each attribute of a test instance, values must be chosen for the free parameters of the similarity function. If the similarity function combines attributes with the merge method, the free parameters will depend on the particular function. If the similarity function uses the additive method of attribute combination, the free parameters are x_0 (for each numeric attribute) and s (for each symbolic attribute). In this section we consider only the additive method of attribute combination. One approach to setting the free parameter values is to determine global values (that is, use the same values for every test instance) using a method such as cross-validation on the training data. However, the best setting for these parameters is likely to vary depending on position in the instance space, so we calculate the parameters for every test instance.

The behaviour of the distance measure as the parameters x_0 and s change is interesting. Consider the probability function for symbolic attributes from Section 4.3.7 as s changes. With a value of s close to 1, training instances with a symbol different to the current one have a low transformation probability, while instances with the same symbol have a high transformation probability. Thus, the distance function exhibits nearest neighbour behaviour. As s approaches 0, the transformation probability directly reflects the probability distribution of the symbols, favouring symbols that occur more frequently. This behaviour is similar to the default rule for many learning schemes, which is simply to predict whichever classification is most likely (regardless of the new instance's attribute values). As s changes, the behaviour of the function varies smoothly between these two extremes. The similarity function for real valued attributes exhibits the same properties. When x_0 is small, the probability function decreases quickly with increasing distance, functioning like a nearest neighbour measure. Conversely, if x_0 is large almost all the instances will have the same transformation probability and will be weighted equally.

5.1.3.1 "Sphere of Influence" Approach

In both these cases the number of instances that are effectively included within the probability distribution vary from 1, when the distribution is nearest neighbour, to the total number of instances *N*, when all instances are weighted equally. If more than one neighbour is nearest, the minimum will be greater than 1. The effective number of instances *E* can be computed for *any* function P* using the following expression:

$$E = \frac{\left(\sum_t \mathrm{P}^*(t \mid a)\right)^2}{\sum_t \mathrm{P}^*(t \mid a)^2}. \tag{5.1}$$

E ranges from n_0, the number of training instances at the smallest distance from instance *a*, to the total number of training instances *N*.

The K* algorithm chooses a value for x_0 (or s) by selecting a number between n_0 and *N* and inverting the expression above. Selecting n_0 gives a nearest neighbour algorithm; choosing *N* gives equally weighted instances. For convenience the number is specified using the "blend parameter" *b*, which varies from $b = 0\%$ (for n_0) and $b = 100\%$ (for *N*), with intermediate values interpolated linearly.

We think of the blend parameter as a "sphere of influence", specifying how many of a's neighbours should be considered important (although there is not a harsh cut off at the edge of the sphere—more a smooth decrease in each instance's relative contribution).

An iterative root finder is used to compute x_0 (or s), with the results cached, so that whenever an instance value reappears the pre-calculated parameters can be used. The x_0 (and s) parameters are calculated for each dimension

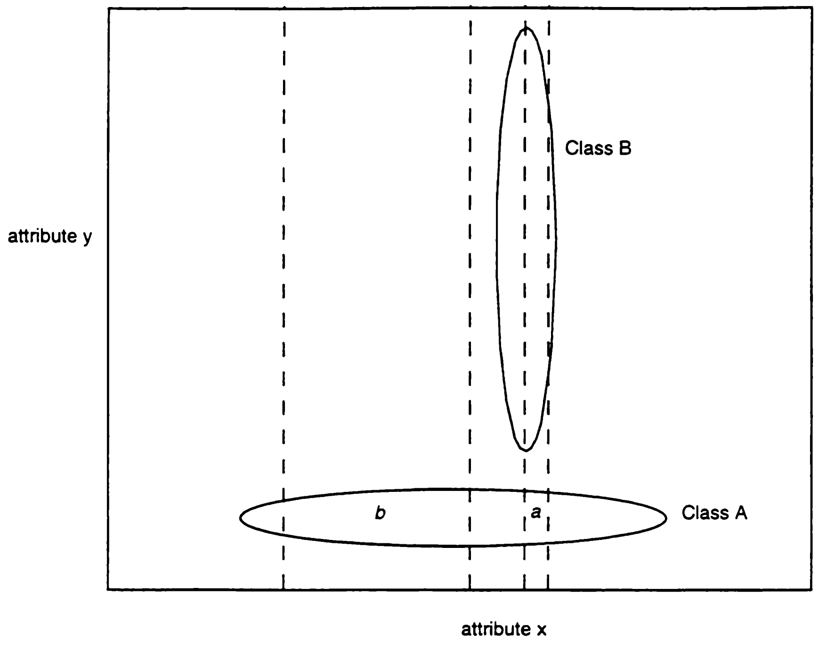

Figure 5.1: Blend sensitivity to other attributes

independently (that is, substituting the probability function for a single attribute into Equation 5.1), but using the same blend parameter, which gives equal weight to each attribute. The size of the final sphere of influence is computed from the combined attribute distance measure. This is usually much smaller than the size specified at the single attribute level (on the order of b^d, where d is the number of attributes).

The drawback to setting the parameters for each attribute independently is that the settings can be affected by instances that are dissimilar on other attributes. Assume that in the situation shown in Figure 5.1 each class contains equal numbers of training instances, and that they are uniformly distributed within the regions marked. The dashed lines surrounding instances *a* and *b* indicate the surrounding x-attribute interval that contains 25% of the training instances. The x-attribute is effectively assigned low importance when classifying instance *b* and high importance when classifying instance *a*, even though in both cases the x-attribute is less relevant than the y-attribute in determining the class. However, to employ the probability function for all attributes in Equation 5.1 would be prohibitively slow due to the additional computation required, and because it would require the free parameters for the other attributes to be initialised by some method.

5.1.3.2 Automatic Determination

The sphere of influence method for setting the free parameters effectively gives each attribute equal relevance. However, performance on many domains can be significantly improved if irrelevant attributes are either ignored entirely, or given lower weight in the similarity function. In general, irrelevant attributes should be assigned a large sphere of influence, while relevant attributes should be assigned a small sphere of influence. Assuming a categorical class attribute, the entropy of the predicted class distribution for each attribute can be calculated with respect to the free parameter (x_0 or s depending on the attribute type) from the training attribute values.

$$Ent(a) = -\sum_{c} \mathrm{P}^*(c \mid a)\log(\mathrm{P}^*(c \mid a)),$$

where $\mathrm{P}^*(c \mid a)$ is the predicted probability of class c for test instance a, calculated using only the current attribute of interest. The parameter value could then be chosen to minimise the entropy of the prediction. However, this will usually reduce the sphere of influence to include only the single nearest neighbour (even if the attribute is irrelevant)[3]. This is clearly not the best plan, even for relevant attributes. For example, if many surrounding neighbours

[3] This problem is analogous to that of overfitting in decision tree construction. It is (barring identical instances with different classes) always possible to construct a decision tree that correctly classifies all training instances, by ensuring that each leaf node corresponds to one training instance. Such a tree typically performs poorly on new instances.

belong to a different class than the nearest neighbour, it makes sense to give their predictions higher weight, in case the single nearest neighbour contains erroneous data. In theory the same entropy curve can be computed if classes were randomly assigned to attribute values. The proposal is that the parameters should be set to maximise the entropy difference between the two curves. That is, set the parameters so that the entropy is maximally different to the expected entropy for an irrelevant attribute.

In terms of implementation, the expected entropy for random class assignment is approximated by averaging the entropy calculated from a number of random permutations of the class attribute. We use 5 permutations—a higher number yields a more accurate approximation but increases classification time. The entropy difference optimisation is computed for each attribute independently (for the same performance reasons as stated in Section 5.1.3.1)—again the procedure may be sensitive to instances that are dissimilar on attributes other than the one being optimised, as well as losing information about the relative importance of attributes. This problem is particularly noticeable with symbolic attributes where, if the attribute is even marginally correlated with the class, the optimisation attempts to weight the attribute as high as possible. This would not be a problem if the attribute is the only predictor attribute. The independent optimisation is unable to determine the relative importance of the attributes. To alleviate this problem, attributes that do not yield an entropy difference higher than a threshold have their parameters set so that the attribute is effectively rendered irrelevant.

5.2 Evaluation

In this section the behaviour of the K* classifier is empirically studied with regard to the issues discussed in Chapter 3. In particular, we examine our claims of the coherent treatment of different attribute types and multiple attributes, making use of instances with missing information, and ease of customisation to the domain. In addition, we evaluate the automatic method for selecting one-step symbolic probabilities described in Section 4.3.8, and the automatic method for selecting stop parameters described in Section 5.1.3.2. Finally, the K* classifier is compared with standard machine learning schemes on a variety of datasets.

Any instance-based learner with an adequate similarity function will eventually reach the optimal error rate (Fix and Hodges 1951), so in most of the following experiments we are primarily interested in examining the learning rate (that is, the change in classifier performance as number of training instances increases). A scheme that learns faster will have an advantage over other schemes when the number of training instances is limited. Alternatively, a fast learning scheme may employ an editing function to minimise storage requirements and classification time, while maintaining the same accuracy as other schemes.

5.2.1 Experimental Methodology

This section describes the features and performance measures common to the following experiments. In many of the experiments, artificial domains are employed. Artificial domains are useful because they can be designed specifically to test a particular hypothesis in isolation. Where artificial datasets are used, the number of test instances is 500. Each data point is the result of 50 trials, with randomly chosen test and training instances for each trial. When two data points are stated to be significantly different, this means that a two-tailed, paired *t*-test, indicates the results to be different at the 95% confidence level. The default settings for the K* classifier are to use additive attribute combination (since the merge method requires a similarity function derived for each particular combination of attribute types) and a manual blend parameter setting of 20%. Except where explicitly stated, these settings are used in all experiments.

Classifier performance is evaluated with respect to error rate and entropy gain. The error rate of a trial is expressed as a percentage of test instances that were not correctly classified. When a test instance produces multiple classifications, it is counted as incorrect (even if the correct class was one of the predictions). Although commonly used in the literature, error rate is a coarse measure of classifier performance, as it is based only on the single prediction offered for a test instance. Since the K* classifier can produce a class distribution as a prediction, we can use this to provide a better measure of how much information the classifier is extracting from the domain.

5.2.1.1 Entropy Gain Measure

The entropy gain measure is effectively a measure of how much less information is required to encode the test instance classes when using the scheme's predictions as opposed to encoding the classes using a naive method. This measure has been applied to machine learning schemes that produce rules (Cleary *et al.*, 1996b). This section describes a comparable measure for instance-based learners (or any machine learning scheme that infers a probability distribution over the classes for each instance).

The number of bits required to encode the category of an instance with class c with respect to a probability distribution P is $-\log_2(\mathrm{P}(c))$. One naive method for obtaining a class probability distribution is simply to count the number of instances of each class that appear in the training data. However, if a class does not appear in the training data, it is assigned a probability of 0, which requires an infinite number of bits to encode. This is called the *zero frequency problem*, discussed further in Witten and Bell (1991). To circumvent this, the counts for each possible class start from 1. If there are N training instances and n possible classes (each of which occurs f_c times in the training data), the probability assigned to class c is:

$$\mathrm{P}_{naive}(c) = \frac{f_c + 1}{N + n}.$$

This probability is independent of the test instances. The associated entropy is

$$\mathrm{entropy}_{naive}(c) = -\log_2(\mathrm{P}_{naive}(c)).$$

The zero frequency problem can also occur with the probability distribution provided by the classifier. To counter this, the scheme's probability distribution is combined with the naive distribution so that the relative probabilities assigned by the scheme are preserved. Two weights α and β control the blending. α and β are initially assigned probabilities of $\frac{1}{n}$. After each prediction, α is incremented by $\mathrm{P}^*(c \mid a)$, and β is incremented by $\mathrm{P}_{naive}(c)$. Thus, if the scheme is consistently providing more accurate predictions than the naive method, $\alpha >> \beta$ and so α will dominate the weighting below. The modified predicted probability for the class of instance a is calculated as

$$\mathrm{P}'(c \mid a) = \frac{\alpha \mathrm{P}^*(c \mid a) + \beta \mathrm{P}_{naive}(c)}{\alpha + \beta},$$

where $\mathrm{P}^*(c \mid a)$ is the probability distribution predicted by K* as given in Equation 5.2. The corresponding entropy is

$$\mathrm{entropy}_{K*}(c \mid a) = -\log_2(\mathrm{P}'(c \mid a)).$$

Where a standard *k*-NN learner is used, the P* probability distribution is replaced by one that assigns probabilities according to the distribution of classes among the *k* nearest neighbours. Thus, a 1-NN learner always assigns a (pre-modification) probability of 1 to its predicted class.

The entropy gain measure is defined as the difference between entropy_{naive} and entropy_{K*}, averaged over all test instances. Although an absolute difference between entropy_{naive} and entropy_{K*} intuitively makes more sense, we use the average, to permit comparison over varying numbers of test instances (which is required for domains with a fixed total number of instances). A negative entropy gain implies the classifier performed no better than predicting the class based solely on the class distribution in the training data. A positive entropy gain implies the classifier has successfully captured domain information when making its predictions. It is possible for a scheme to achieve a high error rate in a domain, and at the same time perform well with regards to the entropy gain. For example, when a classifier provides multiple classifications for an instance, these are typically regarded as incorrect when calculating error rates. Allocating equal probability to multiple classes can improve the entropy gain measure if the scheme has managed to effectively determine which classes are unlikely. Similarly, if a scheme would have produced the actual class as a "second choice", the error rate would increase, but the entropy gain measure would not necessarily decrease, particularly if the scheme has successfully determined which classes are unlikely. For some problems there will be a simple correspondence between these two measures—if one scheme has a lower error rate than another scheme, it will typically also have a higher entropy gain (Cleary *et al.*, 1996b).

5.2.2 Coherent Treatment of Different Attributes

In this section we examine whether the K* learner treats different types of attributes consistently. The method taken is to select a domain and represent it using different types of attributes. Assuming the different representations contain the same information, classification performance on the different datasets should be the same. In this section we only consider domains with one predictor attribute (methods for combining attributes are evaluated in Section 5.2.3).

5.2.2.1 Ultra-violet Domain

Ting (1995) describes a domain where the time of the day is used to predict the level of ultra-violet radiation. If the time is between 11AM and 3:30PM, the level of UV is high, otherwise it is low. The first dataset, UV1, represents the time of the day as two attributes: a symbolic AM/PM indicator; and a real-valued time in the range $[0,12)$. The second dataset, UV2, represents the time of the day as a single real value in the range $[0,24)$. Chapter 3 showed that these representations do not contain the same information—for example, UV2 implicitly contains information about the relationship between 11:59AM and 12:00PM that UV1 does not. Two new datasets are defined for this experiment. UV3, which differs from UV2 by representing the time as an integer hour rather than a real. Information about the minute within the hour present in UV2 is not contained in UV3. The second dataset, UV4, represents the time as a symbolic type, with one symbol for each hour of the day. UV4 does not contain hour-ordering information present in UV3.

The following four methods of classification are examined:

REAL: The UV2 dataset is classified using the similarity function for wrapped space developed in Section 4.3.4. This provides the classification with the information that the time of day is modulo.

INTEGER: The UV3 dataset is classified using the similarity function for wrapped space developed in Section 4.3.4 (effectively the same as that derived in Section 4.3.2.1).

INDEPENDENT: The UV4 dataset is classified using the similarity function for independent symbols developed in Section 4.3.7.

NON-INDEPENDENT: The UV4 dataset is classified using the similarity function for non-independent symbols developed in Section 4.3.8. The one-step transformation probabilities for symbols are set so that each hour has an equal probability of transforming to each of its neighbouring hours. For example, the "13th hour" symbol may transform to the "12th hour" symbol or the "14th hour" symbol each with probability of 0.5. The assigned probabilities also reflect the modulo nature of the symbols—the "1st hour" symbol may transform to either the "2nd hour" or the "24th hour" symbol and similarly for transformations from the "24th hour" symbol.

Figure 5.2 shows the learning rate for these four classifiers. REAL performs better than the other methods particularly as the number of training instances increases. The differences are consistently significant above 55 training instances and are often significant below. REAL is able to locate the exact location of the class boundary at 3:30PM as the number of training instances increases. The other methods are unable to distinguish between instances half an hour either side of the boundary so the lower bound on the error rate is 2.1%, regardless of which class is chosen; INTEGER and NON-INDEPENDENT

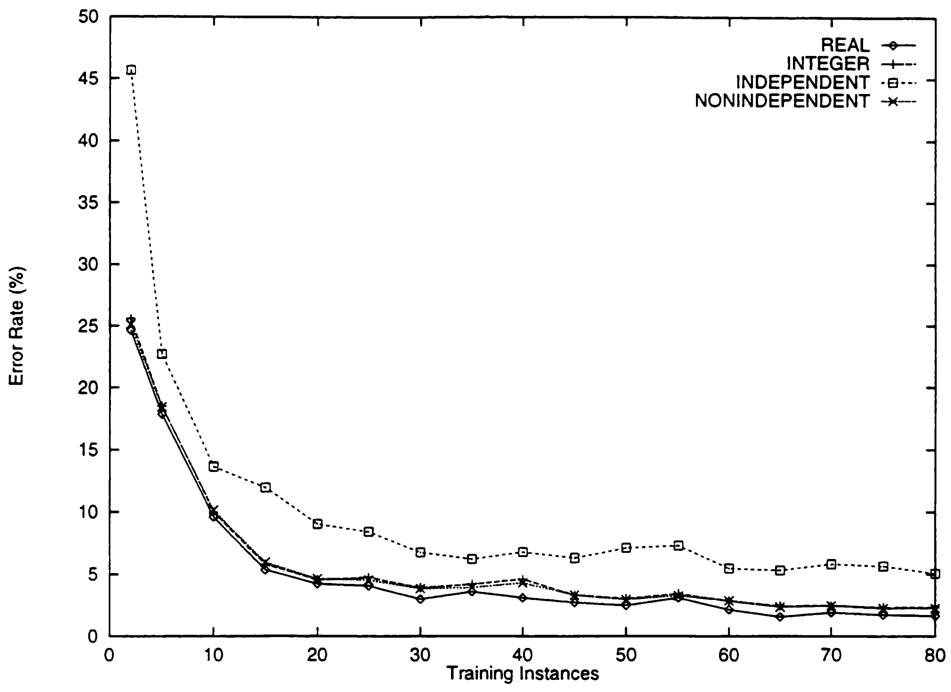

Figure 5.2: Learning rate on UV domain

approach this error rate, while real continues to slowly improve its performance.

INDEPENDENT performs significantly poorer than the other methods for all levels of training instances except for the 5 training instances sample, primarily because INDEPENDENT has no indication of inter-hour relationships. Test instances for which there are no training instances with the same hour are often misclassified; their predictions are based on whichever hour has a majority (which is likely to have class UVLOW). The performance of INDEPENDENT should gradually approach that of INTEGER and NON-INDEPENDENT as the probability of having at least one training instance in each hour period increases.

Figure 5.3 shows the corresponding entropy gain graph. INDEPENDENT performs significantly poorer than the other methods. REAL performs

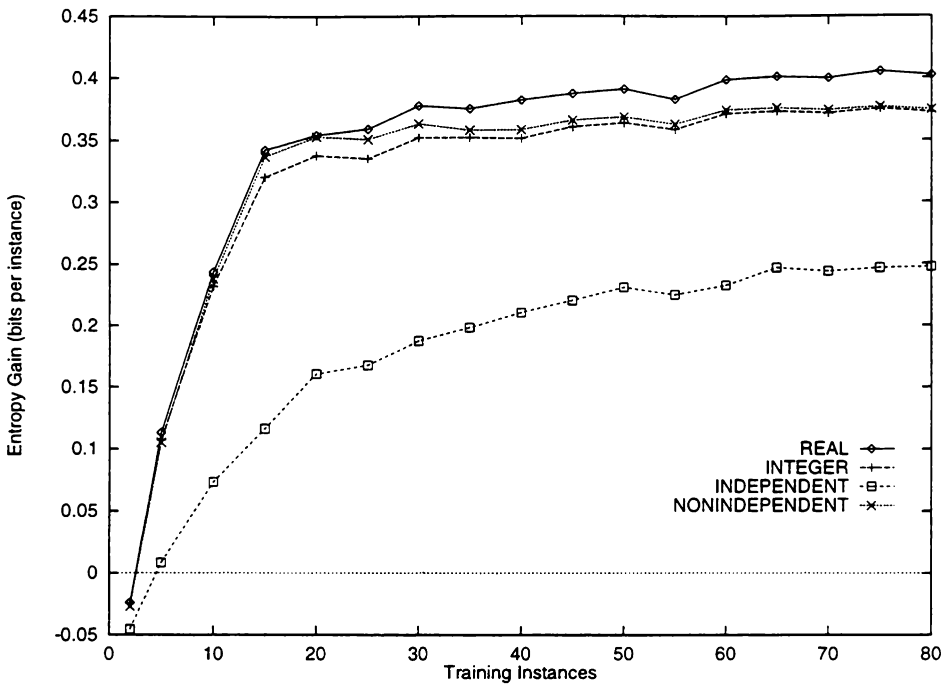

Figure 5.3: Entropy gain on UV domain

significantly better than INTEGER above 2 training instances, and significantly better than NON-INDEPENDENT above 20 training instances. Although the error rates for INTEGER and NON-INDEPENDENT are not significantly different, the corresponding entropy gain results are significantly different above ten training instances (but converge as training levels increase).

