

1 FEATURE WEIGHTING FOR LAZY LEARNING ALGORITHMS

David W. Aha

Navy Center for Applied Research in Artificial Intelligence
Naval Research Laboratory, Code 5510
4555 Overlook Ave, SW
Washington, D.C. 20375-5337
www.aic.nrl.navy.mil/~aha

Abstract:

Learning algorithms differ in the degree to which they process their inputs prior to their use in performance tasks. Many algorithms eagerly compile input samples and use only the compilations to make decisions. Others are lazy: they perform less precompilation and use the input samples to guide decision making. The performance of many lazy learners significantly degrades when samples are defined by features containing little or misleading information. Distinguishing feature relevance is a critical issue for these algorithms, and many solutions have been developed that assign weights to features. This chapter introduces a categorization framework for feature weighting approaches used in lazy similarity learners and briefly surveys some examples in each category.

1.1 INTRODUCTION

Lazy learning algorithms are machine learning algorithms (Mitchell, 1997) that are welcome members of procrastinators anonymous. Purely lazy learners typically display the following characteristics (Aha, 1997):

1. *Defer*: They delay the processing of their inputs until they receive requests for information; they simply store their inputs for future use.
2. *Demand-Driven*: They reply to information queries by combining information from their stored (e.g., training) samples.
3. *Discard*: They delete the constructed query and any intermediate results.

In contrast, *eager* algorithms greedily replace their inputs with an abstraction (e.g., a rule set, decision tree, or neural network) and use it to process queries.

Lazy learners have been designed to induce decision trees (e.g., (Smyth and Cunningham, 1995; Friedman et al., 1996)), rule sets (e.g., (Lachiche and Marquis, 1998)), Bayes classifiers (Kontkanen et al., 1998)), and improve speedup learning algorithms (e.g., (Borrajo and Veloso, 1997)). However, the most frequently studied group of lazy learners are those that use similarity functions to answer queries. These include k -nearest neighbor classifiers and algorithms identified by names satisfying the following grammar:

$$\{\text{case,exemplar,instance,memory}\}\text{-based } \{\text{learning,reasoning}\}.$$

These *lazy similarity algorithms* are the focus of this chapter.

Eager learning algorithms assume their learning bias is appropriate for the performance task. When this assumption is correct, this can yield performance benefits (e.g., increased query response speed). However, there is a risk that this assumption is wrong, and that information lost during eager abstraction is crucial for generating accurate responses to queries. Thus, a key advantage of lazy algorithms is that they can respond to unanticipated queries in ways not available to all eager learners (Barsalou, 1983).

Lazy algorithms have three other computational advantages. First, they have small training costs (e.g., store a sample and update some indices), although this is frequently balanced by higher costs for generating predictions unless a fast indexing scheme is implemented. Small training costs make lazy algorithms particularly well suited for incremental learning tasks, where a data stream continually updates the set of input samples. Second, lazy problem solvers provide efficiency gains through solution reuse; they can store and adapt solutions for subsequent problems, which can greatly reduce problem solving effort. Finally, lazy algorithms can generate precedent explanations, which are preferable to abstract explanations for many tasks.

Only purists and teachers remain interested in purely lazy, standalone learners. Most research and practice with lazy learners involves some form of *caching*, which can be used to “fine tune” a lazy learner (e.g., by storing information on prediction quality) so as to improve its performance. In this chapter, we focus on methods for caching binary or continuous feature weights in the context of feature-value representations for samples. Feature weights are used to denote the relevance of features in similarity computations, allowing similarity functions to emphasize features according to their relevance, assuming accurate weight settings. When some features are irrelevant to the prediction task, learning accurate weight settings can greatly increase learning rates in lazy learners; both theoretical (Langley and Iba, 1993) and empirical analyses (Wettschereck, 1994) suggest that the complexity of lazy learning is exponential in the number of irrelevant features.

This chapter introduces a categorization framework for weighting features in lazy similarity learners, and summarizes some example algorithms. Research on this topic spans several decades and disciplines (e.g., data mining, cognitive psychology, statistics). We focus primarily on contributions to machine learning and case-based reasoning. Our framework complements rather than repeats the

framework presented in (Wettschereck et al., 1997), which is organized around a set of dimensions rather than a structured hierarchy.

1.2 FEATURE WEIGHTING IN A LAZY LEARNING ALGORITHM

This section defines the lazy similarity learning algorithm named k -nearest neighbor (k -NN) (Fix and Hodges, 1951) to exemplify the use of feature weights. We chose k -NN because it is well known, and many other lazy similarity learners can be described according to how they differ from k -NN. Some of k -NN's less relevant details are omitted. Although this section focuses on classification, we will later discuss k -NN variants for other types of tasks (e.g., retrieval, regression).

A *classifier* inputs a query \mathbf{q} and outputs a class prediction. The query and each *training sample* $\mathbf{x} = \{x_1, x_2, \dots, x_{|\mathbf{F}|}\}$ are points in a multidimensional space defined by a feature set \mathbf{F} . The class (or cluster) of a sample is $x_c \in \mathbf{J}$. We assume each feature is either continuous or discrete, although only for this presentation.

A classifier's performance objective is to minimize *expected loss*, or *misclassification risk* R_j , for each class $c_j \in \mathbf{J}$:

$$R_j = \sum_{c'_j \in \mathbf{J}} L_{c_j c'_j} p(c'_j | \mathbf{q}) \quad (1.1)$$

where $L_{c_j c'_j}$ is the loss (cost) associated with mistakenly classifying a sample of class c_j as in class c'_j ($j \neq j'$), and $p(c'_j | \mathbf{q})$ is the probability of classifying \mathbf{q} in class c'_j . k -NN generally assumes that all misclassifications have equal cost:

$$L_{c_j c'_j} = \begin{cases} 0 & j = j' \\ 1 & j \neq j' \end{cases} \quad (1.2)$$

