Chapter 6

Instance-Based Learning: A Survey

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, NY
charu@us.ibm.com

6.1 Introduction

Most classification methods are based on building a model in the training phase, and then using this model for specific test instances, during the actual classification phase. Thus, the classification process is usually a two-phase approach that is cleanly separated between processing training and test instances. As discussed in the introduction chapter of this book, these two phases are as follows:

- *Training Phase:* In this phase, a model is constructed from the training instances.
- *Testing Phase:* In this phase, the model is used to assign a label to an unlabeled test instance.

Examples of models that are created during the first phase of training are decision trees, rule-based methods, neural networks, and support vector machines. Thus, the first phase creates *pre-compiled abstractions* or *models* for learning tasks. This is also referred to as *eager* learning, because the models are constructed in an eager way, without waiting for the test instance. In instance-based

learning, this clean separation between the training and testing phase is usually not present. The specific instance, which needs to be classified, is used to create a model that is *local* to a specific test instance. The classical example of an instance-based learning algorithm is the k-nearest neighbor classification algorithm, in which the k nearest neighbors of a classifier are used in order to create a local model for the test instance. An example of a local model using the k nearest neighbors could be that the majority class in this set of k instances is reported as the corresponding label, though more complex models are also possible. Instance-based learning is also sometimes referred to as *lazy* learning, since most of the computational work is not done upfront, and one waits to obtain the test instance, before creating a model for it [9]. Clearly, instance-based learning has a different set of tradeoffs, in that it requires very little or no processing for creating a *global* abstraction of the training data, but can sometimes be expensive at classification time. This is because instance-based learning typically has to determine the relevant local instances, and create a local model from these instances at classification time. While the obvious way to create a local model is to use a k-nearest neighbor classifier, numerous other kinds of lazy solutions are possible, which combine the power of lazy learning with other models such as locally-weighted regression, decision trees, rule-based methods, and SVM classifiers [15, 36, 40, 77]. This chapter will discuss all these different scenarios. It is possible to use the traditional "eager" learning methods such as Bayes methods [38], SVM methods [40], decision trees [62], or neural networks [64] in order to improve the effectiveness of local learning algorithms, by applying them only on the local neighborhood of the test instance at classification time.

It should also be pointed out that many instance-based algorithms may require a pre-processing phase in order to improve the efficiency of the approach. For example, the efficiency of a nearest neighbor classifier can be improved by building a similarity index on the training instances. In spite of this pre-processing phase, such an approach is still considered lazy learning or instance-based learning since the pre-processing phase is not really a classification model, but a data structure that enables efficient implementation of the run-time modeling for a given test instance.

Instance-based learning is related to but not quite the same as case-based reasoning [1, 60, 67], in which previous examples may be used in order to make predictions about specific test instances. Such systems can modify cases or use parts of cases in order to make predictions. Instance-based methods can be viewed as a particular kind of case-based approach, which uses specific kinds of algorithms for instance-based classification. The framework of instance-based algorithms is more amenable for reducing the computational and storage requirements, noise and irrelevant attributes. However, these terminologies are not clearly distinct from one another, because many authors use the term "case-based learning" in order to refer to instance-based learning algorithms. Instance-specific learning can even be extended to distance function learning, where instance-specific distance functions are learned, which are local to the query instance [76].

Instance-based learning methods have several advantages and disadvantages over traditional learning methods. The lazy aspect of instance-based learning is its greatest advantage. The global pre-processing approach of eager learning algorithms is inherently myopic to the characteristics of specific test instances, and may create a model, which is often not optimized towards specific instances. The advantage of instance-based learning methods is that they can be used in order to create models that are optimized to specific test instances. On the other hand, this can come at a cost, since the computational load of performing the classification can be high. As a result, it may often not be possible to create complex models because of the computational requirements. In some cases, this may lead to oversimplification. Clearly, the usefulness of instance-based learning (as in all other class of methods) depends highly upon the data domain, size of the data, data noisiness and dimensionality. These aspects will be covered in some detail in this chapter.

This chapter will provide an overview of the basic framework for instance-based learning, and the many algorithms that are commonly used in this domain. Some of the important methods such as nearest neighbor classification will be discussed in more detail, whereas others will be covered at a much higher level. This chapter is organized as follows. Section 6.2 introduces the basic framework

for instance-based learning. The most well-known instance-based method is the nearest neighbor classifier. This is discussed in Section 6.3. Lazy SVM classifiers are discussed in Section 6.4. Locally weighted methods for regression are discussed in section 6.5. Locally weighted naive Bayes methods are introduced in Section 6.6. Methods for constructing lazy decision trees are discussed in Section 6.7. Lazy rule-based classifiers are discussed in Section 6.8. Methods for using neural networks in the form of radial basis functions are discussed in Section 6.9. The advantages of lazy learning for diagnostic classification are discussed in Section 6.10. The conclusions and summary are discussed in Section 6.11.