The performance differences observed between REAL and the other methods, and between INDEPENDENT and the other methods are expected, and are due to the domain representations implicitly containing different types of information. INTEGER and NON-INDEPENDENT have few significant differences in performance; small differences are likely due to implementation details. Their similar performance confirms that different attribute types are treated coherently. In addition, the large performance difference between REAL and INDEPENDENT highlight the importance of making use of domain information wherever possible.

5.2.3 Multiple Attributes

The objective of this section is to examine whether multiple attributes are combined coherently. Our approach is to apply transformations to the domain attributes and see whether this has an effect on performance. For example, in some domains it may make sense to perform a "rotation" operation on the attributes with no change in performance.

5.2.3.1 Rotating Linear Decision Boundaries

This simple two-class problem contains linear concept boundaries at an angle that can be varied. Figure 5.4 shows 100 example instances where the concept boundaries are at a 0° angle relative to the x-axis. For each trial, the test and training datasets are rotated about the origin by various angles.

Two methods from Section 4.3.9 for combining attributes are examined: the simple additive method (which corresponds to transforming each attribute sequentially); and the merge method (which allows transformations on each attribute to be intermingled). The merge results do not use the exact function derived in Section 4.3.9 but an approximation that shares the essential characteristic of being based on the Euclidean distance between attribute values.

The results for ten training instances are shown in Figure 5.5. The first noticeable feature is that the merge method error rate is constant, regardless of the concept boundary angle. The additive method error rate varies significantly from the lowest point at 0° and 90° rotation to the highest point around 40° rotation. From 25° to 60° rotation, the additive method performs

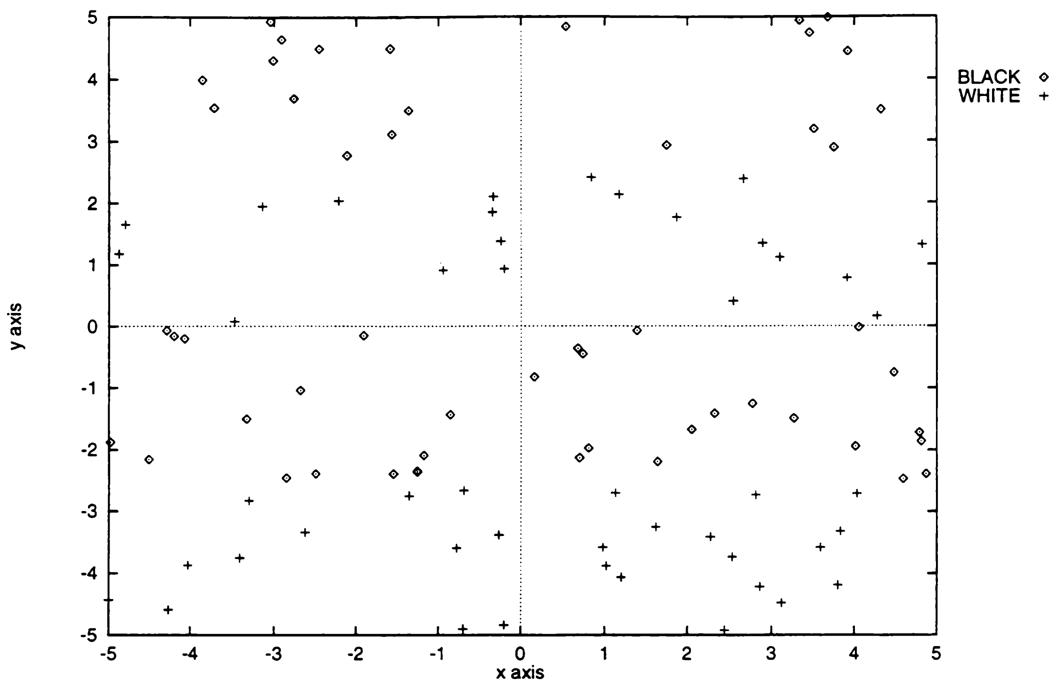

Figure 5.4: 100 example instances for the linear boundary domain, 0° incline

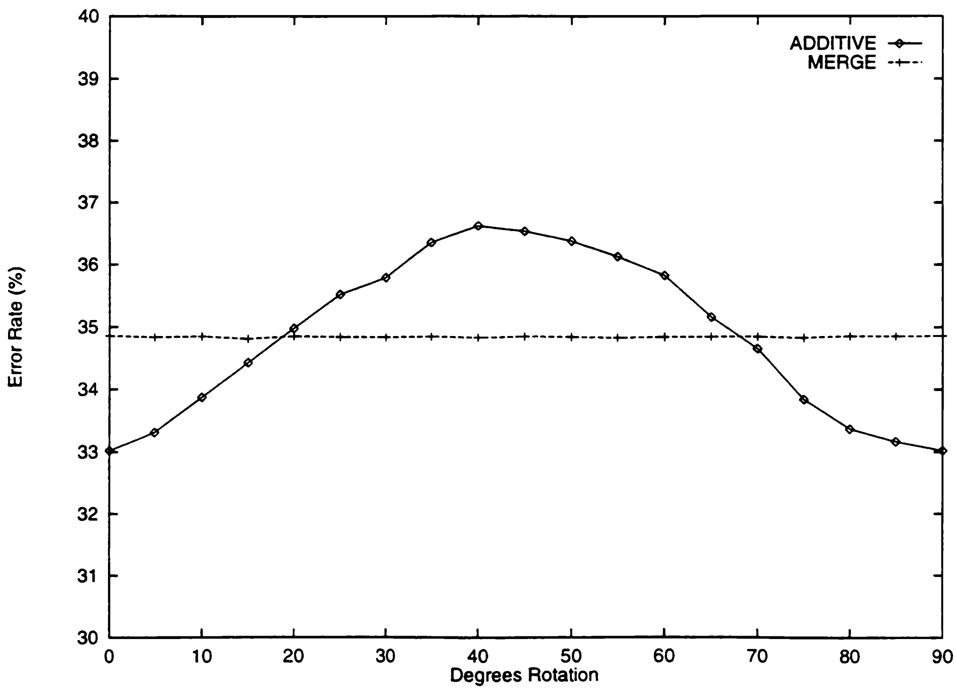

Figure 5.5: Error rates on rotated datasets with ten training instances

significantly poorer than the merge method. From 0° to 10°, and from 75° to 90° rotation, the additive method performs significantly better than the merge method. The conclusion is that the additive method is sensitive to dataset rotation.

To understand these results, consider how the similarity function changes for areas around each instance. Figure 5.6 shows a contour of constant similarity around a training instance using the two methods—the contour is circular for merge combination, and diamond shaped for additive combination. Now consider the decision boundaries formed by pairs of training instances. Figure 5.7 shows the decision boundaries for pairs of training instances; the additive method on the left, the merge method on the right. The boundary between the training instances represents the points at which the similarity to both instances is equal. When the two instances are aligned horizontally, both methods produce the same decision boundary. When the alignment of the two instances is 15°, the additive method produces a predominantly vertical boundary (with a short section at $-45°$).

An interesting effect occurs when the instances are aligned at 45°. The black areas indicate regions where the decision boundary has widened (that is, the distance to both instances is equal), and classification is impossible. Thus, to exactly represent a concept boundary at 45° requires multiple pairs of training instances. (Due to rotational symmetry of the additive method, the figures will still be correct when rotated by 90°.) These effects produce a bias in the additive method toward domains with axis-parallel concept boundaries, and against domains with concept boundaries approaching 45°, and hence explain the curve seen in Figure 5.5.

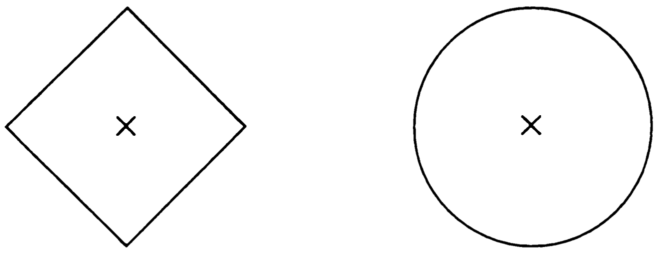
Figure 5.6: Contours of equal similarity for additive and merge combination

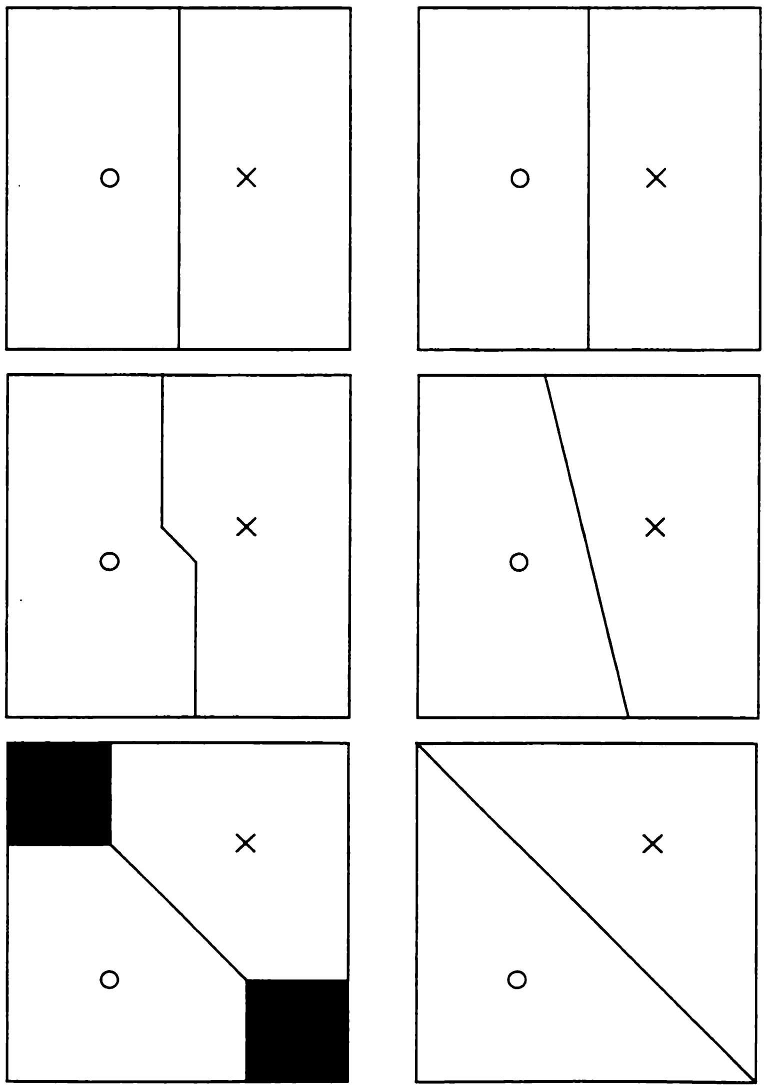
Figure 5.7: Decision boundaries for additive and merge combination

The entropy gain results for ten training instances (Figure 5.8) are much as we would expect given the error rate results. ADDITIVE performs significantly better than MERGE from 0° to 30° rotation and from 60° to 90° rotation. ADDITIVE performs significantly worse than MERGE from 40° to 45° rotation.

A more interesting result is obtained when we plot the entropy gain over a range of training instances, as shown in Figure 5.9. ADDITIVE-0 corresponds to the additive method classifying the unrotated dataset; ADDITIVE-45 is the additive method classifying the dataset rotated 45°; and MERGE shows the results of the merge method (which showed no significant differences with dataset rotation) classifying the unrotated dataset. ADDITIVE-0 has a significantly higher entropy gain than ADDITIVE-45 for all the levels shown (although the results converge with increasing training instances), reflecting the bias of the additive method towards axis-parallel concept boundaries. Initially, the entropy gain for MERGE is almost indistinguishable from the entropy gain of ADDITIVE-45. As the number of training instances increases MERGE achieves significantly better entropy gain than ADDITIVE-45, approaching that of ADDITIVE-0.

These results suggest that the additive method's bias toward axis-parallel concept boundaries is mainly beneficial when the number of training instances is low. Under these conditions, the distance from the actual to the estimated concept boundary for the ADDITIVE is often bounded, while for MERGE the distance is not bounded. As training instances are added, the estimated boundary position is refined to the point where the bounds are not significantly different.

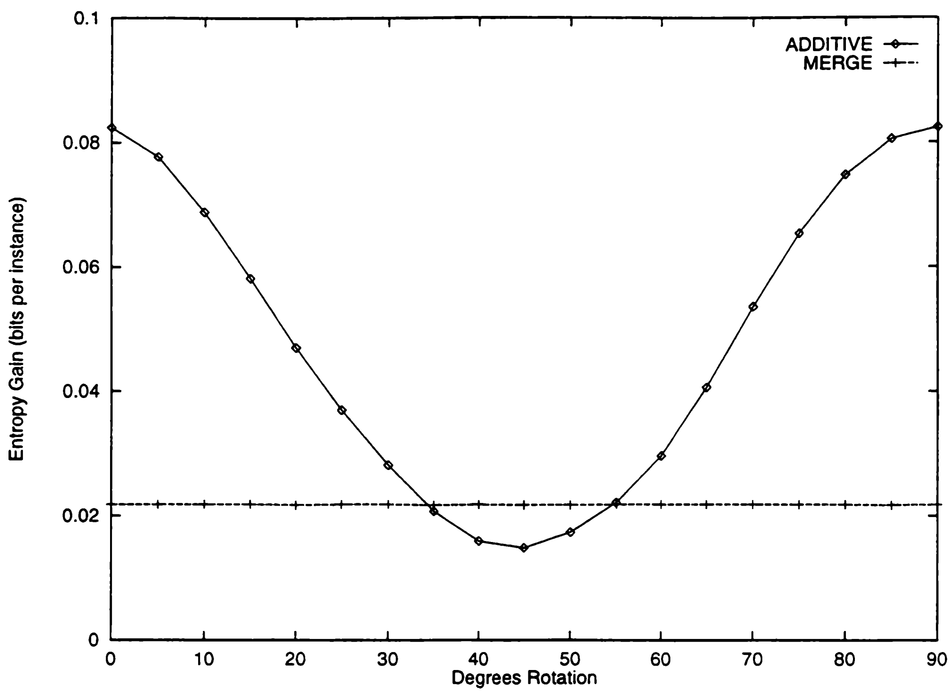

Figure 5.8: Entropy gain with ten training instances

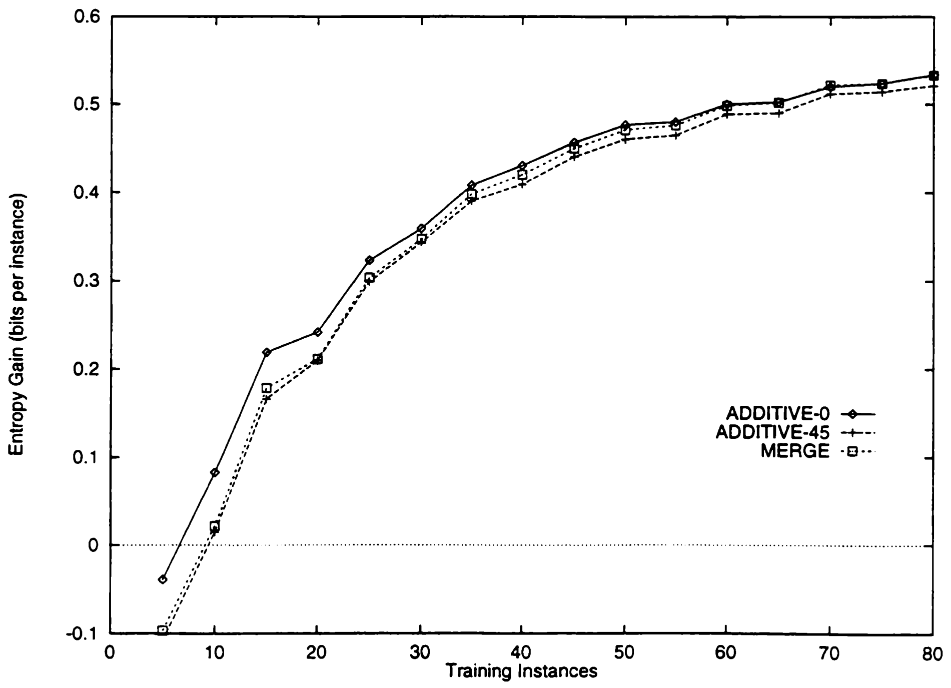

Figure 5.9: Entropy gain with increasing training instances

So, which of the two combination methods is better? The results above show that it depends on the domain. If domain concept can be effectively partitioned into independent sub-concepts, the attributes relevant to different sub-concepts should probably be combined with the additive method. For example, say the task is to predict a person's age group at death, with attributes "number of cigarettes smoked per day" and "CC rating of the person's car." The relevant sub-concepts are "age of people who die driving powerful cars" (who tend to be young, inexperienced drivers) and "people who die as a result of smoking" (who tend to be older). The number of cigarettes smoked per day has no effect on whether or not a person is likely to die while driving a powerful car, and similarly the car CC rating is not a good indicator to whether a person might die as a result of smoking. Any concept boundaries will be axis-parallel, and so the additive method for combining attributes would prove a better choice. There may be some domains where it is reasonable to use a hybrid method, within groups of attributes the merge method may be used, and the results between groups combined with the additive method.

The domain transformation used in this example, rotation, is not applicable to all domains. For example, rotation cannot sensibly be applied to two symbolic attributes. The next section examines a domain where multiple symbolic attributes are combined.

5.2.3.2 Ultra-violet Domain

In this experiment, the transformation applied to the UV domain is to split one symbolic attribute into two separate attributes. The original dataset used is the UV4 dataset described in Section 5.2.2.1; that is, the time of the day is represented as a single symbolic attribute with one symbol for each hour of the day. The modified dataset, called UV5, contains the time of the day represented as one symbolic attribute containing an AM/PM indicator, and one symbolic attribute containing the hour, numbered from 0 to 11. In the previous section, dataset rotation did not alter the domain information implicit in the representation—this is not true for this example. The UV5 representation gives an implied similarity between (for example) 6AM and 6PM, and an implied similarity between 6AM and 10AM. There is also a large implied dissimilarity between 11AM and 0PM. Basic transformation probabilities must be chosen carefully, to ensure the results are not biased by these differences. The classification methods used are:

UV4-INDEPENDENT: The UV4 dataset is classified using the similarity measure for independent symbols developed in Section 4.3.7.

UV5-INDEPENDENT: The UV5 dataset is classified using the similarity measure for independent symbols developed in Section 4.3.7. Attributes are combined using the additive method

UV4-NON-INDEPENDENT: The UV4 dataset is classified using the similarity measure for non-independent symbols developed in Section 4.3.8. One-step probabilities are assigned as follows: each hour has equal probability of transforming to either the preceding hour, the following hour, or 12 hours distant (corresponding to an AM/PM switch).