Since k -NN is not given \mathbf{q} 's class, it instead outputs the most probable class:

$$k\text{-NN}(\mathbf{q}) = \arg \max_{c_j \in \mathbf{J}} p(c_j | \mathbf{q}) \quad (1.3)$$

k -NN differs from other classifiers in how it defines posterior class probabilities:

$$p(c_j | \mathbf{q}) = \frac{\sum_{\mathbf{x} \in \mathbf{K}_{\mathbf{q}}} \begin{cases} K(d(\mathbf{x}, \mathbf{q})) & x_c = c_j \\ 0 & \text{otherwise} \end{cases}}{\sum_{\mathbf{x} \in \mathbf{K}_{\mathbf{q}}} K(d(\mathbf{x}, \mathbf{q}))} \quad (1.4)$$

where $K()$ is a *kernel* function, here defined as

$$K(d(\mathbf{x}, \mathbf{q})) = \frac{1}{d(\mathbf{x}, \mathbf{q})}, \quad (1.5)$$

and where $\mathbf{K}_{\mathbf{q}}$ is the set of \mathbf{q} 's nearest neighbors among a set \mathbf{X} of (previously supplied) *training* samples as determined by the distance function $d()$. That is,

k -NN computes the distance $d(\mathbf{x}, \mathbf{q})$ of \mathbf{q} to each $\mathbf{x} \in \mathbf{X}$ using:

$$d(\mathbf{x}, \mathbf{q}) = \left(\sum_{f \in \mathbf{F}} w(f) \cdot \delta(x_f, q_f)^r \right)^{\frac{1}{r}} \quad (1.6)$$

where k -NN defines $r = 2$ (i.e., Euclidean distance), function $\delta()$ defines how values of a given feature differ:

$$\delta(x_f, q_f) = \begin{cases} |x_f - q_f| & f \text{ is continuous} \\ 0 & f \text{ is discrete and } x_f = q_f \\ 1 & f \text{ is discrete and } x_f \neq q_f \end{cases} \quad (1.7)$$

and $w(f)$ defines the *feature weighting* function in k -NN, which is

$$w(f) = w_f \quad (1.8)$$

$$\forall f \in \mathbf{F} w_f = s \quad (1.9)$$

for some scalar constant s .

Several variants of k -NN exist (Dasarathy, 1991; Aha, 1997). We will focus on alternatives to Equations 1.8 and 1.9 that assign unequal weights to features.

1.3 A CATEGORIZATION OF FEATURE WEIGHTING IN LAZY LEARNERS

1.3.1 Framework

Figure 1.1 displays our categorization framework for feature weighting methods in lazy similarity learners. Highlighting whether an oracle is available reinforces a lesson: exploit expert domain knowledge when it is available. Although this does not guarantee performance improvements, available experts should always be consulted. Furthermore, knowledge-poor approaches will sometimes fail without exploiting domain-specific knowledge (Cain et al., 1991).

Our emphasis is on automated, knowledge-poor approaches to feature weighting and selection (i.e., binary weights). This task can be viewed as a search process through a space of weight settings for a given sample space representation. Figure 1.2 summarizes this view; it corresponds to the *inferred* category shown in Figure 1.1, where less domain knowledge is assumed to be available. This framework emphasizes that search is guided by an evaluation function, which can be categorized by its task-dependent performance measures (e.g., classification accuracy, retrieval efficiency, problem solution quality, or a cost function) and the evaluation method.

Evaluation methods can be dichotomized by whether they use feedback from the performance task. Methods that do not use feedback are called *filters*, whereas methods that use the performance algorithm itself as the evaluation function are *wrappers* (John et al., 1994). Because most of the algorithms mentioned here use either a filter or wrapper model to set feature weights, we tend to ignore feedback methods that are not wrappers.

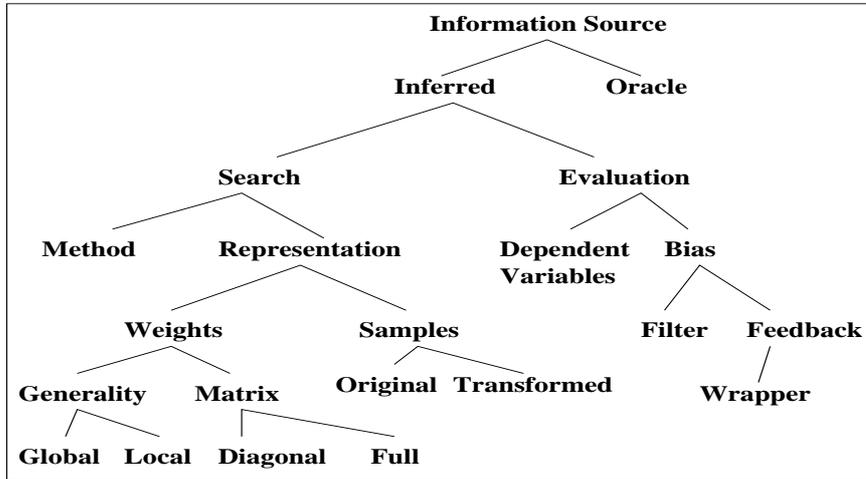


Figure 1.1 A Categorization Framework of Feature Weighting Methods for use in Lazy Similarity Learning Algorithms

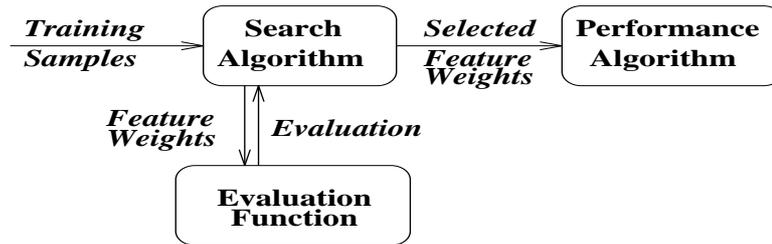


Figure 1.2 Automated Feature Weighting as a Search Task

1.3.2 Inferring Feature Weights

Automated feature weighting methods can be categorized by their search algorithm and evaluation function (Figure 1.1). We focus primarily on search functions, often using the evaluation function category to group algorithms with similar search functions.

Searching for Feature Weight Settings.

Search Algorithms

Some filter methods for weighting features do not perform search, including those that assign weights using simple probabilities (Creecy et al., 1992), mutual information gain (Daelemans and van den Bosch, 1992; Wettschereck et al., 1997), or value-specific differences in concept distributions (Stanfill and Waltz, 1986). However, the vast majority of feature weighting methods do perform search. This search is rarely exhaustive because the size of the weight space is

exponential in the number of features (i.e., $O(y^{|\mathbf{F}|})$), where y is the number of different weight settings a feature can have and \mathbf{F} is the feature set).

Search Algorithms used with Filter Models: Many filter methods for feature weighting use a hill-climbing search. For example, some induce a decision tree, where the features selected for use in similarity computations are those that remain in the tree after pruning. Cardie [(Cardie, 1993)] reported that this approach significantly increased classification accuracies for three natural language learning tasks, while Kibler and Aha [(Kibler and Aha, 1987)] reported more mixed results on two medical classification tasks.

Trott and Leng [(Trott and Leng, 1997; Leng and Trott, 1997)] instead used an *integer programming* (IP) approach to set feature weights for an interactive sample retrieval task, where users iteratively supply query \mathbf{q} 's feature values. In this task, the ordering of features in a stored sample influences the query input process. Leng and Trott's algorithm imposes a fixed ratio, for all samples, between the weight of the last feature and the sum of the weights of all other features in a sample. This ratio can be used to generate a set of linear equations, one per training sample, which can be solved using IP to generate feature weights. For highly engineered applications, this method could be used to remove the need to manually set feature weights for interactive k -NN retrieval algorithms. However, its evaluation has not yet been published.