6.2 Instance-Based Learning Framework

The earliest instance-based algorithms were synonymous with nearest neighbor pattern classification [31, 33], though the field has now progressed well beyond the use of such algorithms. These algorithms were often criticized for a number of shortcomings, especially when the data is high dimensional, and distance-function design is too challenging [45]. In particular, they were seen to be computationally expensive, intolerant of attribute noise, and sensitive to the choice of distance function [24]. Many of these shortcomings have subsequently been addressed, and these will be discussed in detail in this chapter.

The principle of instance-based methods was often understood in the earliest literature as follows:

"*... similar instances have similar classification.*" (Page 41, [11])

However, a broader and more powerful principle to characterize such methods would be:

Similar instances are easier to model with a learning algorithm, because of the simplification of the class distribution within the locality of a test instance.

Note that the latter principle is a bit more general than the former, in that the former principle seems to advocate the use of a nearest neighbor classifier, whereas the latter principle seems to suggest that *locally optimized* models to the test instance are usually more effective. Thus, according to the latter philosophy, a vanilla nearest neighbor approach may not always obtain the most accurate results, but a locally optimized regression classifier, Bayes method, SVM or decision tree may sometimes obtain better results because of the simplified modeling process [18, 28, 38, 77, 79]. This class of methods is often referred to as *lazy learning*, and often treated differently from traditional instance-based learning methods, which correspond to nearest neighbor classifiers. Nevertheless, the two classes of methods are closely related enough to merit a unified treatment. Therefore, this chapter will study both the traditional instance-based learning methods and lazy learning methods within a single generalized umbrella of instance-based learning methods.

The primary output of an instance-based algorithm is a concept description. As in the case of a classification model, this is a function that maps instances to category values. However, unlike traditional classifiers, which use extensional concept descriptions, instance-based concept descriptions may typically contain a set of stored instances, and optionally some information about how the stored instances may have performed in the past during classification. The set of stored instances can change as more instances are classified over time. This, however, is dependent upon the underlying classification scenario being temporal in nature. There are three primary components in all instance-based learning algorithms.

1. *Similarity or Distance Function:* This computes the similarities between the training instances, or between the test instance and the training instances. This is used to identify a locality around the test instance.

2. *Classification Function:* This yields a classification for a particular test instance with the use of the locality identified with the use of the distance function. In the earliest descriptions of instance-based learning, a nearest neighbor classifier was assumed, though this was later expanded to the use of any kind of locally optimized model.

3. *Concept Description Updater:* This typically tracks the classification performance, and makes decisions on the choice of instances to include in the concept description.

Traditional classification algorithms construct explicit abstractions and generalizations (e.g., decision trees or rules), which are constructed *in an eager way in a pre-processing phase*, and are independent of the choice of the test instance. These models are then used in order to classify test instances. This is different from instance-based learning algorithms, where instances are used along with the training data to construct the concept descriptions. Thus, the approach is *lazy* in the sense that knowledge of the test instance is required before model construction. Clearly the tradeoffs are different in the sense that "eager" algorithms avoid too much work at classification time, but are myopic in their ability to create a specific model for a test instance in the most accurate way. Instance-based algorithms face many challenges involving efficiency, attribute noise, and significant storage requirements. A work that analyzes the last aspect of storage requirements is discussed in [72].

While nearest neighbor methods are almost always used as an intermediate step for identifying data locality, a variety of techniques have been explored in the literature beyond a majority vote on the identified locality. Traditional modeling techniques such as decision trees, regression modeling, Bayes, or rule-based methods are commonly used to create an optimized classification model around the test instance. *It is the optimization inherent in this localization that provides the greatest advantages of instance-based learning.* In some cases, these methods are also combined with some level of global pre-processing so as to create a combination of instance-based and model-based algorithms [55]. In any case, many instance-based methods combine typical classification generalizations such as regression-based methods [15], SVMs [54, 77], rule-based methods [36], or decision trees [40] with instance-based methods. Even in the case of pure distance-based methods, some amount of model building may be required at an early phase for learning the underlying distance functions [75]. This chapter will also discuss such techniques within the broader category of instance-based methods.

6.3 The Nearest Neighbor Classifier

The most commonly used instance-based classification method is the nearest neighbor method. In this method, the nearest k instances to the test instance are determined. Then, a simple model is constructed on this set of k nearest neighbors in order to determine the class label. For example, the majority class among the k nearest neighbors may be reported as the relevant labels. For the purpose of this paper, we always use a binary classification (two label) assumption, in which case the use of the majority class is relevant. However, the method can be easily extended to the multi-class scenario very easily by using the class with the largest presence, rather than the majority class. Since the different attributes may be defined along different scales (e.g., age versus salary), a common approach is to scale the attributes either by their respective standard deviations or the observed range of that attribute. The former is generally a more sound approach from a statistical

point of view. It has been shown in [31] that the nearest neighbor rule provides at most twice the error as that provided by the local Bayes probability.