UV5-NON-INDEPENDENT: The UV5 dataset is classified using the similarity measure for non-independent symbols developed in Section 4.3.8. One-step probabilities are assigned as follows: the AM or PM symbol has transformation probability 0 to itself, and 1 to the other symbol; and the hour symbol has equal probability of transforming to the preceding hour or the following hour (in this case there are only 12 hour symbols). The attributes are combined using the additive method

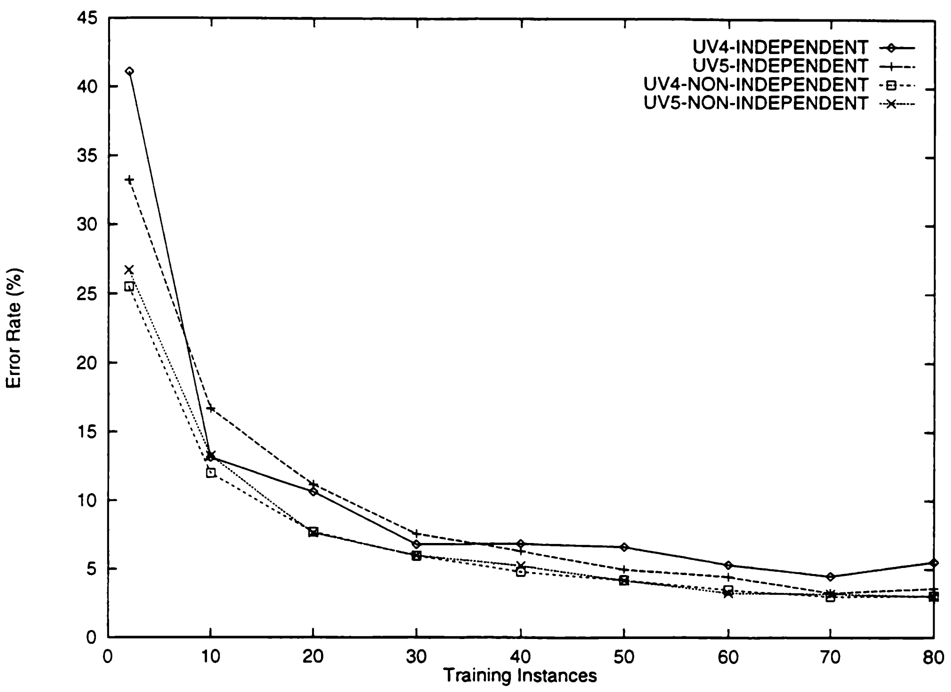

Figure 5.10: Learning rate on UV domain

Figure 5.10 shows the learning rates for the above methods. UV4-INDEPENDENT and UV5-INDEPENDENT are significantly different for 2 training instances, and above 40 training instances. UV4-NON-INDEPENDENT and UV5-NON-INDEPENDENT are not significantly different. The important result is that, although the representations of the domain are different and contain different implicit similarity information, once these differences are adjusted for, there is no significant performance difference between the single attribute representation and two attribute representation. The contributions of each attribute in the two-attribute representation are combined coherently.

5.2.4 Missing Values

In this section the K* classifier is evaluated on datasets containing missing values. The objective is to show that our method for utilising instances with missing values performs better than discarding such instances, and that this method performs comparably to other methods.

The following methods for dealing with instances containing missing values are examined:

DELETE: Ignore any training instance containing missing values; this is similar to DELETE (Dixon, 1979). This method is implemented in the K* classifier by setting the probability of transforming to a missing value to 0; instances containing missing values contribute nothing to the sum in Equation 5.2.

MAXDIFF: Assume the probability of transforming to a missing value is the same as transforming to the furthest value for that attribute. This method is similar to MAXDIFF (Aha, 1990).

NORMAL: Assume the probability of transforming to a missing value is the same as the average transformation probability over the other attributes. This is similar to NORMAL (Dixon, 1979), and IGNORE (Aha, 1990).

AVERAGE: Assume the probability of transforming to a missing value is the same as the average transformation probability over the other instance values, as described in Section 4.3.10. This method is most similar to AVERAGE (Dixon, 1979) and MODEMEAN (Aha, 1990).

These methods are all run-time methods (that is, the similarity to a missing value can be easily computed during classification), as opposed to the

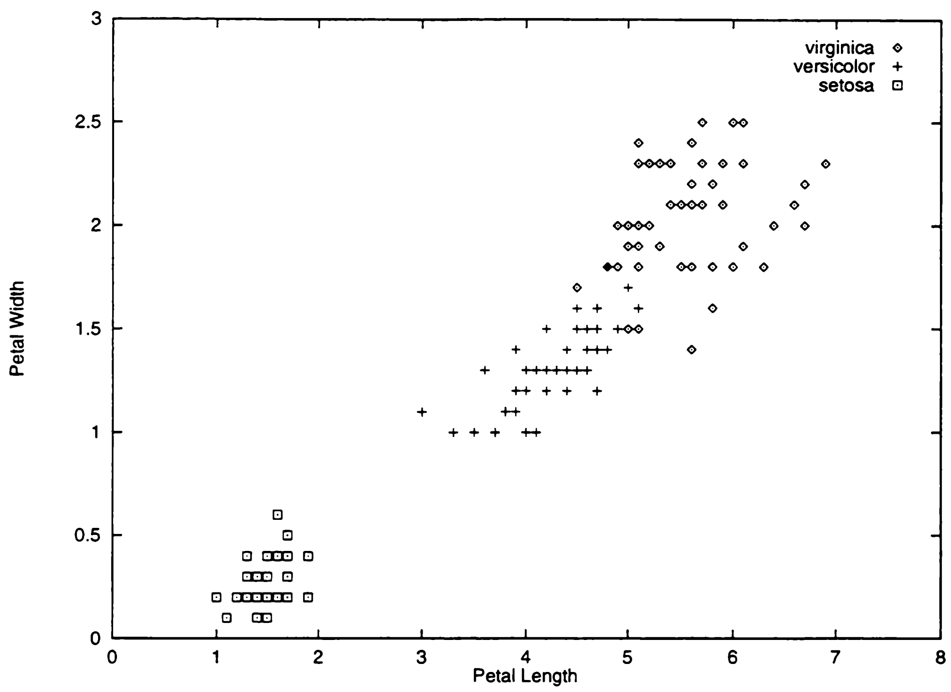

Figure 5.11: Fishers original iris dataset, shown for two attributes

preprocessing methods for filling in missing values described by Dixon (1979).

5.2.4.1 Pseudo-iris Domain

Fisher's well known iris dataset contains measurements taken from 50 examples of each of three species of iris: setosa, versicolor, and virginica. There are four predictor attributes: sepal width, sepal length, petal width, and petal length. The classes can be almost completely differentiated by the petal attributes, as seen in Figure 5.11.

Before attempting classification, consider the behaviour of the different methods for dealing with missing values on this dataset. An instance with a missing value may be viewed as a line (rather than a point), along which we

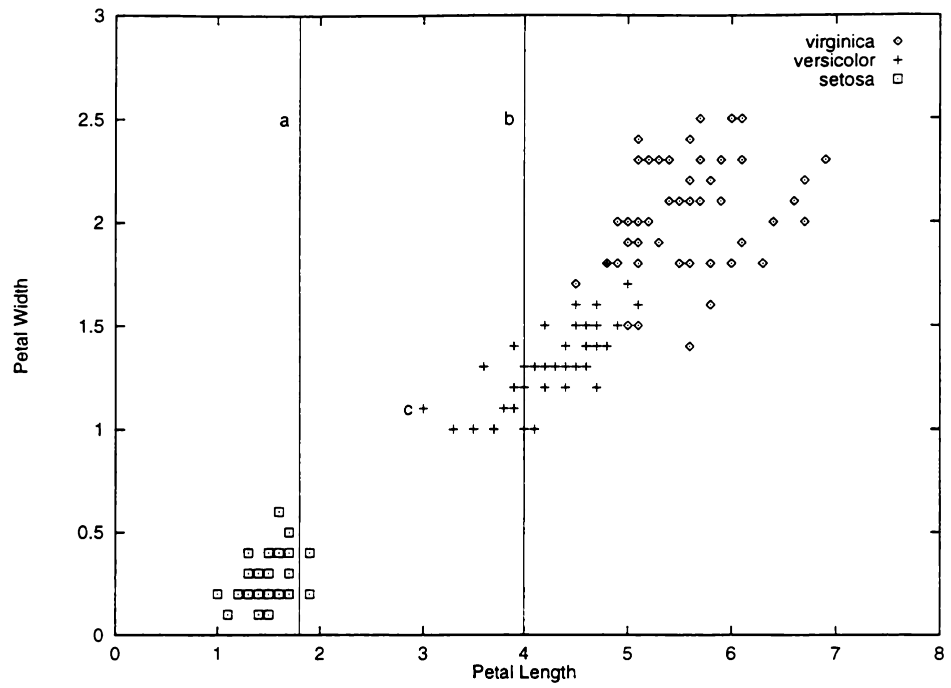

Figure 5.12: Two instances with missing petal width values

are unsure of its position. Figure 5.12 shows the addition of two instances with missing petal width values. The clusters in the data suggest that instance *a* has a petal width value between 0 and 0.5, and that instance *b* has a petal width value between 1 and 1.5. The ideal method for dealing with missing values would return a distance between instances that reflects this expectation.

The first method, DELETE, sets the similarity to instances containing missing values to be 0. MAXDIFF behaves as though the instances have petal width values of either 2.5 or 0.1, whichever is the furthest from the other instance being compared—this method effectively "pushes away" instances with missing values to be used as a last resort.

The behaviour of NORMAL is more interesting. By assuming the similarity to a missing value is the same as the average similarity along the other attributes, the overall similarity reflects our earlier intuition about what the missing

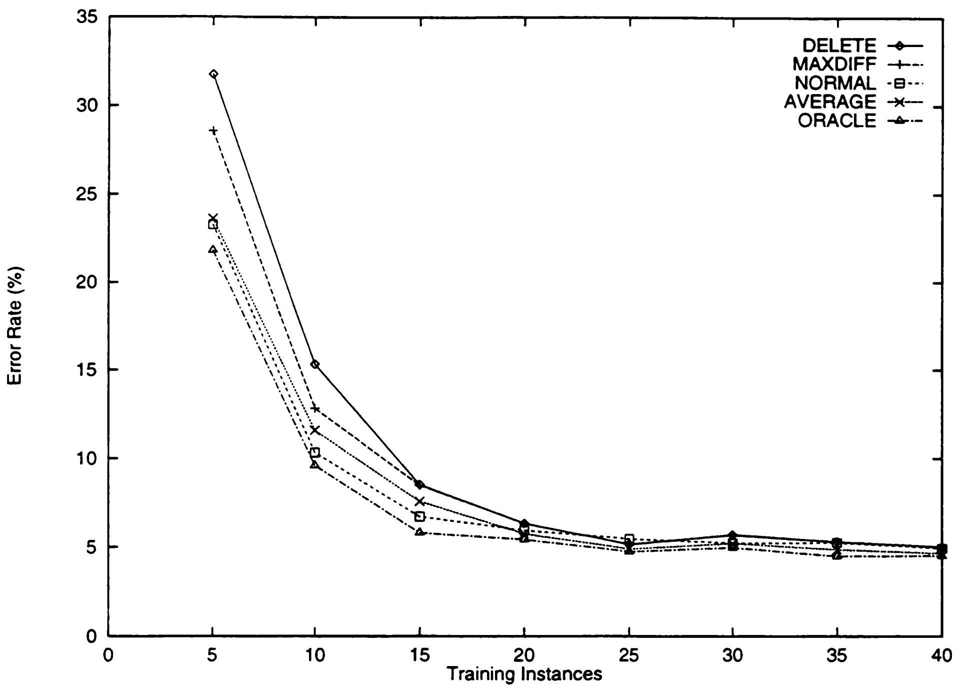

Figure 5.13: Pseudo-iris dataset with 20% missing data

values should be. The similarity from instance *c* to instance *a* will be about the same as if *a* had a petal width of 0.5. The similarity from instance *c* to instance *b* will be about the same as if *b* had a petal width of 1.5. NORMAL is therefore expected to perform well on this dataset.

The behaviour of AVERAGE is only dependent on the attribute containing the missing value. The average petal width value is about 0.7 different from instance *c*'s petal width. In this case the similarity from instance *c* to instance *b* will be about the same as if *b* had a petal width of 1.8. The similarity from instance *c* to instance *a* will be about the same as if *a* had a petal width of 0.4. AVERAGE is not expected to perform as well as NORMAL on this dataset.

To test these theories a pseudo-iris dataset generator was constructed that produces datasets similar to the original iris dataset (only the petal length and petal width attributes are generated). The benefit of using artificial data is that we can create more than the 150 instances in the original dataset.

Figure 5.13 shows the learning rate when 20% of the predictor values are missing, over a variety of training set sizes. The line labelled ORACLE shows the error rate when the training sets do not contain any missing values, and is included for comparison purposes. The baseline accuracy is 33%. Initially there is a wide difference in classification error rate between the different methods, but by the time the number of training instances reaches 40, the error rates are within 1% of each other. The difference between AVERAGE and NORMAL is not significant. MAXDIFF and DELETE perform significantly worse than the other methods when the number of training instances is lower than 20, and below 15 training instances DELETE performs worse than MAXDIFF. These results are consistent with our predictions above.

Figure 5.14 shows the error rate for increasing levels of missing values when the number of training instances is 60. DELETE performs significantly worse than MAXDIFF above 60% missing values, because DELETE often discards so many training instances that most test instances are misclassified or unclassified (which occurs when all training instances are discarded). MAXDIFF performs significantly worse than NORMAL, AVERAGE and ORACLE when there are more than 45% missing values. AVERAGE tends to perform better than NORMAL, although the differences are not significant.

Figure 5.15 shows the error rate for 20 training instances as the level of missing values increases. The differentiation between methods begins at lower levels of missing values than in Figure 5.14. The quantity of "good" training information is about the same, since the error rates in these regions are similar

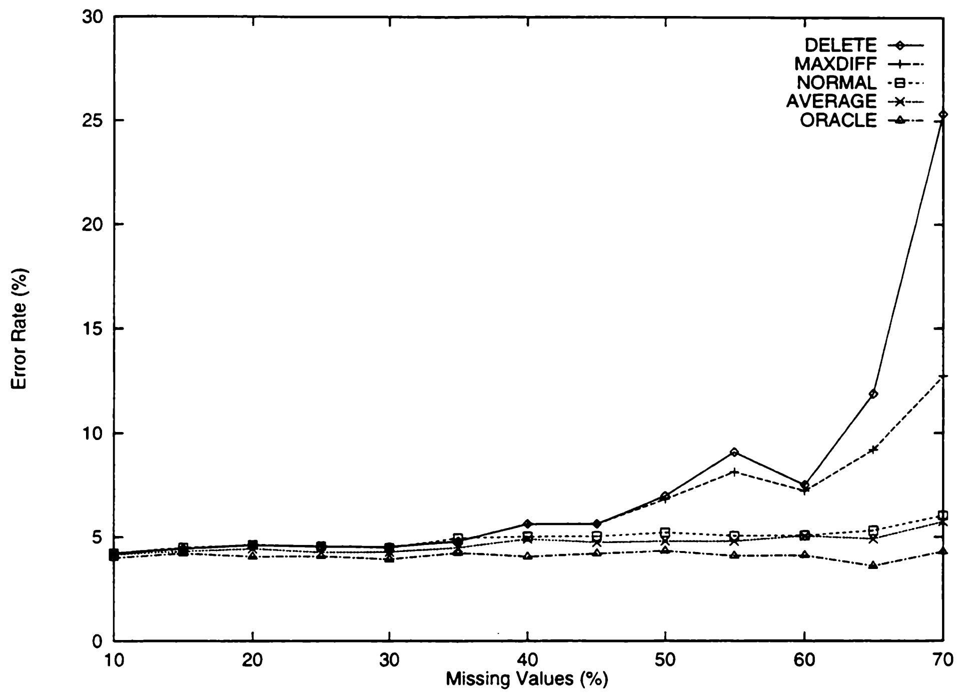

Figure 5.14: Pseudo-iris dataset—60 training instances

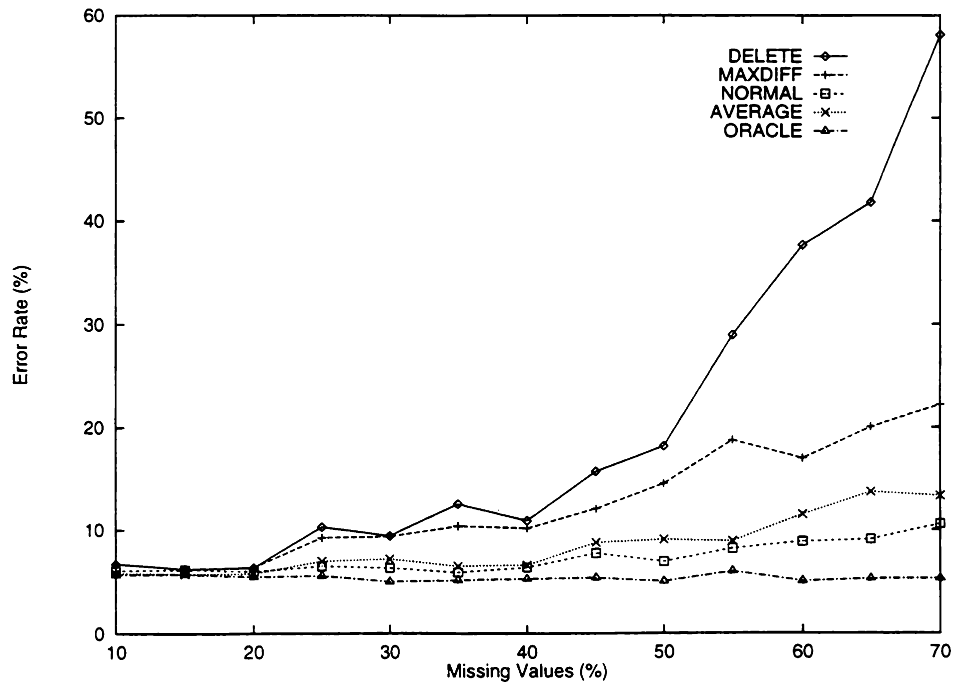

Figure 5.15: Pseudo-iris dataset—20 training instances

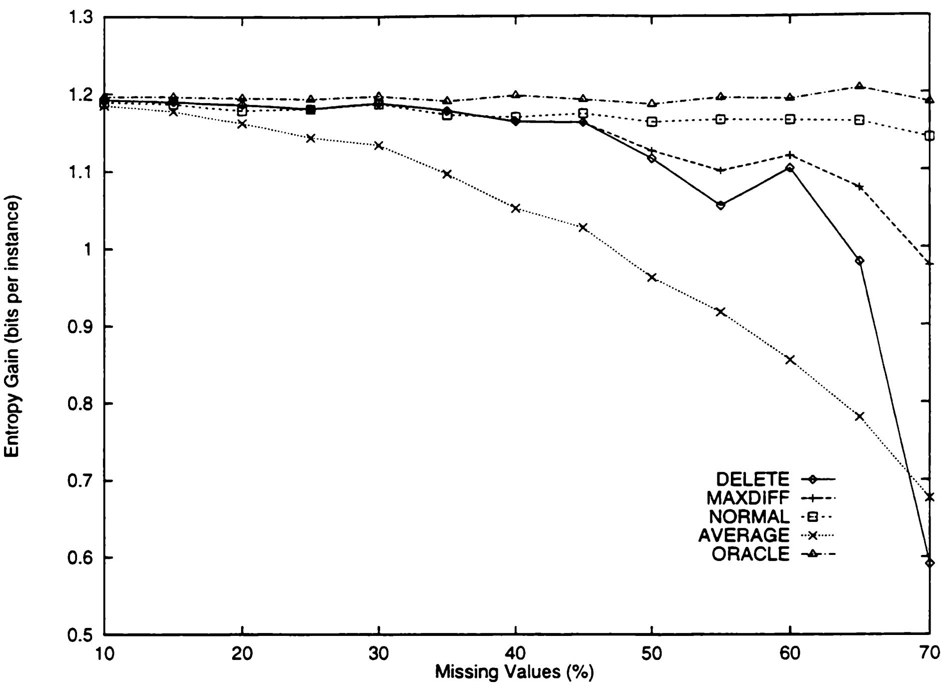

Figure 5.16: Entropy gain for 60 training instances

for both graphs. MAXDIFF and DELETE are significantly different from the other three methods when more than 20% of values are missing. DELETE performs significantly worse than MAXDIFF when more than 40% of values are missing. The AVERAGE error rate is significantly higher than NORMAL when more than 55% of values are missing.

The entropy gain results contain some interesting differences. Figure 5.16 shows the entropy gain for increasing levels of missing values in 60 training instances. The results for DELETE, MAXDIFF, ORACLE, and NORMAL are as expected based on the error rates for these methods. However, the striking difference is that AVERAGE performs significantly worse than all these methods over most of the range shown. The explanation is somewhat complicated.

One can imagine the procedure of choosing a value to substitute for an instance's missing value as translating the instance to a new position in the instance hyperspace, moving along the axis of the attribute containing the missing value. MAXDIFF effectively moves instances to the furthest surface of the bounding hypercube. Instances containing multiple missing values are moved to corners of the hypercube. Thus, these instances only contribute significantly to classification when there are few instances contained within the hypercube. AVERAGE moves instances in a similar manner to MAXDIFF; however, rather than moving them to the bounding hypercube, instances are moved to a hypercube defined by the mean similarity to the test instance for each attribute. The ranking of the class probabilities are determined primarily by the instances inside this hypercube because the instances on the surface of the hypercube will have approximately the same class distribution as occurs globally. Thus, the error rate (which only depends on the most likely predicted class) is not degraded. However, the instances on the surface of the hypercube effectively add noise to the predicted class distribution, producing a significant decrease in entropy gain. This effect is magnified as the proportion of missing values increases, causing the entropy gain to tend towards 0.