Search Algorithms used with Wrapper Models: Many wrapper search methods use a variant of hill climbing. This includes the online weighting method in EACH (Salzberg, 1991), which updates feature weights by a fixed amount Δ after classifying each training sample. Correct classifications cause the weights of matching (mismatching) features to be incremented (decremented). Incorrect classifications cause the weights of mismatching (matching) features to be incremented (decremented). Thus, weights are continuously updated so as to locally maximize classification accuracy. By selecting good values for Δ , EACH (with $k = 1$) consistently improved classification performance in comparison with using Equation 1.9, although performance differences are also due to EACH's lazy replacement of input samples with hyperrectangles.

Salzberg's algorithm influenced the design of IB4 (Aha, 1992), which uses another online, hill-climbing wrapper to set feature weights:

$$w(f) = \max\left(\frac{\text{CumulativeWeight}_f}{\text{WeightNormalizer}_f} - 0.5, 0\right), \quad (1.10)$$

where $\text{CumulativeWeight}_f$ is assumed to converge to half of $\text{WeightNormalizer}_f$ for irrelevant features. IB4 was designed for skewed concept distributions and to avoid the need for a fixed weight adjustment parameter. Let $\Lambda(\mathbf{x}, \mathbf{y}) = \max(p(x_c), p(y_c))$ (i.e., the higher a priori probability of \mathbf{x} and \mathbf{y} 's classes). Then $\text{CumulativeWeight}_f$ is incremented by

$$\text{CumulativeWeight}_f \pm \begin{cases} 1 - \delta(x_f, y_f) \cdot (1 - \Lambda(\mathbf{x}, \mathbf{y})) & \text{if } (x_c = y_c) \\ \delta(x_f, y_f) \cdot (1 - \Lambda(\mathbf{x}, \mathbf{y})) & \text{Otherwise} \end{cases} \quad (1.11)$$

and `WeightNormalizer` is always incremented by $(1 - \Lambda(\mathbf{x}, \mathbf{y}))$. With $k = 1$, IB4 recorded good results vs. using Equation 1.9 on some tasks involving irrelevant features.

IB4 assumes a uniform distribution for irrelevant feature values. Kira and Rendell [(Kira and Rendell, 1992)] introduced the binary weighting algorithm `RELIEF` to remove this constraint. It iterates through a weight-updating procedure m times that (1) selects a random training sample \mathbf{x} , (2) locates \mathbf{x} 's most similar positive (\mathbf{p}) and negative (\mathbf{n}) training samples, and (3) updates each feature's weight using

$$w(f) = w(f) - \delta(x_f, p_f) + \delta(x_f, n_f) \quad (1.12)$$

During classification, weights that exceed a user-specified *relevance* threshold are mapped to 1, while those that do not are mapped to 0. Kira and Rendell reported good results for `RELIEF` on parity tasks. Kononenko [(Kononenko, 1994)] extended `RELIEF` for $k > 1$ and to work on multiclass tasks. Robnik-Šikonja and Kononenko extended it to work on regression tasks. Wettschereck et al. [(Wettschereck et al., 1997)] did not map `RELIEF`'s continuous weights to binary values. Domingos [(Domingos, 1997)] introduced `RC`, an algorithm reminiscent of `RELIEF`. `RC` hill-climbs only if feature selection increases predictive accuracy, as guided by leave-one-out cross validation error (`LOOCE`) on the training set. In contrast, `RELIEF` climbs after classifying each sample.

Conjugate gradient, combined with `LOOCE`, has also been used to guide search for feature weights in lazy similarity learners. Lowe [(Lowe, 1995)] used this approach in the variable kernel similarity metric (`VSM`). It uses knowledge of the function's gradient to direct search, which can substantially increase learning rates when the target function is reasonably smooth. Error derivatives are used to guide the conjugate gradient procedure for each feature weight. The `VSM` performed as well as or better than several other algorithms on two data sets yet required far less training time than some other algorithms. Wettschereck [(Wettschereck, 1994)] investigated a simplification of the `VSM` named k -`NNVSM`. After optimizing k , it uses conjugate gradient to optimize feature weights so as to minimize `LOOCE` training error. k -`NNVSM` can learn good feature weights in a variety of domains although its relatively large number of design decisions can heavily influence its performance. Wilke and Bergmann [(Wilke and Bergmann, 1996)] also used conjugate gradient with `LOOCE` error to search for feature weights. They tested this approach in two algorithms, `KNNacc` and `KNNcost`, and reported substantial decision cost decreases for a credit scoring task.

Several parallel hill climbing algorithms for finding feature weight settings have been investigated. Kelly and Davis [(Kelly and Davis, 1991)] reported that a genetic algorithm (`GA`) search outperformed Equation 1.9 on three classification tasks. Wilson and Martinez [(Wilson and Martinez, 1996)] found that their `GA` outperformed unweighted k -`NN` on only half of 16 tasks, but significant accuracy increases often occurred for tasks with irrelevant features. Chang and Lippman's [(Chang and Lippman, 1991)] `GA` performed feature

selection rather than feature weighting. On one task, they compared their algorithm vs. sequential search algorithms (i.e., forward sequential search (FSS) and backwards sequential elimination (BSE)). Their GA approach reduced error rates using fewer than half the features selected by the other algorithms. Skalak (Skalak, 1994), using mutation operations on a single bit string, reported increased classification accuracies on three of four tasks while selecting approximately half the features per task. His algorithm simultaneously deleted at least 96% of the samples per task.

Aha and Bankert (Aha and Bankert, 1996) explored beam search variants of hill-climbing in sequential feature selection algorithms when $k = 1$. Their algorithm, which stochastically selects feature subsets to evaluate, significantly increased accuracies for some cloud classification data sets. Extensions of this research lead to using a distributed output representation with error-correcting capability, where feature selection is performed independently for each output bit (Ricci and Aha, 1998). This computationally expensive approach significantly increased accuracy on some data sets without significant decreases on any others.

Maron and Moore [(Maron and Moore, 1997)] described an innovative parallel feature selection approach named *schemata racing*. They noted that FSS and BSE are heavily biased towards selecting few or many features, respectively, and are slow when these biases are not met. In contrast, schemata racing hill climbs using *races*, simultaneously run for all features, to perform feature selection. Schemata racing iteratively selects a random training sample, randomly selects a subset of features, and computes the selected sample's LOOCE error. Error statistics are maintained for each feature. If the error when selecting (dropping) a feature is significantly higher than when dropping (selecting) it, that feature will be selected (dropped) in all future races. Racing continues until feature selections are finalized. On several synthetic tasks and four data sets, schemata racing reliably yields accuracies similar to those obtained by FSS and BSS, but often greatly reduces the computational costs of feature selection. Ricci and Aha [(Ricci and Aha, 1998)] reported that a variant of schemata racing performed well vs. no feature selection when using an error-correcting output representation with $k = 1$.