The conclusion to be drawn from this experiment is that most sensible methods for dealing with instances containing missing values perform significantly better than discarding the instances. When the proportion of missing values is high or the number of training instances is low, NORMAL performs better than other methods, followed by AVERAGE. The reason for NORMAL's good performance is the assumption that the similarity along each attribute is roughly the same. In this domain, petal length is strongly correlated with petal width. The next experiment examines the various methods in a domain where there is no such correlation.

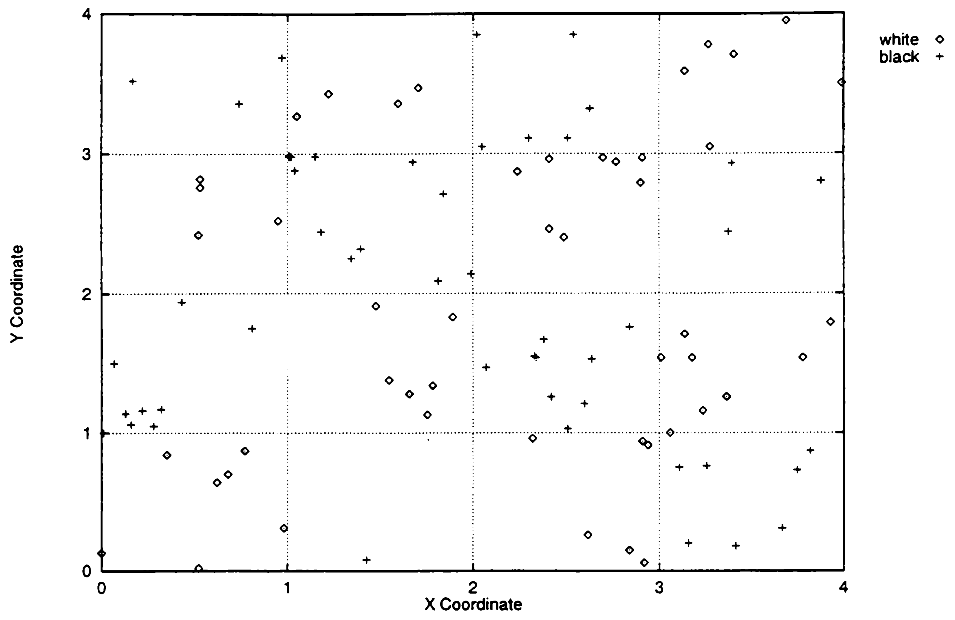

Figure 5.17: 100 example instances from the Checkerboard domain

5.2.4.2 Checkerboard Domain

In this domain, instances have two real valued predictor attributes, an *x* coordinate and a *y* coordinate. An instance is classified as either white or black depending on its position. The arrangement of white and black instances is like a quarter of a checkerboard, as shown in Figure 5.17. Instances are uniformly distributed, so the baseline error rate is 50%.

Figure 5.18 shows the learning rate when 20% of the values are missing. Because the classes are not as easily separable as in the iris domain, learning is much slower. All the methods for dealing with missing values perform significantly worse than ORACLE over the range of training instances shown.

The greatest difference from the results obtained in the iris domain is that NORMAL performs significantly worse than the other methods. This poor performance is due to a lack of clustering in the domain. Because instances are uniformly distributed throughout the predictor space, the assumption that differences in predictor values will be similar is incorrect.

These results are confirmed when we examine the error rate for 300 training instances as the number of missing values increases, as shown in Figure 5.19. Again, NORMAL is significantly poorer than the other methods through the range of missing value levels. AVERAGE performs significantly poorer than MAXDIFF and DELETE until the 70% missing value level. Above 70%, AVERAGE performs significantly better than these two.

When the number of training instances is lower, the results are different. Figure 5.20 shows the results for 50 training instances. NORMAL performs significantly worse than the other methods when the level of missing values is below 50%. Above 65%, there is no significant difference between it and AVERAGE. DELETE and MAXDIFF perform almost identically until the proportion of missing values is so high that DELETE has little or no training data left for classification. At this point, the methods only exhibit marginally smaller error rates than the baseline (DELETE performs worse than the baseline).

The entropy gains for 50 training instances are shown in Figure 5.21. In contrast to the iris domain, AVERAGE performs significantly better than the other methods—entropy gain tends towards zero, rather than being negative. While the error rates for AVERAGE and NORMAL converge in Figure 5.20,

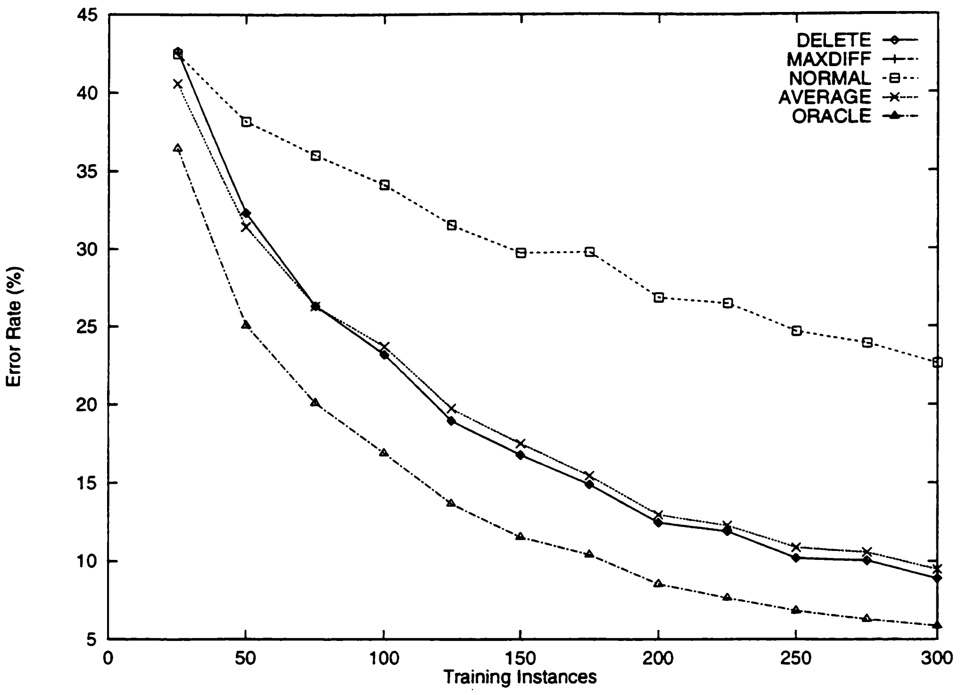

Figure 5.18: Checkerboard dataset—20% missing data

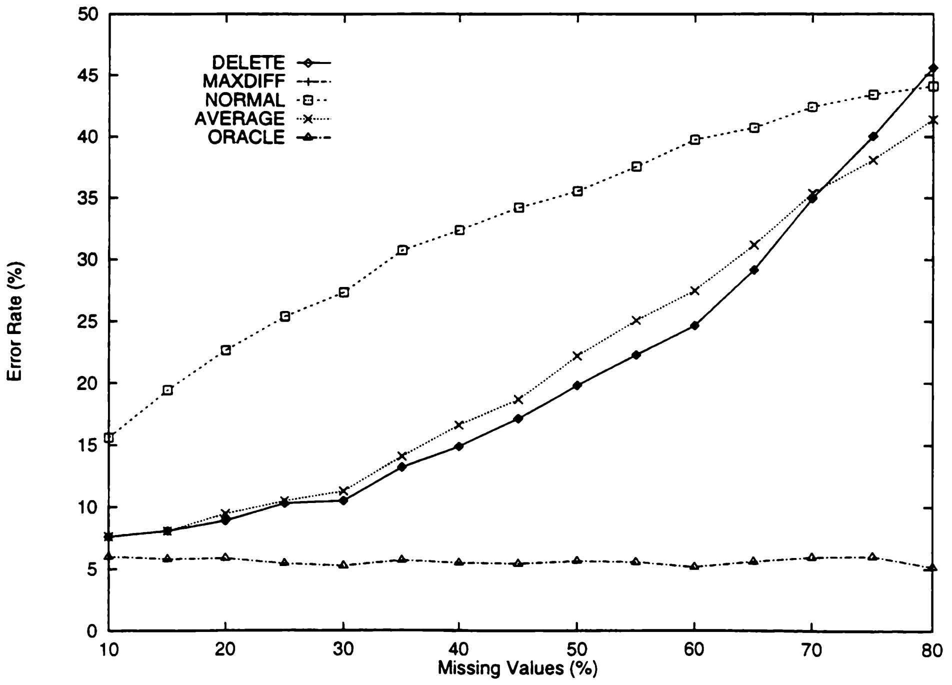

Figure 5.19: Checkerboard dataset—300 training instances

NORMAL performs significantly worse than AVERAGE for the whole range shown. Above 40% missing values, NORMAL, MAXDIFF and DELETE have negative entropy gains.

The results of these experiments are quite different from those obtained in the previous section. In this domain, NORMAL performs considerably worse than the other methods, particularly when the level of missing values is low enough to obtain results better than the baseline error rate. This poor performance is because NORMAL's assumption of correlated predictor attributes is not true of this domain.

In summary, these experiments show that most methods for handling missing values are capable of performing better than discarding the instances altogether. However, some methods are based on assumptions that may not necessarily be true of the domain. AVERAGE is a good "middle of the road" method that is able to perform well under a range of conditions. However, additional knowledge about the domain can provide assistance as to which method might be more suitable. If it is known that attributes are correlated, NORMAL is likely to be a better choice. NORMAL could potentially be improved by incorporating the inter-attribute correlation in the missing value similarity calculation. AVERAGE is independent of the other attributes—its performance may be improved by considering the training instance classes. For example, the similarity to a training instance's missing value would become the expected similarity to the values of other training instances with the same class.

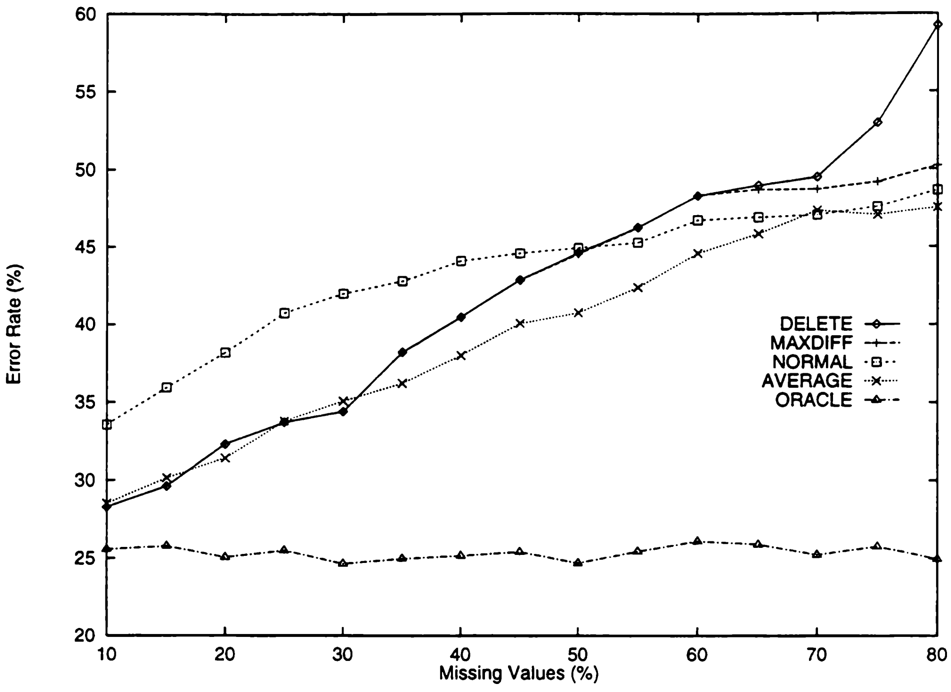

Figure 5.20: Checkerboard dataset—Error rate for 50 training instances

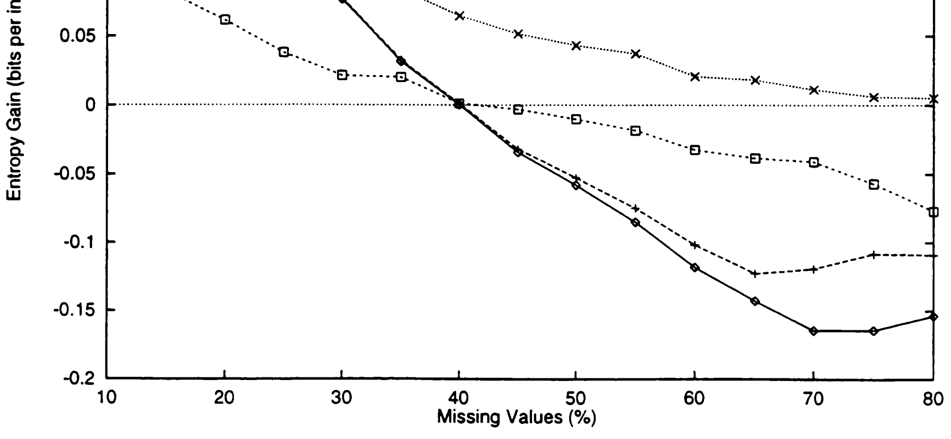

Figure 5.21: Entropy gain for 50 training instances

5.2.5 Automatic Assignment of Symbolic One-step Probabilities

The aim of this section is to establish under what circumstances the non-independent symbolic measure developed in Section 4.3.8, using automatic one-step probability assignment, performs better than the measure for independent symbols developed in Section 4.3.7. As seen in Section 5.2.2.1, one-step probabilities may be manually assigned to improve performance; here we wish to examine the automated method for assigning one-step probabilities.

5.2.5.1 Discretised Pseudo-iris Domain

This experiment utilises the pseudo-iris dataset generator from Section 5.2.4.1, but the petal length and petal width attributes are discretised to form symbolic attributes. The discretisation method divides the range of values for an attribute into a number of equal sized partitions. The resulting symbols should be assigned similarities that are related to their original ordering. The following two methods are compared:

INDEPENDENT: The dataset is classified using the similarity function for independent symbols developed in Section 4.3.7.

NON-INDEPENDENT: The dataset is classified using the similarity function for non-independent symbols developed in Section 4.3.8. The one-step transformation probabilities are automatically assigned using the method given in Equation 4.18.

Figure 5.22 shows the learning rate for the two methods when each attribute is divided into 5 bins. The INDEPENDENT measure generally performs significantly better than the NON-INDEPENDENT measure when the number of training instances is above 15. This is a result of two factors. First, INDEPENDENT performs well because there are few symbols per attribute—relatively few training instances are required to obtain coverage of at least one training instance per test point in the instance space. INDEPENDENT is primarily expected to perform poorly when no training instances exactly match each test instance. Second, there are too few symbols per attribute for customised symbol similarities to be significantly advantageous.

The equivalent entropy gain curve is shown in Figure 5.23. In contrast to the error rate results, NON-INDEPENDENT performs significantly better than INDEPENDENT when there are more than 60 or less than 20 training instances. As the number of training instances increases, the automatic method for setting one-step probabilities is able to more accurately represent the probable class distributions throughout the instance space, but not enough to result in changes to the predicted class.

Figure 5.24 shows the learning rate when each attribute is divided into 15 partitions. In contrast to the previous experiment, INDEPENDENT performs significantly poorer than NON-INDEPENDENT at all levels of training instances shown. This is partially because many areas of the instance space contain no training instances; when there is a symbol mismatch on one attribute, INDEPENDENT takes the most probable other symbol as closest, often resulting in misclassification. NON-INDEPENDENT is able to piece together relationships between symbols so its learning rate is faster.

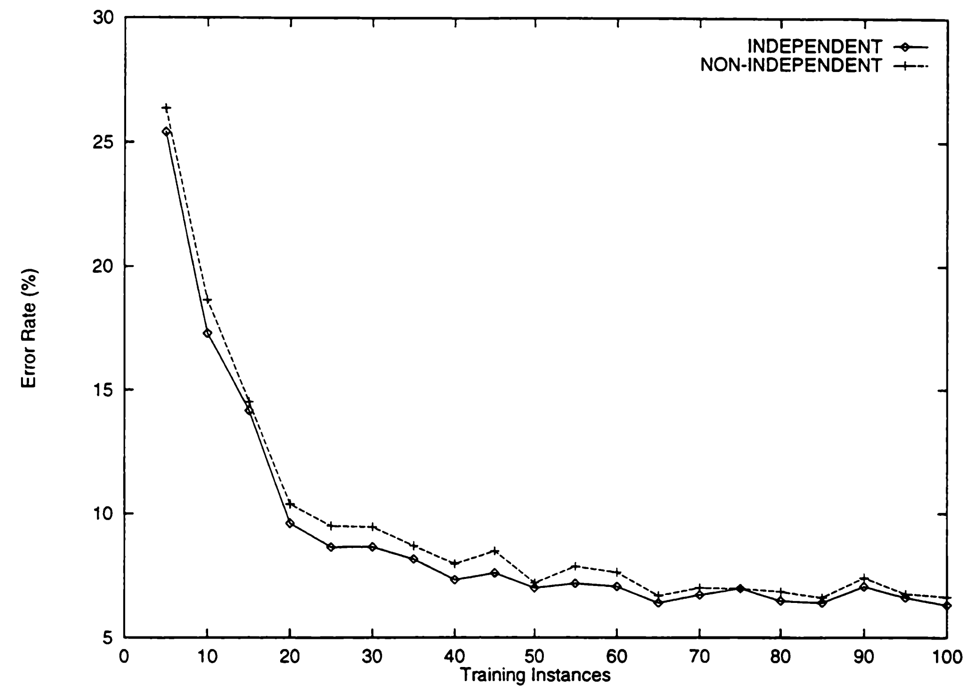

Figure 5.22: Error rate for pseudo-iris domain—5 bins per attribute

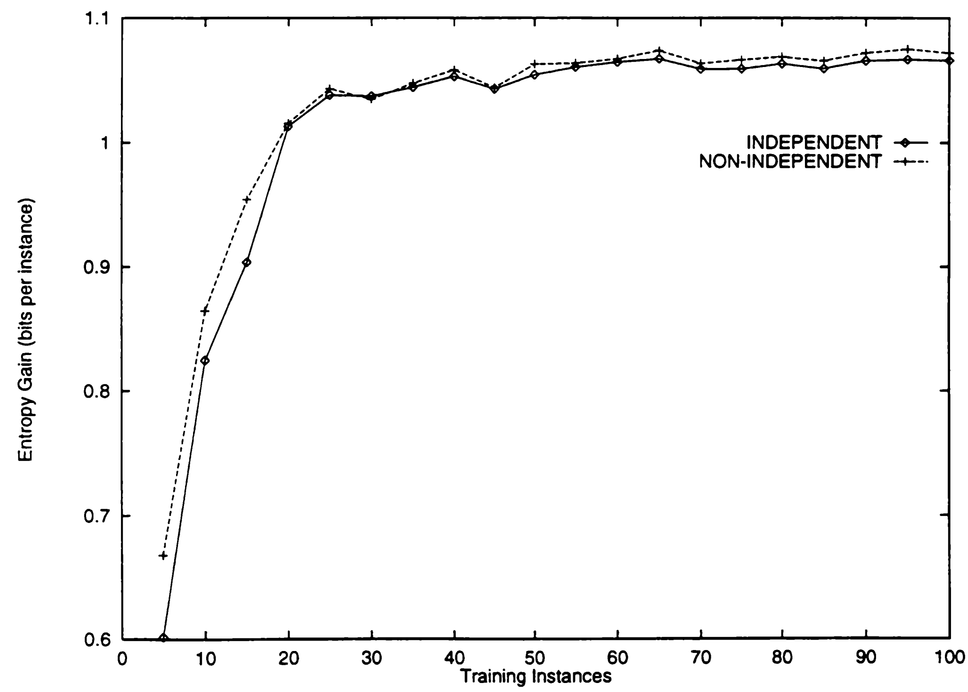

Figure 5.23: Entropy gain for pseudo-iris domain—5 bins per attribute

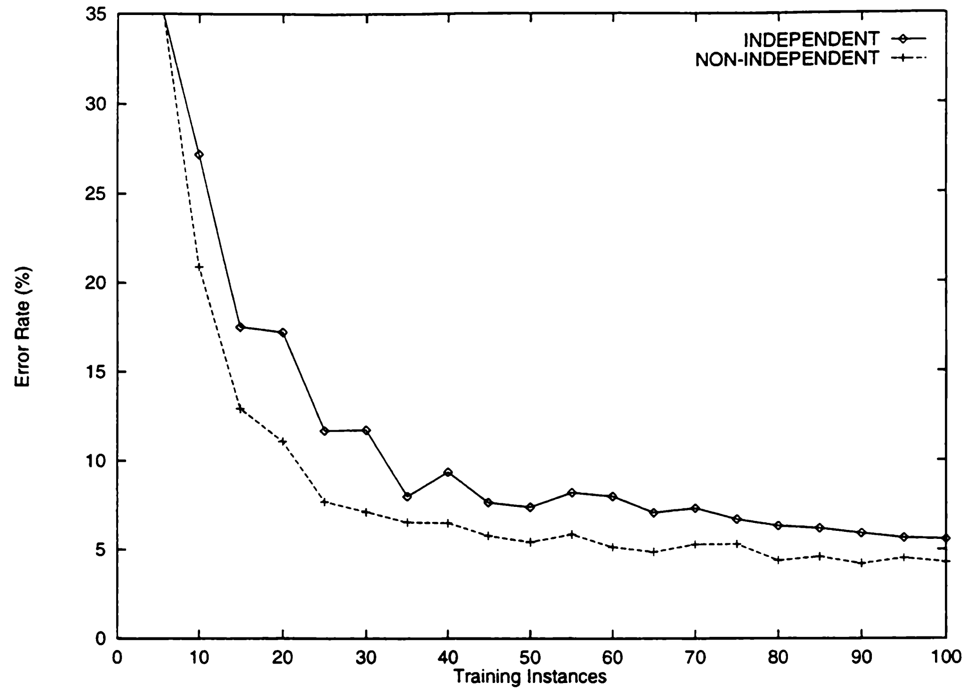

Figure 5.24: Error rate for pseudo-iris domain—15 bins per attribute

These experiments indicate that the automated method for assigning symbol transformation probabilities can significantly outperform the class-blind measure for symbolic attributes in domains with many symbols for each attribute. This domain is relatively simple, so the next section considers a more complex real-world domain.

5.2.5.2 Phoneme Domain

The objective in the phoneme domain is to predict the correct pronunciation of English words, given a database of correct pronunciations. This task has been examined with artificial neural nets (Sejnowski and Rosenberg, 1987) and nearest neighbour methods (Stanfill and Waltz, 1986). Predictor attributes include: the current letter for which pronunciation is to be predicted; the three

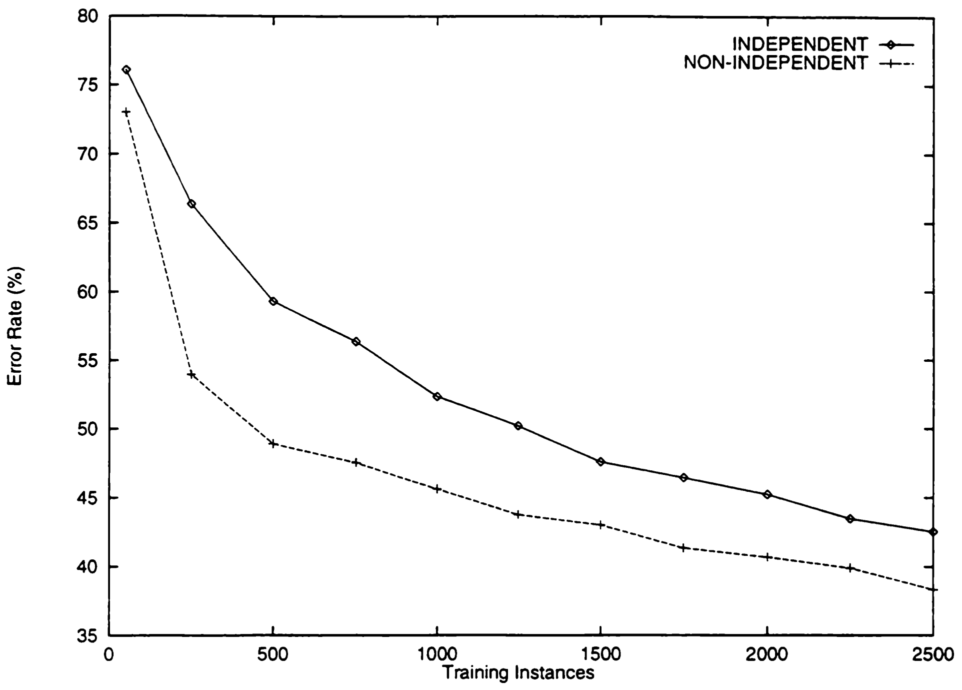

Figure 5.25: Learning rate for the phoneme domain

preceding letters in the word; and the three succeeding letters in the word. These attributes are labelled "cn", "$cn-1$", "$cn-2$", "$cn-3$", "$cn+1$", "$cn+2$", and "$cn+3$" respectively. There are 27 symbols per attribute, and 59 possible classes. Each n-letter word is represented in the database by n instances, one for each letter to be pronounced. 1000 of the most commonly used English words were used in the dataset—those that are not used in training are used in testing. The total number of instances is 5603. Figure 5.25 shows the learning rate for the independent symbolic metric and the non-independent metric using automatic one-step probability assignment. Each data point is the result of ten trials.

In this domain, NON-INDEPENDENT performs significantly better than INDEPENDENT at all the levels of training instances shown, largely due to the high number of symbols per attribute. There are insufficient training instances to fill the instance space, so INDEPENDENT performs poorly. NON-INDEPENDENT builds class-based inter-symbol similarities, allowing it to perform much better when there are mismatches between test and training instances.

5.2.6 Automatic Stop Parameter Setting

This section evaluates the automatic method for setting values for the stop parameter as described in Section 5.1.3.2. In particular, its performance is compared against the manual blend setting method from Section 5.1.3.1.

5.2.6.1 Phoneme Domain

Figure 5.26 shows the learning rate on the phoneme domain for automatic blend setting, in comparison with the default manual blend value of 20%. Results labelled IND use the independent symbol similarity function, and those labelled NON-IND use the non-independent symbol similarity function with automatic one-step probability assignment (the learning rates for manual blend are therefore the same as those obtained in Section 5.1.3.2). In this domain, the automatic blend setting method improves the initial learning rate significantly. After 500 training instances, the error rate is approximately 15% lower than each corresponding error rate using the manual blend setting. If the attributes are ranked according to the average stop probability assigned, the (descending) order is "cn", "$cn-1$", "$cn+1$", "$cn-2$", "$cn+2$", "$cn-3$", and "$cn+3$." Thus, the importance of each character position is proportional to how near it is to the character being pronounced, with a slight bias towards preceding characters.

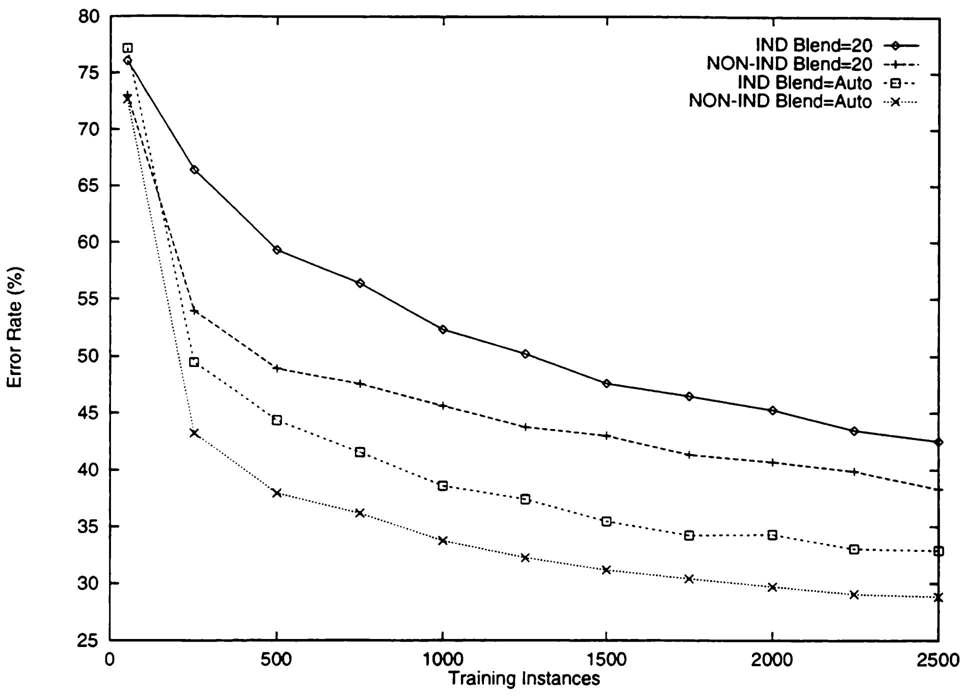

Figure 5.26: Phoneme domain—auto versus manual blend

5.2.6.2 Checkerboard Domain

Figure 5.27 shows the learning rate on the checkerboard domain. In contrast to the results from the last section, the automatic method performs significantly worse than manually setting the blend parameter to its default. Because the blend value is determined for each attribute independently, the blend optimisation effectively sees training instances projected onto one attribute at a time, and the class distribution is uniform with respect to each attribute. Only by considering both attributes at once is the clustering apparent. This domain will prove difficult for any machine learning scheme that attempts optimisation based on a single attribute at a time—for example, C4.5 rarely achieves better than a 50% error rate in this domain.

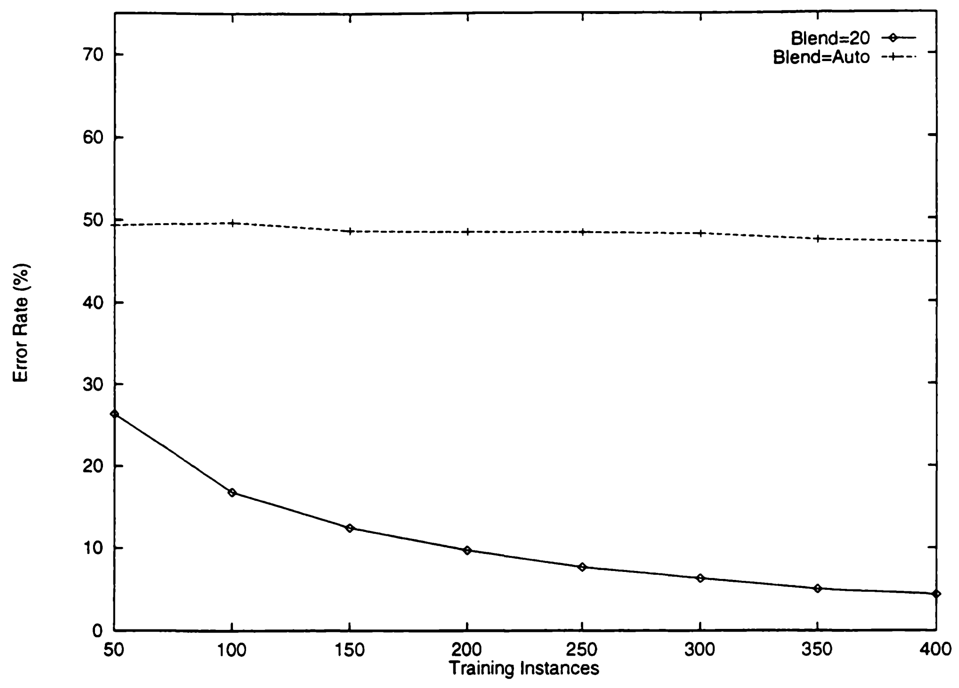

Figure 5.27: Checkerboard domain—auto versus manual blend

In summary, the automatic blend parameter setting method is able to determine attribute importance in domains where the class distributions appear non-random with respect to individual attributes. However, in domains where the relevance of attributes is only readily apparent when training instances are projected onto multiple attributes (as in the checkerboard domain), the automatic method for setting blend parameters performs poorly.

5.2.7 Capturing Domain Information

The objective of this section is to show that customising the similarity function to the domain can give increased performance. There are two types of domain information that should be treated separately—information about the underlying reality of which the instances are representations (which is independent of concept boundaries), and information about the location of class boundaries. Often the task for machine learning algorithms is to infer the location of class boundaries from the training data—providing this information explicitly could be interpreted as "cheating." However, it is perfectly acceptable to provide information about the instance representations. To illustrate these differences we first examine the UV domain.

5.2.7.1 Ultra-violet Domain

In Chapter 3 it was shown that IB1 learned the ultra-violet domain concepts significantly faster when its similarity function was modified to reflect the modulo nature of the time of day. In this experiment these effects are further examined, using the UV4 dataset as described in Section 5.2.2.1. That is, the time of day is represented as a single symbolic attribute, with one symbol for each hour of the day. The following similarity functions are used.

INDEPENDENT: Classification employs the similarity measure for independent symbols developed in Section 4.3.7. In the context of the ultra-violet domain, this similarity function assumes there is no special relationship between one hour and any other hour.

AUTO: Classification employs the similarity measure for non-independent symbols developed in Section 4.3.8. The one-step transformation probabilities are automatically assigned using the method given in Equation 4.18. That is, there are special inter-hour relationships, but they must be inferred from the training instances.

MANUAL1: Uses the same similarity measure as AUTO. One-step probabilities are assigned as follows: each hour has equal probability of transforming to the preceding hour, as the following hour (with hour 0 following hour 23). This similarity function embodies the

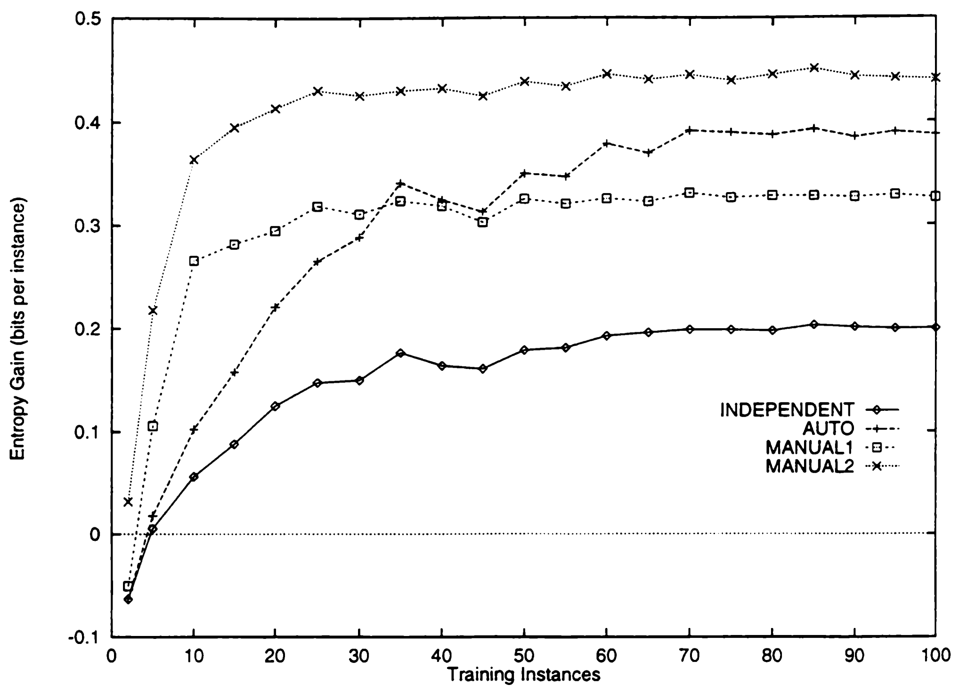

Figure 5.28: Entropy gain for various similarity functions on UV4 dataset

domain knowledge about the ordering of hours, but no information about the domain concepts.

MANUAL2: Uses the same similarity measure as AUTO. The hours are treated as three groups, hours 16–10, hours 11–14, and hour 15. Within each group, one-step probabilities are assigned as follows: each hour has equal probability of transforming to the preceding hour as the following hour (except where this would cross between groups, in which case the transformation returns the instance to itself). This similarity function is therefore provided with information relating to the domain concepts.

Figure 5.28 shows the entropy gain results for a range of training instances. MANUAL2 performs significantly better than the other functions for the entire range shown. This is to be expected, since the similarity function has been given concept boundary information. INDEPENDENT performs significantly worse than the other functions for the entire range shown. We would not initially expect INDEPENDENT to perform significantly worse than the other methods once the number of training instances is large enough to ensure at least one training instance for each hour. However, this is not what we find—INDEPENDENT's entropy gain converges to around 0.2 bits per instance. This effect is due to the default blend value of 20%. Since each hour constitutes approximately 4% of instances, a large proportion of the prediction is derived from instances with an hour different to the current test instance. INDEPENDENT weights each of these other hours equally—however the majority will be UV-LOW instances. With a blend value of 20%, this bias toward the UV-LOW class results in misclassification of all UV-HIGH instances. A similar bias provides a bound on the performance of MANUAL1—the effect is not as prominent since MANUAL1 has hour ordering information. When the blend value is 5%, predictions are primarily based on instances with the same hour as the test instance—INDEPENDENT converges to 0.32 bits per instance and MANUAL1 converges to 0.42 bits per instance; both are closer to the limit of 0.44 bits per instance obtained by MANUAL2. MANUAL1 performs significantly better than AUTO below 30 training instances, and significantly worse above 45 training instances. AUTO learns some of the concept-specific inter-hour relationships that MANUAL2 is provided with manually. Error rate results (not shown) show no significantly different trends from the entropy gain results.

Customisation of the similarity function to include domain information clearly improves the learning rate, particularly when the number of training instances is limited. The following experiment describes an interesting real-world domain that permitted domain customisation[4].

5.2.7.2 Wasp Domain

Predictor attributes in this domain consist of various measurements taken from wasp nests, such as the width, height, and depth of the nest, the type of nest site, the direction of the nest entrance, and the number of layers in the nest. The task is to predict which of two species of wasp (the common wasp *Vespula vulgaris*, and the German wasp *Vespula germanica*) constructed the nest. The dataset contains 226 instances and 12 predictor attributes. 167 of the instances are for the species *Vespula vulgaris*, so the baseline error rate is 26%. It turns out that differentiating the two species given these predictor attributes is actually very difficult (Donovan *et al.*, 1992). However, some differences between species have been identified. For example, German wasps showed no preference for direction of nest entrances while common wasps' nests were more numerous in locations exposed to morning sun. The bias is not significant enough to alter the default prediction. The nest entrance direction attribute is interesting in that it consists of 9 possible values: North, Northeast, East, Southeast, South, Southwest, West, Northwest, and Upwards (judged as when the vertical angle of the entrance was above 45°). There are obvious relationships between the symbols that could be captured within a custom similarity function. In the following experiment we classify the wasp

[4] Finding a real-world domain suitable for customisation is not an easy task, as this requires a dataset with three properties: that there is information in the data for a ML scheme to exploit; that there are attributes better suited to a custom metric; and that those attributes are actually relevant. The relative scarcity of these datasets may indicate an unintentional bias towards creating datasets with only simple numeric and symbolic attributes.

dataset using only the direction attribute for prediction. The following similarity functions are examined.

INDEPENDENT: Classification uses the similarity measure for independent symbols developed in Section 4.3.7. This similarity function assumes there is no special relationship between one direction and any other direction.

AUTO: Classification uses the similarity measure for non-independent symbols developed in Section 4.3.8. One-step transformation probabilities are automatically assigned using the method given in Equation 4.20—that is, there are inter-direction relationships, but they must be inferred from the training instances.

MANUAL: Classification uses the same similarity measure as AUTO. One-step probabilities are assigned as follows. The eight primary directions are taken as modulo, with an equal transformation probability from a primary direction to its neighbours. In addition, we assume a direction may transform to the "Upwards" direction with half this probability. These assumptions (which constitute domain knowledge) assign one-step probabilities of 0.4 to each of the former transformations, and 0.2 to the latter transformation. The one-step probability from the "Upwards" direction to the eight primary directions is assumed to be equal (that is, 0.125). This similarity function embodies intuitive domain knowledge about the ordering of the compass directions.