Some algorithms use best-first search to guide feature weighting. For example, CS-IBL (Tan, 1993) uses a novel approach for incremental feature selection. At each step, it evaluates the lowest-cost feature of the nearest neighbor, terminating when the nearest neighbor's distance is guaranteed to be smaller than any training sample from a competing class. On two tasks, CS-IBL reduced the number of feature evaluations by an order of magnitude, although it increased the total number of classification errors prior to convergence.

Representations of Weight and Sample Space

Most artificial intelligence research on feature weighting for lazy learners imposes constraining assumptions on the representations used for both weights

and samples. The following sections describe these constraints and mentions example algorithms that violate them.

Weight Space: Figure 1.1 lists two categories for distinguishing the feature weight space: *generality* and *matrix*. We discuss these characteristics in turn.

The *generality* of the weight space denotes a weight’s scope. Most algorithms that infer feature weight settings are *global*: their weights apply across the entire space. In contrast, *local* algorithms permit feature weights to vary in different parts of the sample space, and thus model a wider range of tasks. Local feature weighting methods include methods that weight features by *class*¹, *feature value*, and/or *sample space*. Feature value and sample space weighting methods can depend either on the training data or the query itself, which Atkeson et al. (Atkeson et al., 1997) refer to as *point-based* and *query-based* methods, respectively.

Creedy et al. (Creedy et al., 1992) introduced *per category feature importance* (PCF), which assigns feature weights using

$$w(f, c_j) = p(c_j|f) \quad (1.13)$$

Thus, the weight of feature f for class c_j is defined as the conditional probability that a sample \mathbf{x} is in c_j given x_f , averaged across all values for f . PCF assigns high weights to features that have high correlations with the given class. IB4 (Aha, 1992), mentioned previously, also infers class-specific weights. It uses a different weight-updating rate depending on class frequency. More recently, Howe and Cardie [(Howe and Cardie, 1997)] introduced *CDW*, a class-specific weighting algorithm for symbolic features that computes weights using

$$w(f, c_j) = \sum_{v \in f} |r(f, v, \{\mathbf{x}|\mathbf{x} \in c_j\}) - r(f, v, \{\mathbf{x}|\mathbf{x} \notin c_j\})| \quad (1.14)$$

where $r(f, v, X')$ is, for a training subset \mathbf{X}' , $P(x'_f = v|x' \in \mathbf{X}')$. CDW sets a feature’s weight for a class c_j according to the degree to which its distributions of values for c_j differ from its distributions for other classes. CDW outperformed k -NN (i.e., for both $k = 1$ and $k = 10$) on some data sets where feature relevance varied per class, but significantly lowered accuracies for some other data sets. This lead Howe and Cardie to investigate EF-CDW, which transforms the sample representation to include one feature for every feature-value pair. This sample and value-specific algorithm generally outperformed CDW and compared well with unweighted k -NN.

The value-difference metric (Stanfill and Waltz, 1986), one of the first similarity algorithms designed for symbolic features, assigns feature weights according to a feature’s value:

$$w(f, v) = \sqrt{\sum_{c_j \in \mathbf{J}} p(\mathbf{x} \in c_j | \mathbf{x} \in \mathbf{X}, x_f = v)^2} \quad (1.15)$$

This point-based algorithm performed well on a word pronunciation task, although it was not compared with unweighted k -NN. Ricci and Avesani’s [(Ricci and Avesani, 1995)] AASM, designed for continuous features, assigns a feature f ’s weight depending on whether $q_f > x_f$:

$$w(f, \mathbf{x}, \mathbf{q}) = \begin{cases} w_{x_f >} & \text{if } x_f \geq q_f \\ w_{x_f <} & \text{if } x_f < q_f \end{cases} \quad (1.16)$$

where $w_{x_f >}$ and $w_{x_f <}$ are the weights of sample \mathbf{x} at feature f on the respective “sides” of x_f . AASM is point-based because these weight settings are determined prior to being given a query. Ricci and Avesani reported favorable results for AASM as compared to k -NN ($k = 1$) and EACH (Salzberg, 1991) on four data sets.

Value-specific feature weighting approaches constrain weights to be identical for all samples with the same feature value. In contrast, sample-specific algorithms remove this constraint. Several sample-specific algorithms have been investigated. For example, Fukunaga and his colleagues (Short and Fukunaga, 1981; Fukunaga and Flick, 1982) described a query-based algorithm that computes its k nearest neighbors, and then computes each neighbor’s local distance function to find the query’s nearest neighbor. Local distance functions were computed by estimating the finite sample risk from the local neighborhood of a given sample, and then minimizing the difference between the finite sample risk and the asymptotic risk. Myles and Hand (Myles and Hand, 1990) extended this approach for multiclass tasks.

This local weighting approach is sensitive to noisy samples, and its many distinct local distance functions can obscure useful feature information. Therefore, Fukunaga and Flick [(Fukunaga and Flick, 1984)] proposed a global distance function that combines the information from all local functions into one global set of weights for a weighted Euclidean distance function. Similarly, Aha and Goldstone [(Aha and Goldstone, 1992)] combined local with global distance functions to compute sample-specific weights for GCM-ISW. It defines distance for numeric features using

$$d(\mathbf{x}, \mathbf{q}) = \sqrt{\sum_{f \in \mathbf{F}} w(f, \mathbf{x}, \mathbf{q}) \cdot \delta(x_f, q_f)^2} \quad (1.17)$$

where sample-specific weights are dynamically assigned using

$$w(f, \mathbf{x}, \mathbf{q}) = w(f, \mathbf{x}) \cdot \sqrt{1 - |x_f - q_f|} + w(f) \cdot \left(1 - \sqrt{1 - |x_f - q_f|}\right) \quad (1.18)$$

Equation 1.18 adds the contributions of \mathbf{x} ’s sample-specific weight $w(f, \mathbf{x})$ to the global weight $w(f)$. This distance function depends more on sample-specific weights when \mathbf{q} is similar to \mathbf{x} , and depends more on the global weights otherwise. Likewise, the updating algorithm for sample-specific weights has more influence when \mathbf{q} and \mathbf{x} are similar. GCM-ISW correlated significantly better with subject data than did its non-weighting and global weighting variants for

a categorization task where feature relevance varied in different parts of the sample space.

Domingos’s [(Domingos, 1997)] RC algorithm uses another sample-specific feature weighting algorithm. It drops features from \mathbf{x} if they (1) differ from \mathbf{x} ’s nearest sample and (2) removing them does not decrease overall LOOCE error. Using k -NN with $k = 1$, RC easily outperformed FSS and BSE on 24 data sets, and its benefits increased with increasing context dependency of feature relevance.

Bonzano et al. [(Bonzano et al., 1997)] describe a sample-specific weighting algorithm ISAC for problem solving tasks. Like RELIEF (Kira and Rendell, 1992; Kononenko, 1994) and RC (Domingos, 1997), ISAC randomly selects a training sample \mathbf{x} and updates feature weights based on \mathbf{x} ’s differences with similar samples. However, RELIEF learns only global weights, RC is limited to binary weights, and both use only the nearest positive and negative samples to update feature weights for classification tasks. In contrast, ISAC learns continuous weight settings using the k -nearest neighbors. Weight updates are based on whether a neighbor’s solution satisfies the selected sample’s problem. Positive updates are made for \mathbf{x} using

$$w_i(x) = w_i(x) + \text{Correct}_x / \text{Incorrect}_x \quad (1.19)$$

where Correct_x (Incorrect_x) is the number of times that \mathbf{x} has been correctly (incorrectly) retrieved. Negative updates decrement weights by the same ratio. Failure-driven weight updating was more valuable than success-driven updating for their air traffic control task, and weighting was more appropriate for training sets that contain redundant information.

Friedman [(Friedman, 1994)] described an alternative approach for local feature weighting that integrates strategies of lazy similarity learning with decision tree induction. His SCYTHE algorithm recursively zooms in on query \mathbf{q} along the most relevant feature, where local feature relevance is estimated from the estimated reduction in prediction error given that feature’s value. The most relevant feature is scaled at each recursive step such that a fixed fraction of the training samples fall outside of a predetermined range around \mathbf{q} . After discarding these samples, the new “most relevant” feature is determined, and the process is repeated until only k training samples remain. SCYTHE performed well in comparison to k -NN on several classification tasks.

Cardie and Howe’s [(Cardie and Howe, 1997)] sample-specific weighting approach induces a pruned decision tree from the training set to perform feature selection; features in the path that classify \mathbf{q} are selected. Continuous weights are then assigned to these features according to their entropy on the entire training set. Their algorithm performed well, in the context of minority class predictions, on a set of natural language learning tasks in comparison with unweighted k -NN.

The weight *matrix* of a feature weighting algorithm characterizes the types of weight settings that it can infer, where the matrix’s two dimensions are the

sample space’s features. Using no weights corresponds to a *diagonal* matrix whose values are all the same positive constant. Simple weighting methods also use a diagonal matrix, but allow weights to vary (i.e., feature selection approaches assign binary values while feature weighting approaches assign continuous values on the diagonal), where larger weights denote greater relevance.

Searching a continuous weight space does not always yield settings superior to feature selection. Kohavi et al. [(Kohavi et al., 1997)] showed that, for some data sets, increasing the number of feature weight settings beyond two yields few accuracy benefits, and can sometimes degrade performance. Their DIET algorithm uses a best-first search, guided by 10-fold cross validation, in a weight space where each feature has a weight in $\{0, 1/v, 2/v, \dots, (v-1)/v, 1\}$. Starting with “middle” weight settings, DIET explores all variants that increase or decrease each weight by $1/v$ (i.e., minding the boundaries). The best-performing weight set is used as the next starting point. DIET stops when continued search does not seem promising. On tests with synthesized tasks and several data sets, DIET often performed better using smaller values for v , which is evidence that overfitting can occur when using larger values for v .

Using diagonal matrices assumes that the performance task can be best modelled by stretching and shrinking the sample space along the individual feature dimensions. This prevents the representation of feature interactions, where some combination might prove particularly relevant. In contrast, a *full* matrix, where any matrix element can have a non-zero entry, allows arbitrary scaling of the sample space (Atkeson et al., 1997; Friedman, 1994). Mohri and Tanaka [(Mohri and Tanaka, 1994)] describe how to use full matrices with symbolic features, while Hastie and Tibshirani [(Hastie and Tibshirani, 1996)] describe local weighting methods that use full matrices. Weights need not be constants, but could be a polynomial or any arbitrarily complex function (e.g., a connectionist network).

Sample Space: Using full matrices relates to transforming the sample space. Algorithms that simply assign feature weights are usually insensitive to interacting or correlated features. In contrast, *principal components analysis* (PCA) can transform a sample space, re-representing it with a list of polynomial equations of the original features, ordered by nonincreasing variance. However, PCA’s bias is not always appropriate; features with low variance might actually have high predictive relevance.

QM2Y (Mohri and Tanaka, 1994) is a lazy similarity learner that attempts to solve this deficiency in PCA. It assigns the absolute values computed by *Quantification Method II* (QM2) (Hayashi, 1952) to set its feature weights in the following variant of Equation 1.6, where symbolic features with V_f values are replaced with $|V_f|$ binary features before transformation:

$$d(\mathbf{x}, \mathbf{q}) = \sum_{f' \in \mathbf{F}'} \left(\sum_{f \in \mathbf{F}} w(f', f) x_f \cdot \sum_{f \in \mathbf{F}} w(f', f) q_f \right)^2 \quad (1.20)$$

where \mathbf{F}' is the set of transformed (i.e., new) features and $w(f', f)$ is the weight of the given feature f for the transformed feature f' . QM2 locates a new set of features such that the sums of the squared distances of each sample to its projections on the successive feature dimensions are minimized. After transformation, data dimensionality is reduced by removing the transformed features with lowest variation. New feature values are computed using

$$x'_f = \sum_{f \in \mathbf{F}} w(f', f) x_f \quad (1.21)$$

where these weights are calculated to maximize the ratio of the variance between each class's samples to the variance of all samples.

QM2Y performed best among the set of lazy similarity algorithms tested by Mohri and Tanaka, but hypotheses explaining why have not yet been published. We briefly discuss knowledge-intensive similarity learners that transform the sample space in Section 1.3.3.

Evaluation Function.

Dependent Variables

Most feature weighting methods for lazy similarity learners have been developed for classification tasks, and where classification accuracy was the dominant dependent variable. Other variables exist, including those concerning memory requirements and speed. Also, other contexts exist, such as retrieval, where measures including precision and recall are more relevant.