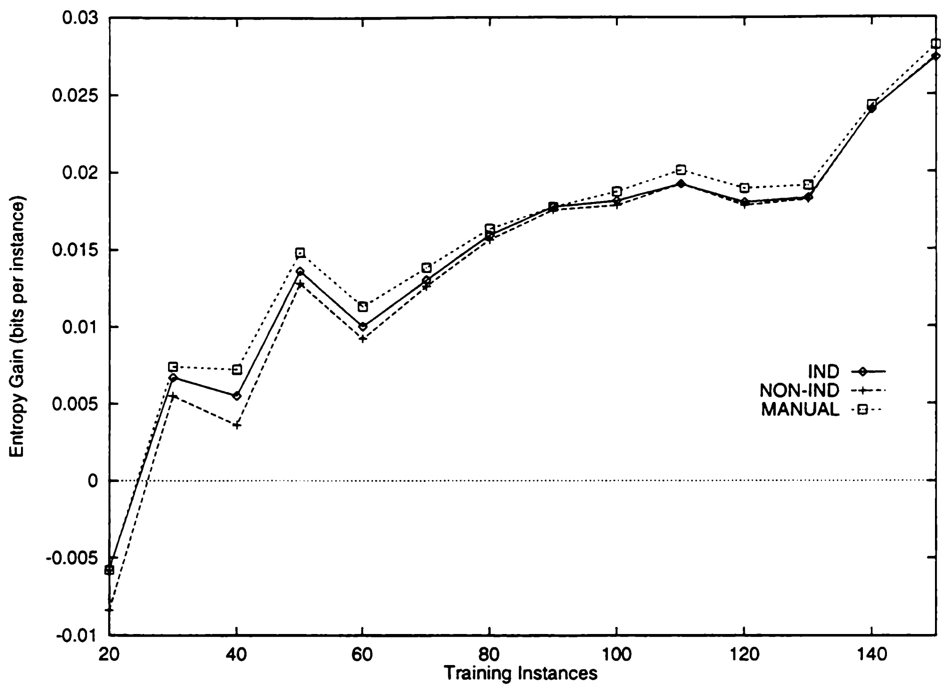

Figure 5.29: Entropy gain for various similarity functions on wasp domain

For this experiment the dataset is randomly split into a training set with a specified number of instances, and a test set containing the remaining instances. For each level of training data, results are averaged over 50 trials. Due to the large number of common wasps in relation to the relevance of the nest entrance direction, the error rate for the three similarity functions is never better than the default accuracy—the only significant difference between the methods is that AUTO performs worse than the other methods below 80 training instances (due to the relative lack of training data, as discussed in Section 5.2.5). However, the entropy gain for these similarity functions (Figure 5.29) shows some interesting differences. As with the error rate, AUTO performs significantly worse than the other methods below 80 training instances. MANUAL performs significantly better than both INDEPENDENT and AUTO above 30 training instances (except between 80 and 90 training instances, and 140 training instances, where the difference is not significant).

The domain knowledge captured within MANUAL results in improved performance in this domain.

There are several conclusions to be drawn from these experiments. Domain customisation provides the most benefit when the quantity of training is relatively low. There are two reasons for this. First, discontinuities in the similarity function are more likely to cause classification errors when training data is limited (because the average distance between a test instance and the nearest training instances is increased). Second, as the quantity of training data increases, some domain information can be learned (for example, by the automatic method for one-step probability assignment). The benefits of domain customisation are more visible when there are relatively few non-customisable attributes (simply because the contribution from the customised attributes is lower).

5.2.7.3 Varying Numbers of Features

The previous experiments have been dealing with instances that can be represented naturally as a fixed number of attributes. In tasks such as comparing multiple high and low pressure systems of weather maps, the number of features can vary between instances. In this experiment instances have between one and three numeric features, each in the range 0-10. The class of an instance is determined by taking the average of the feature values and converting the result to one of ten symbolic bins. To represent these instances with a constant number of attributes (which is required by typical instance-based learners), missing values are added to instances until each instance contains three predictor attributes. The following classifiers are examined.

NORMAL: Classification uses the similarity measure for numeric attributes developed in Section 4.3.3. This similarity function assumes there is no special relationship between attributes.

MULTIREAL: Classification uses the similarity measure for multiple attributes developed in Section 4.3.9. This function considers possible mappings between attributes.

IB1 K=1: Classification uses the IB1 instance-based learner (Aha, Kibler and Albert, 1991), with predictions obtained from the single nearest neighbour.

IB1 K=3: Classification uses the IB1 instance-based learner, with predictions obtained by voting among the three nearest neighbours.

IB1 K=5: Classification uses the IB1 instance-based learner, with predictions obtained by voting among the five nearest neighbours.

The number of training instances for this experiment varies from 2 to 100, and the number of test instances is 500. The entropy gain results are presented in Figure 5.30. The benefits of using an appropriate similarity function for this domain are clear—above ten training instances MULTIREAL performs significantly better than the other methods. MULTIREAL is not sensitive to either the position of attribute values in the representation or to the number of attribute values present. IB1 employing a single neighbour for prediction performs very poorly; in order to make a correct prediction, there must usually be an exact match to the current test instance. As more neighbours are considered, IB1's performance improves, but remains significantly worse than NORMAL when more than ten training instances are present. This experiment illustrates the potential for K* theory to allow instance-based learning to be applied to domains that don't naturally lend themselves to the usual instance representation of a fixed number of either symbolic or numeric attributes.

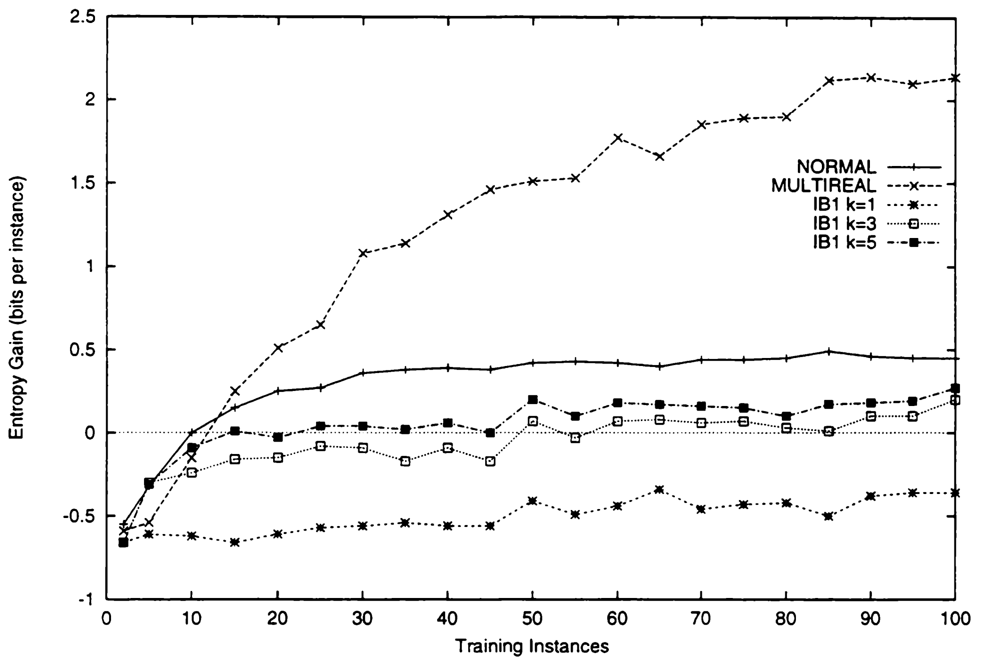

Figure 5.30: Entropy gain for various classifiers with varying numbers of attributes.

5.2.8 Comparison with Other Machine Learning Schemes

This section evaluates the K* classifier in relation to other machine learning schemes. The purpose is not to show that K* has superior performance to all other machine learning schemes, but that it performs well under typical circumstances. The following schemes are employed: a state of the art decision tree learner C4.5 (Quinlan and Rivest, 1989); and the instance-based learners IB1 (Aha, Kibler and Albert, 1991) and PEBLS (Cost and Salzberg, 1993). IB1 is perhaps the simplest practical instance-based learner; it employs the simple distance function described in Chapter 3, treats missing values as maximally different from the current value. IB1 bases predictions on a

Dataset (abbr)	Size	Missing	Classes	Binary	Symbolic	Numeric
Breast-cancer (BC)	286	9	2	3	6	0
Chess (CH)	3196	0	2	35	1	0
Glass (GL)	214	0	7	0	0	9
Glass2 (G2)	163	0	2	0	0	9
Heart disease (HD)	303	7	5	1	0	12
Hepatitis (HE)	155	167	2	0	0	19
Horse colic (HO)	368	1927	2	2	13	7
Hypothyroid (HY)	3163	5329	2	18	0	7
Iris (IR)	150	0	3	0	0	4
Labor (LA)	57	326	2	3	5	8
Lymphography (LY)	148	0	4	9	6	3
Mushroom (MU)	8124	2480	2	4	18	0
Sick euthyroid (SE)	3163	5329	2	18	0	7
Soybean (SO)	47	0	4	13	8	0
Vote (VO)	435	392	2	16	0	0
Vote no phys (V1)	435	381	2	15	0	0

Table 5.1: Dataset characteristics

majority vote among the *k* nearest neighbours (values of *k*=1, *k*=3, *k*=5, and *k*=7 were used, labelled as 1NN, 3NN, 5NN, and 7NN respectively). C4.5 and PEBLS on the other hand are more sophisticated schemes—C4.5 can build complex decision trees involving many attributes, and PEBLS employs a sophisticated distance function for symbolic attributes in its classification. K* results are obtained with default settings; that is, the blend value is 20%, and symbolic attributes use the simple function that assumes independent symbols. In addition, results are obtained for two different K* settings: K*(n) differs from standard K* by employing the symbolic function for non-independent symbols with automatic one-step probability assignment; and K*(e) differs from standard K* in that it uses automatic blend setting.

Many of the datasets used are commonly seen in the machine learning literature. The datasets are the same as those used by Holte (1993), and were originally taken from the UCI Machine Learning Database Repository. Table 5.1 lists the datasets and their characteristics. "Size" shows the total number of instances. "Classes" lists the number of possible classes. "Binary",

Data	K*	C4.5	1NN	3NN	5NN	7NN	PEBLS	K*(n)	K*(e)
BC	27.05	30.10+	30.93+	29.03+	**26.80**	27.13	32.95+	28.70+	30.43+
CH	4.15	**0.77–**	10.10+	5.41+	5.10+	5.44+	2.98–	4.29+	4.66+
G2	**16.73**	27.42+	22.84+	23.49+	24.95+	26.62+	23.78+	**16.73**	31.20+
GL	**26.63**	33.04+	31.51+	36.16+	40.93+	40.88+	39.45+	**26.63**	38.90+
HD	25.09	27.92+	23.57	19.84–	**19.18–**	19.65–	22.80–	25.17	28.23+
HE	19.70	35.25+	18.94	17.06–	**16.75–**	16.98–	19.92	19.70	20.15
HO	23.42	23.26	21.92–	18.56–	**18.18–**	18.21–	21.73–	23.17	22.66
HY	2.20	8.99+	2.96+	2.82+	2.88+	2.95+	3.37+	**2.17**	2.67+
IR	5.73	6.27	5.02–	4.78–	**4.08–**	**4.08–**	5.88	5.73	6.27
LA	**9.05**	29.68+	24.21+	20.00+	21.05+	22.74+	9.26	**9.05**	16.84+
LY	**17.20**	24.72+	20.32+	19.12+	18.40	18.24	17.44	17.52	24.24+
MU	**0.00**	**0.00**	**0.00**	**0.00**	0.01	0.02	**0.00**	**0.00**	**0.00**
SE	**5.70**	24.65+	7.78+	7.54+	7.33+	7.29+	6.51+	5.78+	8.29+
SO	**0.00**	2.00+	**0.00**	**0.00**	**0.00**	1.50	**0.00**	7.00+	**0.00**
VO	6.73	7.49	7.27	6.62	6.76	6.81	**5.51–**	6.73	6.57
V1	**9.19**	15.81+	11.84+	9.86+	9.70	9.65	11.49+	**9.19**	11.62+

Table 5.2: Error rates for UCI datasets

"Symbolic", and "Numeric" indicate the numbers of each type of predictor attributes. "Missing" shows the number of unknown attribute values. The datasets were partitioned into two-thirds training, one-third test instances. 25 different partitions were made for each dataset. Schemes were run on all 25 partitions and the results averaged. These experiments were obtained using the WEKA machine learning workbench (Holmes, Donkin and Witten, 1994).

Table 5.2 shows the error rates each scheme achieved for the datasets. Error rates postfixed by '+' are significantly higher (i.e. worse than) than the error rate obtained by K*. Error rates postfixed by '–' are significantly lower than the error rate obtained by K*. The best error rates are shown in bold.

Overall, the K* classifier performs well in comparison to the other schemes used. For example, K* performs significantly better than the decision tree methods for 11 of the 16 datasets. Only once did C4.5 perform significantly better than K*. K* also performs comparably to the instance-based learners 1NN and PEBLS. K* has significantly lower error rates than 1NN for 9 of the

Data	K*	C4.5	1NN	3NN	5NN	7NN	PEBLS	K*(n)	K*(e)
BC	**0.08**	–0.04–	–0.02–	0.05–	**0.08**	**0.08**	–0.05–	0.07–	0.00–
CH	0.53	**0.93+**	0.50–	0.54+	0.52–	0.49–	0.66+	0.53	0.61+
G2	**0.41**	0.11–	0.24–	0.30–	0.29–	0.27–	0.23–	**0.41**	0.12–
GL	**1.08**	0.64–	0.74–	0.96–	0.96–	0.95–	0.44–	**1.08**	0.68–
HD	0.34	0.19–	0.29–	0.41+	0.44+	**0.45+**	0.31–	0.34	0.25–
HE	0.15	0.011–	0.11	0.17	**0.18+**	**0.18+**	0.11	0.15	0.04–
HO	0.23	0.23	0.21	0.29+	**0.30+**	0.29+	0.22	0.23	0.22
HY	0.15	**0.16+**	0.10–	0.11–	0.11–	0.11–	0.11–	0.15	0.12–
IR	1.21	1.16	1.20	1.22	**1.24+**	**1.24+**	1.17–	1.21	1.20
LA	**0.55**	0.12–	0.24–	0.35–	0.28–	0.24–	0.51	**0.55**	0.39–
LY	**0.62**	0.29–	0.47–	0.58–	0.56–	0.54–	0.55–	0.61	0.39–
MU	0.74	**1.00+**	0.74	0.74	0.74	0.74	0.74	0.74	0.74
SE	**0.19**	0.16–	0.11–	0.13–	0.14–	0.13–	0.16–	0.18–	0.09–
SO	**1.80**	1.62–	**1.80**	1.79	1.74–	1.59–	**1.80**	1.79	**1.80**
VO	0.58	**0.71+**	0.50–	0.58	0.58	0.57–	0.59	0.58	0.57
V1	**0.51**	0.46–	0.45–	0.49–	0.50–	0.50–	0.45–	**0.51**	0.45–

Table 5.3: Entropy gains for UCI datasets

datasets, and significantly higher error rates for 2 datasets. K* has significantly lower error rates than PEBLS for 6 datasets, and significantly higher error rates for 4 datasets. The primary explanation for the better overall performance of K* over these instance-based learners is that 1NN and PEBLS make predictions based on the single nearest neighbour, while K* incorporates all training instances. Increasing the number of neighbours considered (bringing the learners closer to the k* philosophy) increases their performance—IB1 using voting among the 5 nearest neighbours performs almost as well as K*.

Table 5.3 shows the entropy gain results for each of the schemes. Entropy gain results postfixed by '+' are significantly higher (i.e. better than) than the corresponding K* result. Entropy gain results postfixed by '–' are significantly lower than the corresponding K* result. The best entropy gain results are shown in bold. In general, a positive score signifies that the scheme has managed to extract significant domain information in making its predictions.

K* performs significantly better than the other schemes for the majority of datasets, with some caveats. The entropy gain calculation is different for the instance-based learners and the decision tree learners—a different method is used to combine each scheme's predicted distribution with the naive distribution when dealing with the zero-frequency problem. The entropy gain figures are similar in practice, but care should be exercised when making conclusions about performance differences between instance-based and decision tree schemes. As with the results in Table 5.2, much of K*'s good performance is due to weighting all neighbours into its predictions.

When examining the entropy gain figures for individual datasets, one interesting feature is that the results for the BC dataset are often negative or close to zero. This indicates that there is little information to be learned from the predictor attributes (indeed, the baseline error rate for this dataset is 30%, similar to all the schemes' error rates). A similar comment may be made about the HE dataset, where the entropy gain figures are very small.

K*(e) performs poorly on many of the other datasets. One cause is that individual attributes are often poor predictors. K*(e) sets the stop parameter for attributes independently, and if an individual attribute is a poor predictor, it is effectively discarded by the parameter optimisation. Poor performance can also arise when this is not the case, particularly for enumerated attributes. Often the optimisation assigns high relevance to all attributes that have some relevance because the optimiser cannot determine whether other attributes may be more relevant. This problem occurs on the VO dataset, where there are many attributes that are reasonable predictors. One attribute is an excellent predictor, but its weight is not set higher than the other attributes.

K*(n) does not perform significantly better than K* in spite of using a more advanced metric for symbolic attributes. K*(n) has a slightly lower error rate than K* on two datasets, but significantly higher on four datasets. However, according to the findings of Section 5.2.5, K*(n) is only expected to perform better than K* on datasets with many symbolic attributes and with multiple symbols per attribute. Most of the domains contain predominantly numeric and binary attributes. Examining Table 5.1, the datasets K*(n) is most likely to perform on well are BC, HO, LA, LY, MU, and SO. K*(n) does achieve lower error rates than K* for the BC and HO datasets (although the difference is not significant for BC). K*(n) achieves significantly higher entropy gains than K* on all these datasets except LA and SO—on the SO dataset K*(n) performed significantly poorer than K*. The reason why K*(n) did not perform as well as expected in LA and SO is that these datasets contain few instances. As identified in Section 5.2.5, learning accurate one-step probabilities requires sufficient training data.

In summary, K* performs well in comparison to other machine learning schemes. However, on the datasets tested, the automatic method for setting the blend parameter described in Section 5.1.3.2 does not perform as well as expected, primarily because the parameters are set for each attribute independently. The advanced metric for symbolic attributes can improve the performance of the classifier on datasets where there are large symbolic attributes, provided there is sufficient training data. A heuristic method for selecting the appropriate metric for symbolic attributes could take these factors into consideration.

5.3 Conclusions

This chapter described the practical implementation of an instance-based learner using similarity functions developed within the proposed framework. The basic similarity functions employed were those developed as examples in

Chapter 4. The instance-based learner required two additional problems to be solved.

The first of these problems was how to choose appropriate values for the free parameters of the similarity function. Ideally these parameters should be set to reflect the relative importance of attributes to the classification domain. The first approach to setting these parameters effectively assumed that each attribute has approximately equal relevance, and the second approach attempted to automatically determine each attribute's relevance by examining the entropy of the predicted class distribution. Both methods currently set the parameters for each attribute independently; it turns out that this approach causes problems during classification.

The second problem was how to use the similarities between test and training instances to provide a prediction. This problem is treated following the K* philosophy by incorporating transformations to all training instances. Instances are effectively weighted by their transformation probability. A predicted probability distribution can be calculated for categorical attributes, and the expected value calculated for numeric attributes.

In empirical evaluation, it was shown that the K* classifier meets the objectives set out in Chapter 1: different attribute types are themselves treated coherently and can be combined avoiding biases; missing values can be handled intelligently; domain information can be used to customise the similarity function; and the scheme performs well in comparison to other methods.

Although different attribute types are not always directly comparable due to the different information they are capable of representing, it is possible to place some attribute types "on an even playing field." In these cases the different attribute types have shown similar performance, indicating our treatment is consistent. Both the additive and merge methods for combining attributes (as described in Chapter 4) appear to work well—the choice of method should depend on the domain characteristics. The additive method introduces biases that may or may not be appropriate to the domain, while the merge method appears to avoid introducing biases. We use the additive method in practice because it greatly simplifies the similarity function. Missing values may be treated within the framework by several methods, each reflecting different assumptions about the domain. None of these methods is consistently better over all domains, although domain characteristics have been identified to assist in the choice of a suitable method. Some knowledge of the domain can also assist in customising the similarity function. We have shown that customisation gives improved performance, particularly when training data is limited. Finally, the K* classifier using default similarity functions has been shown to perform well in comparison to other machine learning schemes with regard to classification accuracy. Memory usage, while not specifically addressed, is typical of instance-based learners. Classification speed depends on the similarity function employed—for functions with an analytic form, classification time increases linearly with the number of instances and the number of attributes. For more complex similarity functions, such as that used in Section 5.2.7.3, classification time can increase exponentially with the number of attributes.

Chapter 6

Conclusions

This thesis proposed a method for designing similarity functions that included natural solutions to several difficulties identified in previous systems. The chief difficulty was that of measuring similarity for vastly different types of objects. Another problem lies in tailoring the similarity function to the domain characteristics. There are other problems to be faced, such as dealing with objects that are missing information. The design framework should be able to encompass all these problems.