An important and frequently overlooked dependent variable is *cost* (Turney, 1995). For some tasks, some types of prediction errors are more costly than others, which violates Equation 1.2. Wilke and Bergmann's [(Wilke and Bergmann, 1996)] algorithm KNN_{COST} defines the cost of a decision as

$$\sum_{c_i \in J} \sum_{c_j \in J} P_{ij} C_{ij} \quad (1.22)$$

where P_{ij} and C_{ij} are entries in a confusion matrix and a cost matrix, respectively. Using a conjugate gradient search procedure to assign weight settings for a weighted-distance variant of k -NN (with $k = 11$) on a classification task, they reported substantial reductions in cost for KNN_{COST} in comparison to an accuracy-optimizing weighting approach and a non-weighting variant.

Features can have evaluation costs. Tan [(Tan, 1993)] defined feature costs for a mobile robot's actions as an increasing function of their duration. In CS-IBL, the first stage selects the sample that maximizes the ratio of expected match success to feature cost:

$$\frac{\prod_{f \in \mathbf{F}'} P_{if}}{\sum_{f \in \mathbf{F}'} C(f) \prod_{k=1}^{f-1} (1 - P_{ik})} \quad (1.23)$$

where \mathbf{F} is the set of features whose values are known for a stored sample \mathbf{x} but not the query \mathbf{q} , P_{if} is the probability that feature f has the value x_i among training samples, and $C(f)$ is the cost incurred for evaluating f .

Smyth and Cunningham [(Smyth and Cunningham, 1995)] investigated the situation where features were free or had a fixed non-zero cost. Their NODAL_{cbr} algorithm integrates a lazy decision tree approach (Friedman et al., 1996) with a lazy similarity learner. Given \mathbf{q} , NODAL_{cbr} computes a set of similar samples using \mathbf{q} 's free features. An information theoretic criterion then determines the most informative non-free feature whose value, if known, would maximally reduce the remaining samples. Reminiscent of `SCYTHE` (Friedman, 1994), this step recurses until a most similar (rather than k) sample remains. NODAL_{cbr} consistently reduced the number of feature evaluations needed to retrieve correct solutions for two tasks.

NODAL_{cbr} , Trott and Leng's [(Trott and Leng, 1997)] IP algorithm, and ISAC (Bonzano et al., 1997) perform case retrieval, but none address the context of planning. Muñoz-Avila and Hüllen [(Muñoz Avila and Hüllen, 1996)] describe one weighting method for planning tasks, where weights, based on an algorithm similar to the one used in `EACH` (Salzberg, 1991), denote sample-specific relative feature relevance. Over a sequence of problem solving episodes, their weighting model increased the reliability that the retrieved sample solves the query problem (for two tasks). In particular, using their weighting method greatly reduced both the number of retrieved samples and solution failures in comparison with the same algorithm when not learning weights.

Evaluation Bias

Feature weighting algorithms that use an evaluation function other than the performance function are susceptible to bias mismatches between these functions, which can degrade performance (John et al., 1994). Thus John et al. advocate using wrapper rather than filter models (e.g., to set feature weights). Some evidence suggests that wrapper models are superior to filter models in this context (e.g., (Wettschereck et al., 1997)) when the dependent variable is classification accuracy. However, wrapper models are often more computationally expensive (i.e., when they evaluate a large number of feature weight settings). Filter models and efficient variants of wrapper models should be considered for tasks where computational expense is a concern.

1.3.3 Using Domain-Specific Information to Weight Features

Several researchers have reported benefits from using domain-specific information to assign feature weights (i.e., when an *oracle* is available). This is commonly done in case-based reasoning applications (e.g., (Shimazu et al., 1994)).

Domain-specific knowledge can usefully guide transformations of the sample space. For example, IB3-CI (Aha, 1991) induces feature combinations using knowledge of the target domain to bias its search. It significantly increased k -NN classification accuracy on one task. Turney instead examined the intro-

duction of a separate set of *contextual* features \mathbf{F}' , disjoint from the initial set \mathbf{F} . He used these to transform each sample \mathbf{x} into a sample \mathbf{y} :

$$y_i = (x_i - \mu_i(\mathbf{F}')) / \sigma_i(\mathbf{F}') \quad (1.24)$$

where $\mu_i(\mathbf{F}')$ is the expected value and $\sigma_i(\mathbf{F}')$ is the expected variation of x_i . Turney reported impressive accuracy increases when using variants of this weighting procedure on two tasks.

Domain-specific knowledge can also be used to select features directly. ROBBIE (Fox and Leake, 1995), which modifies the set of features used to retrieve samples using a k -NN variant, improved retrieval efficiency for a navigation task. Barletta and Mark [(Barletta and Mark, 1988)] described how an explanation-based learning (EBL) method can be used to select features for a lazy learner, while Cain et al. [(Cain et al., 1991)] used EBL to select the relevant features for each sample. Their lazy similarity algorithm uses these features to bias classification predictions.

1.4 ADDITIONAL CONTRIBUTIONS

1.4.1 Formal Contributions

Formal analyses that address feature weighting in lazy similarity learners include Satoh and Okamoto's [(Satoh and Okamoto, 1994)] PAC-learning analysis for weighting features, and Ling and Wang's [(Ling and Wang, 1997)] algorithm for computing optimal weights for a restricted set of classification tasks, given knowledge of the concept boundary's location. Their approach can be used to evaluate the performance of heuristic methods for learning feature weight settings on some synthesized classification tasks. Also, Griffiths and Bridge [(Griffiths and Bridge, 1997)] describe PAC analyses for VS-CBR, a feature selecting variant of k -NN that selectively retains samples. They stress the importance of defining VS-CBR's hypothesis space, and of analysing its constituent algorithms.

1.4.2 Other Roles for Weights in Lazy Learners

A "weighted" lazy learner does not necessarily refer to feature weighting. For example, distance-weighted variants of k -NN, which assign relevance weights to samples as a function of their similarity to a query, have been studied for several years (Dudani, 1975). Related algorithms (e.g., EACH (Salzberg, 1991)) assign weights to samples based on their recorded performance in prediction tasks. See (Atkeson et al., 1997; Wettschereck et al., 1997) for more discussion on this topic.

Hastings et al. [(Hastings et al., 1995)] show how weights can help adapt solutions from those retrieved by CARMA, a lazy similarity algorithm for predicting rangeland pest consumption. It uses a hill-climbing algorithm that varies weights to minimize root mean squared error. CARMA can learn these weights in one of two modes: one set of weights for the entire training set or one set per sample. These weights have helped to improve CARMA's performance.

1.4.3 Avoiding Feature Selection

Several methods essentially remove the need for assigning feature weights. For example, Güvenir and his colleagues (e.g., (Güvenir and Sirin, 1996)) recommend voting among the individual features' matches. This avoids the need to combine the contributions of different features in a similarity function. Alternatively, feature transformation can obviate the need for feature weighting. For example, Chang and Lippman's [(Chang and Lippman, 1991)] GA recursively searches for constrained polynomial combinations of features. It reduced k -NN's error from 19% to zero on one task by replacing the original representation with a single feature. Finally, Datta and Kibler [(Datta and Kibler, 1997)] argued that, for some similarity functions, a feature's range of values determines a feature's impact on similarity computations.

1.5 SUMMARY

This chapter introduced a categorization framework for feature weighting algorithms in lazy similarity learners. Our focus was on the search algorithms used in automated weighting methods, although algorithms exemplifying other categories of this framework were discussed. This chapter contrasts with the framework presented in (Wettschereck et al., 1997), which focused on a set of unrelated dimensions rather than a categorization hierarchy. We did not discuss relations to feature weighting approaches used in other types of learning algorithms. Additional information on feature weighting in lazy learners can be found in (Dasarathy, 1991) and (Aha, 1997).

Although many more contributions on feature weighting for lazy similarity learners have been published (e.g., with similarity functions other than those based on Minkowski metrics), space constraints prevented their discussion. Also, our categorization framework is incomplete, and several dimensions for distinguishing these algorithms were not flushed out (e.g., offline vs. online algorithms, sensitivity to solution cluster distributions), but instead were briefly mentioned in the context of specific algorithms.

Several issues are good targets for further research. These include using full weight matrices, performance tasks other than classification, average case analyses, and examining how cognitively plausible categorization algorithms can impact the design of computationally effective lazy learners.

Acknowledgments

This research was supported by the Office of Naval Research.

Notes

1. Or training cluster, for tasks other than classification.

References

- Aha, D. W. (1991). Incremental constructive induction: An instance-based approach. In *Proceedings of the Eighth International Workshop on Machine Learning*, pages 117–121, Evanston, IL. Morgan Kaufmann.
- Aha, D. W. (1992). Tolerating noisy, irrelevant, and novel attributes in instance-based learning algorithms. *International Journal of Man-Machine Studies*, 36:267–287.
- Aha, D. W., editor (1997). *Lazy Learning*. Kluwer, Norwell, MA.
- Aha, D. W. and Bankert, R. L. (1996). A comparative evaluation of sequential feature selection algorithms. In Fisher, D. and Lenz, J.-H., editors, *Artificial Intelligence and Statistics V*. Springer, New York.
- Aha, D. W. and Goldstone, R. L. (1992). Concept learning and flexible weighting. In *Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society*, pages 534–539, Bloomington, IN. Lawrence Erlbaum.
- Atkeson, C., Moore, A., and Schaal, S. (1997). Locally weighted learning. *AI Review*, 11:11–73.
- Barletta, R. and Mark, W. (1988). Explanation-based indexing of cases. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 541–546, St. Paul, MN. Morgan Kaufmann.
- Barsalou, L. W. (1983). Ad hoc categories. *Memory & Cognition*, 11:211–227.
- Bonzano, A., Cunningham, P., and Smyth, B. (1997). Using introspective learning to improve retrieval in CBR: A case study in air traffic control. In *Proceedings of the Second ICCBR Conference*, pages 291–302, Providence, RI. Springer.
- Borrajo, D. and Veloso, M. (1997). Lazy incremental learning of control knowledge for efficiently obtaining quality plans. *AI Review*, 11:371–405.
- Cain, T., Pazzani, M. J., and Silverstein, G. (1991). Using domain knowledge to influence similarity judgement. In *Proceedings of a Case-Based Reasoning Workshop*, pages 191–202, Washington, DC. Morgan Kaufmann.
- Cardie, C. (1993). Using decision trees to improve case-based learning. In *Proceedings of the Tenth ICML*, pages 25–32, Amherst, MA. Morgan Kaufmann.
- Cardie, C. and Howe, N. (1997). Improving minority-class prediction using case-specific feature weights. In *Proceedings of the Fourteenth ICML*, pages 57–65, Nashville, TN. Morgan Kaufmann.
- Chang, E. I. and Lippman, R. P. (1991). Using genetic algorithms to improve pattern classification performance. In Lippman, R., Moody, J., and Touretzky, D. S., editors, *Advances in Neural Information Processing Systems 3*. Morgan Kaufmann, Denver, CO.
- Creecy, R. H., Masand, B. M., Smith, S. J., and Waltz, D. L. (1992). Trading mips and memory for knowledge engineering. *CACM*, 35:48–64.
- Daelemans, W. and van den Bosch, A. (1992). Generalization performance of backpropagation learning on a syllabification task. In *Proceedings of TWLT3: Connectionism and Natural Language Processing*, pages 27–37, Enschede, The Netherlands. Unpublished.

- Dasarathy, B. V. (1991). *Nearest neighbor(NN) norms: NN pattern classification techniques*. IEEE Computer Society Press, Los Alamitos, CA.
- Datta, P. and Kibler, D. (1997). Learning symbolic prototypes. In *Proceedings of the Fourteenth ICML*, pages 158–166, Tahoe City, CA. Morgan Kaufmann.
- Domingos, P. (1997). Context-sensitive feature selection for lazy learners. *AI Review*, 11:227–253.
- Dudani, S. (1975). The distance-weighted k-nearest neighbor rule. *IEEE Transactions on Systems, Man, and Cybernetics*, 6:325–327.
- Fix, E. and J. L. Hodges, Jr. (1951). Discriminatory analysis, nonparametric discrimination, consistency properties. Technical Report 4, United States Air Force, School of Aviation Medicine.
- Fox, S. and Leake, D. L. (1995). Using introspective reasoning to refine indexing. In *Proceedings of the Fourteenth IJCAI*, pages 391–397, Montreal. Morgan Kaufmann.
- Friedman, J. H. (1994). Flexible metric nearest neighbor classification. Technical report, Stanford University, Department of Statistics.
- Friedman, J. H., Kohavi, R., and Yun, Y. (1996). Lazy decision trees. In *Proceeding of the Thirteenth National Conference on Artificial Intelligence*, pages 717–724, Portland, OR. AAAI Press.
- Fukunaga, K. and Flick, T. (1982). A parametrically-defined nearest neighbor distance measure. *Pattern Recognition Letters*, 1:3–5.
- Fukunaga, K. and Flick, T. (1984). An optimal global nearest neighbor metric. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6:314–318.
- Griffiths, A. G. and Bridge, D. G. (1997). PAC analyses of a ‘similarity learning’ IBL algorithm. In *Proceedings of the Second ICCBR Conference*, pages 445–454, Providence, RI. Springer.
- Güvenir, H. A. and Sirin, I. (1996). Classification by feature partitioning. *Machine Learning*, 23:47–68.
- Hastie, T. and Tibshirani, R. (1996). Discriminant adaptive nearest neighbor classification. *IEEE Pattern Analysis and Machine Intelligence*, 18:607–616.
- Hastings, J., Branting, L. K., and Lockwood, J. A. (1995). Case adaptation using an incomplete causal model. In *Proceedings of the First ICCBR Conference*, pages 181–192, Sesimbra, Portugal. Springer.
- Howe, N. and Cardie, C. (1997). Examining locally varying weights for nearest neighbor algorithms. In *Proceedings of the Second ICCBR Conference*, pages 455–466, Providence, RI. Springer.
- J. D. Kelly, Jr. and Davis, L. (1991). A hybrid genetic algorithm for classification. In *Proceedings of the Twelfth IJCAI*, pages 645–650, Sydney, Australia. Morgan Kaufmann.
- John, G., Kohavi, R., and Pflieger, K. (1994). Irrelevant features and the subset selection problem. In *Proceedings of the Eleventh ICML*, pages 121–129, New Brunswick, NJ. Morgan Kaufmann.