The design framework (called K*) proposed in Chapter 4, centres around treating object similarity as the probability of transforming from one object to another. An alternative view is that object dissimilarity can be regarded as the complexity of such transformations. Transformations between objects are taken to consist of sequences of smaller, basic transformations. The set of basic transformations chosen will depend on the type of objects undergoing comparison, but often a reasonable set can be determined intuitively. Each basic transformation is assigned a probability dependent on the domain, and so a probability can be calculated for any transformation sequence as a whole. There are usually many possible transformations between two objects. The decision to choose any one transformation path (such as that with the highest probability) would be arbitrary, and would introduce more problems (such as how to decide between multiple transformation paths that have the same probability). The K* approach considers all possible transformation paths (with their corresponding probability), rather than just the most likely path, combining the philosophies of both Occam's razor and Epicurus' principle of multiple explanations. Considering all possible transformation paths can be likened to the Solomonoff-Levin universal prior. This view of similarity as transformation probability is general and can be applied to any type of object by changing the set of basic transformations.

Several properties are typically held by traditional similarity functions, and these may also hold for similarity functions designed within the K* framework under certain conditions. Similarity functions are typically symmetric, although Tversky (1977) argues from a psychological perspective that human notions of similarity are often not symmetric. K* similarity functions are not necessarily symmetric, however it has been shown that the functions may be made symmetric by imposing conditions on the basic transformations. Similarly, the triangle inequality can be shown to hold given similar conditions. Whether these properties are desirable depends on the domain for which the similarity function is intended. A third property distinguishing K* dissimilarity functions from traditional dissimilarity functions is that the dissimilarity between an object and itself is typically non-zero. The dissimilarity an object has to itself varies in accordance with the intuitive notion of self-similarity described by Tversky.

In Chapter 4 several example similarity functions were developed. These examples illustrate potential approaches to dealing with common problems for similarity functions, and show that the framework satisfies the basic objectives identified in Chapter 1: the design framework handles different object types consistently, including multiple objects; the framework can handle missing information; the framework permits similarity from several sources to be combined coherently; the resulting functions are smooth with respect to small changes in the instances.

Chapter 5 built on the results of Chapter 4 to implement an instance-based learner employing K* similarity. Several specific problems were solved by following the K* design philosophy where possible. One area of difficulty was to find methods for setting the free parameters for the similarity function. Two methods were implemented, but both were limited by practical considerations. The K* classifier was evaluated along several dimensions to examine the behaviour of the similarity function in more detail. Experimentation supported the conclusion that different attribute types are treated consistently and that domain customisation of the similarity function can improve classifier performance. It was anticipated that the simple method for combining similarity from multiple attributes could introduce biases during classification (which may or may not be beneficial depending on the domain). Domain characteristics were identified that suggest which method for treating missing information is likely to be most appropriate. The K* classifier was found to perform well in relation other machine learning schemes.

6.1 Conclusions

A little domain information can go a long way. We have seen in the case of the K* learner that domain information may be used to choose an appropriate method for treating missing values and for combining similarity measures from multiple attributes. Domain information is also beneficial when choosing the basic method for measuring similarity for individual attributes. There are often many methods for modelling transformations between instances, and domain information is crucial for deciding which models are appropriate. Failure to correctly adapt the similarity function to the domain will lead to a loss in performance. However, in comparison with most general-purpose similarity functions, the advantages of domain customisation are primarily apparent when training data is limited (since this is when discontinuities in the general-purpose similarity function are more likely to cause incorrect classification). This finding is consistent with the findings of Fix and Hodges (1951) that any reasonable similarity function will eventually converge to the optimal error rate as training data increases. Domain customisation may not be worthwhile in domains where ample training data is available and storage requirements are not an issue.

These comments generalise to applications other than the K* learner. If sensitivity to the domain characteristics is paramount, the similarity function must capture as much domain information as possible. The design framework that this thesis proposed is the only method that I am aware of that permits such domain customisation of similarity functions.

Perhaps of more importance to machine learning as a field is the potential for instance-based learning to be extended to domains that require more complex instance representations than a fixed collection of numeric and symbolic attributes. For example, in Chapter 5 the K* classifier was applied to a domain with varying numbers of attributes, and shown to perform significantly better than a traditional machine learning algorithm. In these domains, decision tree and rule inducing schemes must either be custom developed (generally with no guiding principles to ensure robustness), or transform the instances to the learner's native representation (potentially losing information in the process). This thesis provides a framework for constructing instance-based learners for such domains.

6.2 Future Work

This thesis has uncovered several issues that deserve further investigation. The central objective of this thesis is the proposal of a design framework for similarity functions and the demonstration of its feasibility with a practical implementation. Issues not directly related to this objective have been relegated to future work, and these may be primarily categorised as improvements to the K* classifier, and developing further applications incorporating K* based similarity.

The K* classifier was not designed to be the best machine learning scheme—its purpose is to provide a test-bed for different similarity functions. It turns out that the K* classifier does perform very well; however, there are a number of improvements that could be made to yield even better performance. The first is to implement a method for automatically setting the blend parameters for multiple attributes simultaneously, perhaps using methods such as conjugate gradient descent. The current automatic method sets the blend parameter for each attribute independently so the relative importance of attributes cannot be determined. Other, cheaper heuristic methods for setting attribute weights (such as those described in Section 3.4) could provide easy performance improvements over the basic K* classifier.

The K* classifier implements a variety of basic measures for different attribute types. The experiments in Chapter 5 identified possible criteria for selecting one measure over another, and these could be incorporated into a heuristic method for automatically choosing appropriate metrics for attributes. Similar heuristics could choose an appropriate method for missing value handling.

A further enhancement to the K* classifier is to implement an editing mechanism to reduce storage requirements. For example, a method similar to that used by IB3 would involve each training instance having a record of how important it has been in classification. The importance updating procedure could be dependent on the relative probability of the instance's contribution towards a test instance, along with whether it supported the correct decision. The importance would be maintained as a "probability that this instance supports the correct classification."

Another area for future research is to carry out further work in geometric domains. This would involve developing more similarity functions for specific geometric applications—incorporating all these results together for a practical geometric application is a large task, and outside the scope of this thesis. However, we have shown that the framework can handle all the types of problems encountered in geometric domains.

Appendix A

The following similarity function development models a finite number of instance positions as having a reflective barrier placed at each edge position.

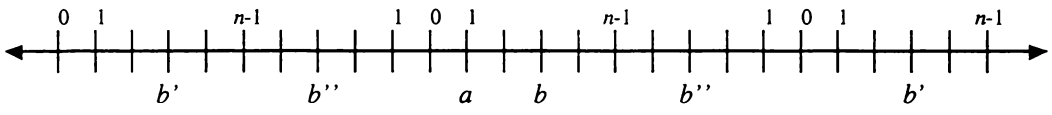

Figure A.1: Mapping finite positions using reflection at edge positions

Assume that at position 0 the possible transformations are to transform **right** to position 1 and **left** to position 1 (having reflected immediately upon starting the transformation). At position $n-1$ the possible transformations are **left** to position $n-2$ and **right** to position $n-2$ (having reflected immediately upon starting the transformation). The mapping that can be carried out on the integers (again assuming p(**left**)=p(**right**)) is shown in Figure A.1.

We want to calculate the probability of transforming from position a to position b $P_{\mathfrak{S}_n}{}^*(b \mid a)$. First assume $b \geq a$ and $0 < b < n-1$, then

$$\begin{aligned} P_n{}^*(b \mid a) = &\sum_{k \geq 0} P_\infty{}^*(2(n-1)k + (b-a)) + \sum_{k \geq 0} P_\infty{}^*(2(n-1)(k+1) - (b-a)) \\ &+ \sum_{k \geq 0} P_\infty{}^*(2(n-1)k + (b+a)) + \sum_{k \geq 0} P_\infty{}^*(2(n-1)(k+1) - (b+a)) \end{aligned}$$

where $P_{3_\infty}^*(i) = ce^{-mi}$ is the probability function over all integers as defined in Equation 4.13.

The first sum incorporates the transformation from a to b as well as all b's to the right. The second sum takes the transformations to all b's to the left. The third sum takes the transformations to all b''s to the left of a, and the fourth sum takes the transformations to all b''s to the right. Note that if b were at position 0 or position $n-1$, each transformation would be included in the sums twice. For example, the probabilities for programs finishing one position to the left of a (i.e. $b=0$) would be counted once by the first sum of the equation, and once by the third sum.

$$\begin{aligned}
P_n^*(b \mid a) &= \sum_{k\geq 0} ce^{-m(2(n-1)k+(b-a))} + \sum_{k\geq 0} ce^{-m(2(n-1)(k+1)-(b-a))} \\
&\quad + \sum_{k\geq 0} ce^{-m(2(n-1)k+(b+a))} + \sum_{k\geq 0} ce^{-m(2(n-1)(k+1)-(b+a))} \\
&= ce^{-m(b-a)}\sum_{k\geq 0} e^{-m2(n-1)k} + ce^{-m(2(n-1)-(b-a))}\sum_{k\geq 0} e^{-m2(n-1)k} \\
&\quad + ce^{-m(b+a)}\sum_{k\geq 0} e^{-m2(n-1)k} + ce^{-m(2(n-1)-(b+a))}\sum_{k\geq 0} e^{-m2(n-1)k} \\
&= c\frac{e^{-m(b-a)} + e^{-m(2(n-1)-(b-a))} + e^{-m(b+a)} + e^{-m(2(n-1)-(b+a))}}{1-e^{-m2(n-1)}} \\
&= c\frac{e^{m((n-1)-(b-a))} + e^{-m((n-1)-(b-a))} + e^{m((n-1)-(b+a))} + e^{-m((n-1)-(b+a))}}{e^{m(n-1)} - e^{-m(n-1)}} \\
&= c\frac{\cosh(m((n-1)-(b-a))) + \cosh(m((n-1)-(b+a)))}{\sinh(m(n-1))}.
\end{aligned}$$

Similarly, we find the following form will also hold for the case when $b < a$

$$P_{3_n}^*(b \mid a) = c\frac{\cosh(m((n-1)-|b-a|)) + \cosh(m((n-1)-(b+a)))}{\sinh(m(n-1))}.$$

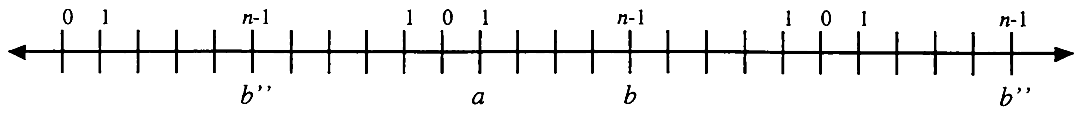

Figure A.2: Mapping when *b*=*n*-1

When $b=0$ or $b=n-1$, the first two terms of the previous equation are sufficient to include the transformations to all images of *b*. Figure A.2 depicts the situation when $b \geq a$ (i.e. $b=n-1$).

The expression encompassing the transformation probability from *a* to *b* and all its images is

$$\mathrm{P}_{\mathfrak{I}n}{}^{*}(b \mid a)=\sum_{k \geq 0} \mathrm{P}_{\mathfrak{I}\infty}{}^{*}(2(n-1)k+(b-a))+\sum_{k \geq 0} \mathrm{P}_{\mathfrak{I}\infty}{}^{*}(2(n-1)(k+1)-(b-a)).$$

Substituting in our equation for $\mathrm{P}_{\mathfrak{I}\infty}$,

$$\begin{aligned}
\mathrm{P}_{\mathfrak{I}n}{}^{*}(b \mid a) &= \sum_{k \geq 0} c e^{-m(2(n-1)k+(b-a))}+\sum_{k \geq 0} c e^{-m(2(n-1)(k+1)-(b-a))} \\
&= c e^{-m(b-a)} \sum_{k \geq 0} e^{-m2(n-1)k}+c e^{-m(2(n-1)-(b-a))} \sum_{k \geq 0} e^{-m2(n-1)k} \\
&= c \frac{e^{-m(b-a)}+e^{-m(2(n-1)-(b-a))}}{1-e^{-m2(n-1)}} \\
&= c \frac{e^{m((n-1)-(b-a))}+e^{-m((n-1)-(b-a))}}{e^{m(n-1)}-e^{-m(n-1)}} \\
&= c \frac{\cosh(m((n-1)-(b-a)))}{\sinh(m(n-1))}.
\end{aligned}$$

Again a similar development for the *b*<*a* (i.e. $b=0$) case allows us to use the following expression for both:

$$\mathrm{P}_{\mathfrak{I}n}{}^{*}(b \mid a)=c \frac{\cosh(m((n-1)-|b-a|))}{\sinh(m(n-1))}.$$

Appendix B

The following tables show one-step transformation probabilities obtained from the Brown corpus, as described in Section 4.3.8. Each column gives the letter transformed from, and each row gives the letter transformed to (the sum of each column is 1).

	A	B	C	D	E	F	G	H	I
A	0.131	0.052	0.039	0.033	0.085	0.035	0.040	0.016	0.125
B	0.010	0.036	0.018	0.012	0.007	0.014	0.014	0.025	0.007
C	0.015	0.037	0.057	0.018	0.011	0.028	0.030	0.037	0.018
D	0.016	0.031	0.023	0.074	0.044	0.056	0.051	0.044	0.011
E	0.132	0.059	0.046	0.138	0.178	0.113	0.112	0.054	0.116
F	0.010	0.021	0.021	0.033	0.021	0.036	0.026	0.023	0.009
G	0.010	0.018	0.019	0.025	0.018	0.022	0.025	0.021	0.008
H	0.011	0.090	0.065	0.061	0.023	0.053	0.058	0.133	0.019
I	0.113	0.035	0.041	0.020	0.067	0.029	0.029	0.025	0.142
J	0.000	0.004	0.003	0.001	0.000	0.002	0.002	0.002	0.001
K	0.004	0.007	0.005	0.009	0.005	0.007	0.007	0.011	0.004
L	0.030	0.066	0.038	0.041	0.030	0.039	0.035	0.050	0.021
M	0.009	0.034	0.026	0.025	0.014	0.025	0.023	0.042	0.011
N	0.055	0.037	0.041	0.072	0.070	0.057	0.054	0.047	0.058
O	0.093	0.059	0.036	0.042	0.075	0.064	0.048	0.018	0.094
P	0.015	0.029	0.023	0.014	0.015	0.020	0.019	0.025	0.011
Q	0.000	0.004	0.001	0.001	0.000	0.001	0.001	0.000	0.000
R	0.034	0.071	0.059	0.067	0.044	0.060	0.056	0.087	0.037
S	0.037	0.043	0.051	0.091	0.058	0.074	0.070	0.057	0.031
T	0.030	0.069	0.135	0.087	0.052	0.084	0.126	0.079	0.026
U	0.040	0.018	0.014	0.009	0.027	0.012	0.015	0.010	0.038
V	0.001	0.019	0.012	0.011	0.002	0.008	0.011	0.030	0.003
W	0.006	0.018	0.027	0.016	0.009	0.017	0.021	0.027	0.008
X	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.002	0.001
Y	0.007	0.006	0.008	0.036	0.023	0.028	0.023	0.012	0.006
Z	0.000	0.002	0.001	0.001	0.000	0.001	0.001	0.002	0.001
sp	0.190	0.133	0.191	0.063	0.121	0.116	0.101	0.120	0.197

Table B.1: One Step Probabilities from the Brown Corpus, A-I

	J	K	L	N	M	O	P	Q	R
A	0.014	0.043	0.058	0.028	0.063	0.098	0.058	0.033	0.044
B	0.038	0.017	0.025	0.020	0.008	0.012	0.022	0.063	0.018
C	0.049	0.022	0.028	0.032	0.018	0.015	0.036	0.036	0.030
D	0.033	0.056	0.039	0.039	0.040	0.022	0.027	0.036	0.043
E	0.026	0.099	0.090	0.068	0.124	0.124	0.090	0.010	0.089
F	0.031	0.024	0.022	0.023	0.019	0.020	0.023	0.028	0.023
G	0.020	0.022	0.016	0.018	0.015	0.012	0.019	0.022	0.018
H	0.083	0.094	0.066	0.089	0.036	0.013	0.067	0.024	0.077
I	0.030	0.041	0.036	0.030	0.060	0.090	0.041	0.004	0.044
J	0.009	0.001	0.002	0.003	0.001	0.002	0.002	0.019	0.002
K	0.005	0.011	0.007	0.008	0.005	0.003	0.005	0.001	0.008
L	0.043	0.044	0.069	0.044	0.035	0.022	0.042	0.037	0.045
M	0.041	0.030	0.027	0.041	0.015	0.013	0.030	0.036	0.029
N	0.037	0.056	0.060	0.043	0.135	0.037	0.033	0.023	0.062
O	0.104	0.037	0.040	0.039	0.039	0.154	0.060	0.279	0.039
P	0.030	0.017	0.021	0.024	0.009	0.016	0.033	0.028	0.018
Q	0.013	0.000	0.001	0.002	0.000	0.004	0.002	0.038	0.001
R	0.068	0.075	0.067	0.071	0.054	0.032	0.055	0.037	0.070
S	0.056	0.070	0.052	0.056	0.060	0.038	0.044	0.083	0.061
T	0.081	0.079	0.070	0.071	0.051	0.037	0.078	0.067	0.071
U	0.005	0.014	0.017	0.015	0.022	0.029	0.027	0.001	0.015
V	0.015	0.021	0.013	0.018	0.006	0.001	0.013	0.001	0.016
W	0.020	0.019	0.018	0.023	0.010	0.006	0.019	0.001	0.018
X	0.001	0.001	0.001	0.003	0.001	0.001	0.002	0.001	0.001
Y	0.008	0.022	0.014	0.014	0.019	0.009	0.009	0.001	0.017
Z	0.001	0.002	0.001	0.001	0.001	0.000	0.001	0.000	0.001
sp	0.140	0.082	0.139	0.175	0.154	0.190	0.162	0.091	0.138

Table B.2: One Step Probabilities from the Brown Corpus, J-R

	S	T	U	V	W	X	Y	Z	sp
A	0.046	0.026	0.119	0.009	0.024	0.060	0.034	0.026	0.071
B	0.010	0.012	0.010	0.030	0.015	0.008	0.006	0.026	0.009
C	0.024	0.045	0.016	0.039	0.045	0.021	0.014	0.035	0.028
D	0.055	0.037	0.014	0.043	0.033	0.027	0.083	0.039	0.012
E	0.110	0.070	0.124	0.030	0.058	0.076	0.164	0.050	0.070
F	0.026	0.021	0.010	0.020	0.021	0.015	0.037	0.019	0.013
G	0.021	0.027	0.010	0.021	0.022	0.011	0.026	0.019	0.009
H	0.047	0.047	0.020	0.165	0.079	0.042	0.038	0.126	0.030
I	0.035	0.020	0.101	0.021	0.031	0.046	0.024	0.062	0.066
J	0.001	0.001	0.000	0.002	0.002	0.001	0.001	0.002	0.001
K	0.007	0.006	0.003	0.014	0.007	0.004	0.008	0.010	0.002
L	0.033	0.031	0.026	0.056	0.039	0.028	0.033	0.053	0.027
M	0.022	0.020	0.014	0.047	0.031	0.037	0.021	0.040	0.021
N	0.065	0.039	0.057	0.044	0.037	0.048	0.078	0.043	0.051
O	0.044	0.030	0.081	0.007	0.023	0.046	0.042	0.017	0.067
P	0.014	0.017	0.020	0.026	0.021	0.024	0.011	0.024	0.015
Q	0.001	0.001	0.000	0.000	0.000	0.001	0.000	0.000	0.000
R	0.057	0.047	0.034	0.098	0.060	0.045	0.061	0.082	0.039
S	0.088	0.065	0.035	0.052	0.051	0.072	0.105	0.048	0.041
T	0.092	0.219	0.030	0.083	0.154	0.053	0.088	0.074	0.065
U	0.015	0.009	0.045	0.009	0.010	0.029	0.009	0.011	0.027
V	0.008	0.009	0.003	0.040	0.016	0.008	0.004	0.028	0.005
W	0.015	0.031	0.007	0.030	0.035	0.014	0.011	0.026	0.015
X	0.002	0.001	0.002	0.002	0.001	0.009	0.001	0.001	0.002
Y	0.028	0.016	0.006	0.007	0.011	0.013	0.046	0.010	0.004
Z	0.001	0.001	0.000	0.003	0.001	0.001	0.001	0.008	0.001
sp	0.135	0.151	0.211	0.104	0.173	0.265	0.053	0.119	0.311

Table B.3: One Step Probabilities from the Brown Corpus, S-space

The following tables show the P* transformation probabilities obtained from the Brown corpus, when the stop probability is 0.2.

	A	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
A	0.266	0.049	0.047	0.046	0.057	0.047	0.047	0.042	0.065
B	0.009	0.214	0.011	0.010	0.009	0.010	0.010	0.013	0.009
C	0.018	0.023	0.226	0.019	0.018	0.021	0.021	0.023	0.018
D	0.022	0.026	0.025	0.234	0.028	0.031	0.030	0.029	0.021
E	0.089	0.074	0.071	0.089	0.296	0.084	0.084	0.072	0.086
F	0.014	0.016	0.016	0.018	0.016	0.218	0.017	0.016	0.013
G	0.011	0.013	0.013	0.015	0.013	0.014	0.215	0.014	0.011
H	0.028	0.045	0.040	0.040	0.031	0.038	0.039	0.254	0.029
I	0.059	0.043	0.043	0.039	0.050	0.041	0.041	0.039	0.264
J	0.001	0.002	0.001	0.001	0.001	0.001	0.001	0.001	0.001
K	0.004	0.005	0.004	0.005	0.004	0.005	0.005	0.006	0.004
L	0.026	0.033	0.028	0.029	0.026	0.028	0.028	0.031	0.024
M	0.014	0.019	0.018	0.018	0.015	0.018	0.017	0.021	0.014
N	0.046	0.043	0.043	0.049	0.049	0.047	0.046	0.045	0.047
O	0.057	0.049	0.044	0.045	0.053	0.049	0.046	0.040	0.057
P	0.013	0.016	0.015	0.013	0.013	0.014	0.014	0.015	0.012
Q	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
R	0.037	0.045	0.042	0.044	0.039	0.042	0.042	0.048	0.037
S	0.039	0.041	0.043	0.050	0.044	0.047	0.047	0.044	0.038
T	0.050	0.061	0.073	0.064	0.056	0.063	0.071	0.064	0.050
U	0.022	0.017	0.016	0.015	0.019	0.016	0.016	0.015	0.021
V	0.005	0.009	0.008	0.007	0.005	0.007	0.007	0.011	0.005
W	0.010	0.013	0.015	0.013	0.011	0.013	0.014	0.015	0.011
X	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Y	0.010	0.010	0.010	0.016	0.013	0.014	0.013	0.011	0.010
Z	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
sp	0.146	0.133	0.144	0.118	0.131	0.129	0.126	0.129	0.148

Table B.4: P* Probabilities from the Brown Corpus, A-I

	J	K	L	M	N	O	P	Q	R
A	0.042	0.047	0.051	0.045	0.053	0.061	0.051	0.047	0.048
B	0.015	0.011	0.012	0.012	0.009	0.010	0.012	0.019	0.011
C	0.025	0.020	0.021	0.022	0.019	0.018	0.022	0.022	0.021
D	0.027	0.031	0.028	0.028	0.028	0.024	0.025	0.027	0.028
E	0.068	0.081	0.080	0.075	0.087	0.087	0.079	0.066	0.079
F	0.017	0.016	0.016	0.016	0.015	0.015	0.016	0.017	0.016
G	0.014	0.014	0.013	0.013	0.013	0.012	0.013	0.014	0.013
H	0.044	0.046	0.040	0.045	0.034	0.029	0.040	0.032	0.042
I	0.041	0.043	0.043	0.041	0.048	0.055	0.044	0.038	0.044
J	0.202	0.001	0.001	0.001	0.001	0.001	0.001	0.004	0.001
K	0.004	0.205	0.005	0.005	0.004	0.004	0.004	0.004	0.005
L	0.029	0.029	0.233	0.029	0.027	0.025	0.029	0.028	0.029
M	0.021	0.019	0.018	0.221	0.016	0.015	0.019	0.019	0.018
N	0.042	0.046	0.047	0.044	0.260	0.043	0.042	0.039	0.047
O	0.056	0.044	0.045	0.045	0.046	0.268	0.049	0.088	0.045
P	0.016	0.013	0.014	0.015	0.012	0.013	0.216	0.015	0.014
Q	0.003	0.001	0.001	0.001	0.001	0.001	0.001	0.207	0.001
R	0.044	0.045	0.044	0.044	0.041	0.036	0.041	0.038	0.244
S	0.044	0.046	0.043	0.044	0.044	0.040	0.041	0.048	0.045
T	0.063	0.063	0.061	0.061	0.056	0.052	0.062	0.059	0.061
U	0.014	0.016	0.017	0.016	0.018	0.020	0.018	0.014	0.016
V	0.008	0.009	0.008	0.009	0.006	0.005	0.008	0.005	0.008
W	0.014	0.013	0.013	0.014	0.011	0.010	0.013	0.010	0.013
X	0.001	0.001	0.001	0.002	0.001	0.001	0.001	0.001	0.001
Y	0.010	0.013	0.011	0.011	0.012	0.010	0.010	0.009	0.012
Z	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
sp	0.135	0.123	0.134	0.141	0.137	0.146	0.139	0.127	0.134

Table B.5: P* Probabilities from the Brown Corpus, J-R

	S	T	U	V	W	X	Y	Z	sp
A	0.048	0.044	0.064	0.040	0.044	0.052	0.046	0.044	0.055
B	0.010	0.010	0.009	0.014	0.011	0.009	0.009	0.013	0.009
C	0.020	0.025	0.018	0.024	0.024	0.020	0.018	0.023	0.021
D	0.031	0.028	0.022	0.029	0.027	0.025	0.036	0.028	0.022
E	0.084	0.075	0.087	0.067	0.073	0.077	0.094	0.072	0.076
F	0.017	0.016	0.014	0.016	0.016	0.015	0.019	0.016	0.014
G	0.014	0.015	0.012	0.014	0.014	0.012	0.015	0.014	0.011
H	0.037	0.037	0.030	0.060	0.043	0.035	0.035	0.052	0.033
I	0.043	0.039	0.057	0.038	0.041	0.046	0.040	0.046	0.050
J	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
K	0.005	0.004	0.004	0.006	0.005	0.004	0.005	0.005	0.004
L	0.027	0.027	0.025	0.032	0.029	0.026	0.027	0.031	0.026
M	0.017	0.017	0.015	0.022	0.019	0.019	0.017	0.021	0.017
N	0.048	0.043	0.047	0.044	0.043	0.045	0.051	0.044	0.045
O	0.046	0.043	0.055	0.037	0.041	0.047	0.045	0.040	0.051
P	0.013	0.014	0.014	0.015	0.014	0.014	0.012	0.015	0.013
Q	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
R	0.042	0.040	0.037	0.050	0.043	0.039	0.043	0.047	0.038
S	0.249	0.046	0.039	0.044	0.043	0.046	0.053	0.043	0.040
T	0.065	0.289	0.051	0.065	0.077	0.057	0.064	0.062	0.058
U	0.016	0.015	0.222	0.015	0.015	0.019	0.015	0.015	0.019
V	0.007	0.007	0.005	0.213	0.008	0.006	0.006	0.011	0.006
W	0.012	0.016	0.011	0.016	0.216	0.012	0.012	0.015	0.012
X	0.001	0.001	0.001	0.001	0.001	0.202	0.001	0.001	0.001
Y	0.014	0.012	0.010	0.010	0.011	0.011	0.217	0.011	0.009
Z	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.202	0.001
sp	0.133	0.136	0.150	0.126	0.140	0.158	0.116	0.130	0.368

Table B.6: P* Probabilities from the Brown Corpus, S-space

Bibliography

Aha, D.W. (1990), *A Study of Instance-Based Algorithms for Supervised Learning Tasks*, PhD Thesis, Department of Information and Computer Science, University of California, Irvine, Technical Report 90-42.

Aha, D.W. (1992), Tolerating Noise, Irrelevant and Novel Attributes in Instance-Based Learning Algorithms, *International Journal of Man Machine Studies*, vol. 36, pp. 267–287.

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

Allison, L. and C.N. Yee (1990), Minimum Message Length Encoding and the Comparison of Macromolecules, *Bulletin of Mathematical Biology*, vol. 52, no. 3, pp. 431–453.

Almuallim, H., Y. Akiba and S. Kaneda (1995), On Handling Tree-Structured Attributes in Decision Tree Learning, in *Proceedings of the Twelfth International Conference on Machine Learning*, pp.12–20, Tahoe City, CA: Morgan Kaufmann

Bankert, R.L. and D.W. Aha (1995), Automated Identification of Cloud Patterns in Satellite Imagery, to appear in *Proceedings of the*

Fourteenth Conference on Weather Analysis and Forecasting, Dallas, TX: American Meteorological Society.

Baxter, R.A. and J.J. Oliver (1994), *MDL and MML: Similarities and Differences (Introduction to Minimum Encoding Inference—Part III)*, Technical Report 207, Department of Computer Science, Monash University, Australia.

Berliner, H.J. (1980), Backgammon Computer Program Beats World Champion, *Artificial Intelligence*, vol. 14, pp. 205–220.

Biberman, Y. (1994), A Context Similarity Measure, in *Proceedings of the Seventh European Conference on Machine Learning*, pp. 49–63. Catania, Italy: Springer-Verlag.

Bradshaw, G.L. (1986), Learning by Disjunctive Spanning, in T.M. Mitchell, J.G. Carbonell and R.S. Michalski (Eds), *Machine Learning: A guide to current research*, Boston, MA: Kluwer Academic Publishers.

Chaitin, G.J. (1969), On the Length of Programs for Computing Finite Binary Sequences: Statistical Considerations, *Journal of the ACM*, vol. 16, pp. 145–159.

Cleary, J., G. Holmes, S.J. Cunningham and I.H. Witten (1996a), MetaData for Database Mining, *Proceedings of the IEEE Metadata Conference*, Silver Spring, MD, April 16–18.

Cleary, J., S. Legg and I.H. Witten (1996b), An MDL Estimate of the Significance of Rules, *Proceedings of the Information, Statistics and Induction in Science Conference*, pp. 43–53, Melbourne, Australia, World Scientific.

Cost, S. and S. Salzberg (1993), A Weighted Nearest Neighbor Algorithm for Learning with Symbolic Features, *Machine Learning*, vol. 10, pp. 57–78.

Cover, T. and P. Hart (1967), Nearest Neighbor Pattern Classification, *IEEE Transactions on Information Theory*, vol. 13, pp. 21–27.

Dasarathy, B.V. (Ed) (1990), *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press.

Dempster, A.P., N.M. Laird, and D.B. Rubin (1977), Maximum Likelihood from Incomplete Data via the EM Algorithm, *Journal of the Royal Statistical Society*, vol. 39. no. 1, pp.1-38.

Devroye, L.P. (1978), A Universal K-Nearest Neighbor Procedure in Discrimination, in *Proceedings of the Conference on Pattern Recognition and Image Processing*, IEEE Computer Society Press, Los Alamitos, Calif., pp. 142–147.

Dixon, J.K. (1979), Pattern Recognition with Partly Missing Data, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no.10, pp. 617–621.

Donovan, B.J., A.M.E. Howie, N.C. Schroeder, A.R. Wallace, and P.E.C Read (1992), Comparative characteristics of nests of Vespula germanica (F.) and Vespula vulgaris (L.) (Hymenoptera: Vespinae) from Christchurch City, New Zealand, *New Zealand Journal of Zoology*, vol. 19, pp. 61–71.

Dudani, S.A. (1976), The Distance-Weighted Nearest Neighbor Rule, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 6, no. 4, pp. 325–327.

Fayyad, U.M. and K.B. Irani (1992), On the Handling of Continuous-Valued Attributes in Decision Tree Generation, *Machine Learning*, vol. 8, pp. 87–102.

Fayyad, U.M. and K.B. Irani (1993), Multi-Interval Discretization of Continuous-Valued Attributes for Classification Learning, in *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, pp. 1022–1027, Morgan-Kaufmann.

Fix, E. and J.L. Hodges (1951), Discriminatory Analysis: Nonparametric Discrimination: Consistency Properties, from *Project 21-49-004, Report Number 4*, USAF School of Aviation Medicine, Randolph Field, Texas, pp. 261–279.

Fix, E. and J.L. Hodges (1952), Discriminatory Analysis: Nonparametric Discrimination: Small Sample Performance, from *Project 21-49-004, Report Number 11*, USAF School of Aviation Medicine, Randolph Field, Texas, pp. 280–322.

Fogarty, T.C. (1992), First Nearest Neighbor Classification on Frey and Slate's Letter Recognition Problem, *Machine Learning*, vol. 9, pp. 387–388.

Francis, W.N and H. Kucera (1982), *Frequency Analysis of English Usage: Lexicon and Grammar*, Houghton Mifflin, Boston.

Fukunaga, K. and T.E. Flick (1984), An Optimal Global Nearest Neighbor Metric, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 6, no. 3, pp. 314–318.

Gates, G.W. (1972), The Reduce Nearest Neighbor Rule, *IEEE Transactions on Information Theory*, vol. 13, no. 3, pp. 431–433.

Graham, R.L., D.E. Knuth, and O. Patashnik (1994), *Concrete Mathematics*, Addison Wesley, New York.

Hart, P.E. (1968), The Condensed Nearest Neighbor Rule, *IEEE Transactions on Information Theory*, vol. 14, no. 3, pp. 515–516.

Hastie, T. and R. Tibshirani (1995), Discriminant Adaptive Nearest Neighbor Classification, in *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pp. 142–149, AAAI Press.

Holmes, G., A. Donkin, and I.H. Witten (1994), WEKA: A Machine Learning Workbench, in *Proceedings of the Second Australia and New Zealand Conference on Intelligent Information Systems*, Brisbane, Australia.

Holte, R.C. (1993), Very Simple Classification Rules Perform Well on Most Commonly Used Datasets, *Machine Learning*, vol. 11, pp. 63–91.

Inglis, S. and I. Witten (1995), Document Zone Classification, in *Proceedings of DICTA-95*, pp. 631–636, University of Queensland, Australia.

Jabbour, K., J.F.V. Riveros, D. Landsbergen, and W. Meyer (1988), ALFA: Automated Load Forecasting Assistant, *IEEE Transactions on Power Systems*, vol. 3, no. 3, pp. 908–913.

Jabbour, K. and W. Meyer (1989), GAuLF: Gas Automated Load Forecaster, in *Proceedings of the 32nd Midwest Symposium on Circuits and Systems*, Champaign, pp. 20–23.

Jones, E.K. and A. Roydhouse (1993), *Intelligent Retrieval of Historical Meteorological Data*, Technical Report CS-TR-93/8, Victoria University of Wellington.

Keller, J.M, M.R. Gray and J.A. Givens (1985), A Fuzzy *k*-Nearest Neighbor Algorithm, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 15, no. 4, pp. 580–585.

Kolmogorov, A.N. (1965), Three Approaches to the Quantitative Definition of Information, *Problems of Information Transmission*, vol. 1, no. 1, pp. 1–7.

Lee, C. (1994), An Instance-Based Learning Method for Databases: An Information Theoretic Approach, in *Proceedings of the Seventh European Conference on Machine Learning*, pp. 387–390. Catania, Italy: Springer-Verlag.

Legg, S. (1995), *Minimum Information Estimation of Linear Regression Models*, 420 Report, Department of Computer Science, University of Waikato, New Zealand.

Levin, L.A. (1974), Laws of Information Conservation (Non-growth) and Aspects of the Foundation of Probability Theory, *Problems of Information Transmission*, vol. 10, no. 3, pp. 206–210.

Li, M. and P. Vitanyi (1992), Inductive Reasoning and Kolmogorov Complexity, *Journal of Computer and System Sciences*, vol. 44, pp 343–384.

Li, M. and P. Vitanyi (1993), *Introduction to Kolmogorov Complexity and its Applications*, Springer-Verlag, New York.

Lowe, D.G. (1995), Similarity Metric Learning for a Variable-Kernel Classifier, *Neural Computation*, vol. 7, no. 1, pp. 72–85.

Macleod, J.E., A. Luk and D.M. Titterington (1987), A Re-Examination of the Distance-Weighted *k*-Nearest Neighbor Classification Rule, *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 17, no. 4, pp. 689–696.

Mehta, M., J. Rissanen and R. Agrawal (1995), MDL-based Decision Tree Pruning, in *Proceedings of the First International Conference on Knowledge Discovery and Data Mining*, pp. 216–221, AAAI Press.

Oates, W.J. (Ed) (1957), *The Stoic and Epicurean Philosophers: The Complete Extant Writings of Epicurus, Epictetus, Lucretius, Marcus Aurelius*, Random House, New York.

Oliver, J.J. and R.A. Baxter (1994), *MML and Bayesianism: Similarities and Differences (Introduction to Minimum Encoding Inference—Part II*, Technical Report 206, Department of Computer Science, Monash University, Australia.

Oliver, J.J. and D. Hand (1994), *Introduction to Minimum Encoding Inference—Part I*, Technical Report 205, Department of Computer Science, Monash University, Australia.

Porter, B.W., R Bareiss and R.C. Holte (1990), *Concept Learning and Heuristic Classification in Weak-Theory Domains*, Artificial Intelligence, vol. 45, pp. 229–263.

Quinlan, J.R. and R.L. Rivest (1989), Inferring Decision Trees Using the Minimum Description Length Principle, *Information and Computation*, vol. 80, pp. 227–248.

Rachlin, J., S. Kasif, S. Salzberg and D.W. Aha (1994), Towards a Better Understanding of Memory-Based Reasoning Systems, in *Proceedings* *of the Eleventh International Conference on Machine Learning*, pp. 242–250. Morgan-Kaufmann.

Riesbeck, C. and R. Schank (1989), *Inside Case-Based Reasoning*, Hillsdale, N.J., L. Erlbaum.

Rissanen, J. (1989), *Stochastic Complexity in Statistical Inquiry*, World Scientific.

Rosch, E. (1975), Cognitive Reference Points, *Cognitive Psychology*, vol. 7, pp. 532–547.

Salzberg, S. (1991), A Nearest Hyperrectangle Learning Method, *Machine Learning*, vol. 6, pp. 251–276.

Sejnowski, T.J and C.R. Rosenberg (1987), Parallel Networks that Learn to Pronounce English Text, *Complex Systems*, vol. 1, pp. 145–168.

Short, R.D. and K. Fukunaga (1981), The Optimal Distance Measure for Nearest Neighbor Classification, *IEEE Transactions on Information Theory*, vol. 27, no. 5, pp. 622–627.

Solomonoff, R.J. (1964), A Formal Theory of Inductive Inference, Part 1 and Part 2, *Information and Control*, vol. 7, pp. 1–22, 224–254.

Spitzer, F. (1975), *Principles of Random Walk*, Springer-Verlag, New York.

Stanfill, C. and D. Waltz (1986), Toward Memory-Based Reasoning, *Communications of the ACM*, vol. 29, no. 12, pp. 1213–1228.

Thomas, G.B. and R.L. Finney (1988), Calculus and Analytical Geometry, 7th edition, Addison-Wesley, Reading, Massachusetts.

Ting, K.M. (1995), *Common Issues in Instance-Based and Naive Bayesian Classifiers*, PhD Thesis, Basser Department of Computer Science, University of Sydney.

Tversky, A. (1977), Features of Similarity, *Psychological Review*, vol. 84, no. 4, pp. 327–352.

Wilson, D.R. and T.R. Martinez (1997), Improved Heterogeneous Distance Functions, *Journal of Artificial Intelligence Research*, vol. 6, pp. 1–34.

Valiant, L.G. (1984), A Theory of the Learnable, *Communications of the ACM*, vol. 27, no. 11, pp. 1134–1143.

Wax, N. (Ed) (1954), *Selected Papers on Noise and Stochastic Processes*, Dover Publications, New York.

Wettschereck, D. (1994), A Hybrid Nearest-Neighbor and Nearest Hyperrectangle Algorithm, in *Proceedings of the Seventh European Conference on Machine Learning*, pp. 323–335. Catania, Italy: Springer-Verlag.

Wettschereck, D. and D.W. Aha (1995), Weighting Features, in *Proceedings of the First International Conference on Case-Based Reasoning*, Lisbon, Portugal: Springer-Verlag.

Wettschereck, D. and T.G. Dietterich (1995), An Experimental Comparison of the Nearest-Neighbor and Nearest-Hyperrectangle Algorithms, *Machine Learning*, vol. 19, pp. 5–27.

Witten, I.H. and T.C. Bell (1991), The zero-frequency problem: estimating the probabilities of novel events in adaptive text compression, *IEEE Transactions on Information Theory*, vol. 37, no. 4, pp. 1085–94.

Yee, C.N. and L. Allison (1993), Reconstruction of Strings Past, *Computer Applications in the Biosciences*, vol. 9, no. 1, pp. 1–7.

Zhang, J. (1992), Selecting Typical Instances in Instance-Based Learning, in *Proceedings of the Ninth International Workshop on Machine Learning*, pp. 470–479. Morgan Kaufmann.