- Kibler, D. and Aha, D. W. (1987). Learning representative exemplars of concepts: An initial case study. In *Proceedings of the Fourth International Workshop on Machine Learning*, pages 24–30, Irvine, CA. Morgan Kaufmann.
- Kira, K. and Rendell, L. A. (1992). The feature selection problem: Traditional methods and a new algorithm. In *Proceedings of the Tenth IJCAI*, pages 129–134, San Jose, CA. AAAI Press.
- Kohavi, R., Langley, P., and Yun, Y. (1997). The utility of feature weighting in nearest neighbor algorithms. In van Someren, M. and Widmer, G., editors, *Poster Papers: Ninth ECML*. Unpublished, Prague, Czech Republic.
- Kononenko, I. (1994). Estimating attributes: Analysis and extensions of relief. In *Proceedings of the Seventh ECML*, pages 171–182, Catania, Italy. Springer.
- Kontkanen, P., Myllymaki, P., Silander, T., and Tirri, H. (1998). Bayes optimal instance-based learning. In *Proceedings of the Tenth ECML*, Chemnitz, Germany. Springer.
- Lachiche, N. and Marquis, P. (1998). Scope classification: An instance-based learning algorithm with a rule-based characterization. In *Proceedings of the Tenth ECML*, Chemnitz, Germany. Springer.
- Langley, P. and Iba, W. (1993). Average-case analysis of a nearest neighbor algorithm. In *Proceedings of the Thirteenth IJCAI*, pages 889–894, Chambéry, France. Morgan Kaufmann.
- Leng, B. and Trott, J. (1997). Automatically tuning question weights in CBR Express 2.x. Unpublished Manuscript.
- Ling, X. C. and Wang, H. (1997). Towards optimal weights setting for the 1-nearest neighbour learning algorithm. *AI Review*, 11:255–272.
- Lowe, D. (1995). Similarity metric learning for a variable-kernal classifier. *Neural Computation*, 7:72–85.
- Maron, O. and Moore, A. W. (1997). The racing algorithm: Model selection for lazy learners. *AI Review*, 11:192–225.
- Mitchell, T. (1997). *Machine Learning*. McGraw-Hill, New York.
- Mohri, T. and Tanaka, H. (1994). An optimal weighting criterion of case indexing for both numeric and symbolic attributes. In Aha, D. W., editor, *Case-Based Reasoning: Papers from the 1994 Workshop*. AAAI Press, Menlo Park, CA.
- Myles, J. and Hand, D. (1990). The multi-class metric problem in nearest neighbor discrimination rules. *Pattern Recognition*, 23:1291–1297.
- Muñoz Avila, H. M. and Hüllen, J. (1996). Feature weighting by explaining case-based planning episodes. In *Proceedings of the Third European Workshop on Case-Based Reasoning*, pages 280–294, Lausanne, Switzerland. Springer.
- Ricci, F. and Aha, D. W. (1998). Error-correcting output codes for local learners. In *Proceedings of the Tenth ECML*, Chemnitz, Germany. Springer.
- Ricci, F. and Avesani, P. (1995). Learning a local similarity metric for case-based reasoning. In *Proceedings of the First ICCBR Conference*, pages 301–312, Sesimbra, Portugal. Springer.

- Salzberg, S. L. (1991). A nearest hyperrectangle learning method. *Machine Learning*, 6:251–276.
- Satoh, K. and Okamoto, S. (1994). Toward PAC-learning of weights from qualitative distance information. In Aha, D. W., editor, *Case-Based Reasoning: Papers from the 1994 Workshop*. AAAI Press, Menlo Park, CA.
- Shimazu, H., Shibata, A., and Nihei, K. (1994). Case-based retrieval interface adapted to customer-initiated dialogues in help desk operations. In *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pages 513–518, Seattle, WA. AAAI Press.
- Short, R. and Fukunaga, K. (1981). The optimal distance measure for nearest neighbor classification. *IEEE Transactions on Information Theory*, 27:622–627.
- Skalak, D. (1994). Prototype and feature selection by sampling and random mutation hill climbing algorithms. In *Proceedings of the Eleventh ICML*, pages 293–301, New Brunswick, NJ. Morgan Kaufmann.
- Smyth, B. and Cunningham, P. (1995). A comparison of incremental case-based reasoning and inductive learning. In Haton, J. P. and Keane, M., editors, *Advances in Case-Based Reasoning*. Springer, Berlin.
- Stanfill, C. and Waltz, D. (1986). Toward memory-based reasoning. *CACM*, 29:1213–1228.
- Tan, M. (1993). Cost-sensitive learning of classification knowledge and its application in robotics. *Machine Learning*, 13:7–34.
- Trott, J. R. and Leng, B. (1997). An engineering approach for troubleshooting case bases. In *Proceedings of the Second ICCBR Conference*, pages 178–189, Providence, RI. Springer.
- Turney, P. D. (1995). Cost-sensitive classification: Empirical evaluation of a hybrid genetic decision tree induction algorithm. *Journal of Artificial Intelligence Research*, 2:369–409.
- Wettschereck, D. (1994). A study of distance-based machine learning algorithms. Doctoral dissertation, Oregon State University, Department of Computer Science.
- Wettschereck, D., Aha, D. W., and Mohri, T. (1997). A review and empirical comparison of feature weighting methods for a class of lazy learning algorithms. *AI Review*, 11:273–314.
- Wilke, W. and Bergmann, R. (1996). Considering decision costs while learning of feature weights. In *Proceedings of the Third European Workshop on Case-Based Reasoning*, pages 460–472, Lausanne, Switzerland. Springer.
- Wilson, D. R. and Martinez, T. R. (1996). Instance-based learning with genetically derived attribute weights. In *Proceedings of the International Conference on Artificial Intelligence, Expert Systems, and Neural Networks*, pages 11–14. Unpublished.