Such an approach may sometimes not be appropriate for imbalanced data sets, in which the rare class may not be present to a significant degree among the nearest neighbors, even when the test instance belongs to the rare class. In the case of cost-sensitive classification or rare-class learning the majority class is determined after weighting the instances with the relevant costs. These methods will be discussed in detail in Chapter 17 on rare class learning. In cases where the class label is continuous (regression modeling problem), one may use the weighted average numeric values of the target class. Numerous variations on this broad approach are possible, both in terms of the distance function used or the local model used for the classification process.

- The choice of the distance function clearly affects the behavior of the underlying classifier. In fact, the problem of distance function learning [75] is closely related to that of instance-based learning since nearest neighbor classifiers are often used to validate distance-function learning algorithms. For example, for numerical data, the use of the euclidian distance assumes a spherical shape of the clusters created by different classes. On the other hand, the true clusters may be ellipsoidal and arbitrarily oriented with respect to the axis system. Different distance functions may work better in different scenarios. The use of feature-weighting [69] can also change the distance function, since the weighting can change the contour of the distance function to match the patterns in the underlying data more closely.

- The final step of selecting the model from the local test instances may vary with the application. For example, one may use the majority class as the relevant one for classification, a cost-weighted majority vote, or a more complex classifier within the locality such as a Bayes technique [38, 78].

One of the nice characteristics of the nearest neighbor classification approach is that it can be used for practically any data type, as long as a distance function is available to quantify the distances between objects. Distance functions are often designed with a specific focus on the classification task [21]. Distance function design is a widely studied topic in many domains such as time-series data [42], categorical data [22], text data [56], and multimedia data [58] or biological data [14]. Entropy-based measures [29] are more appropriate for domains such as strings, in which the distances are measured in terms of the amount of effort required to transform one instance to the other. Therefore, the simple nearest neighbor approach can be easily adapted to virtually every data domain. This is a clear advantage in terms of usability. A detailed discussion of different aspects of distance function design may be found in [75].

A key issue with the use of nearest neighbor classifiers is the *efficiency* of the approach in the classification process. This is because the retrieval of the k nearest neighbors may require a running time that is linear in the size of the data set. With the increase in typical data sizes over the last few years, this continues to be a significant problem [13]. Therefore, it is useful to create indexes, which can efficiently retrieve the k nearest neighbors of the underlying data. This is generally possible for many data domains, but may not be true of all data domains in general. Therefore, scalability is often a challenge in the use of such algorithms. A common strategy is to use either indexing of the underlying instances [57], sampling of the data, or aggregations of some of the data points into smaller clustered pseudo-points in order to improve accuracy. While the indexing strategy seems to be the most natural, it rarely works well in the high dimensional case, because of the curse of dimensionality. Many data sets are also very high dimensional, in which case a nearest neighbor index fails to prune out a significant fraction of the data points, and may in fact do worse than a sequential scan, because of the additional overhead of indexing computations.

Such issues are particularly challenging in the streaming scenario. A common strategy is to use very fine grained clustering [5, 7] in order to replace multiple local instances within a small cluster (belonging to the same class) with a pseudo-point of that class. Typically, this pseudo-point is the

centroid of a small cluster. Then, it is possible to apply a nearest neighbor method on these pseudo-points in order to obtain the results more efficiently. Such a method is desirable, when scalability is of great concern and the data has very high volume. Such a method may also reduce the noise that is associated with the use of individual instances for classification. An example of such an approach is provided in [7], where classification is performed on a fast data stream, by summarizing the stream into micro-clusters. Each micro-cluster is constrained to contain data points only belonging to a particular class. The class label of the closest micro-cluster to a particular instance is reported as the relevant label. Typically, the clustering is performed with respect to different time-horizons, and a cross-validation approach is used in order to determine the time-horizon that is most relevant at a given time. Thus, the model is instance-based in a dual sense, since it is not only a nearest neighbor classifier, but it also determines the relevant time horizon in a lazy way, which is specific to the time-stamp of the instance. Picking a smaller time horizon for selecting the training data may often be desirable when the data evolves significantly over time. The streaming scenario also benefits from laziness in the temporal dimension, since the most appropriate model to use for the same test instance may vary with time, as the data evolves. It has been shown in [7], that such an "on demand" approach to modeling provides more effective results than eager classifiers, because of its ability to optimize for the test instance from a temporal perspective. Another method that is based on the nearest neighbor approach is proposed in [17]. This approach detects the changes in the distribution of the data stream on the past window of instances and accordingly re-adjusts the classifier. The approach can handle symbolic attributes, and it uses the Value Distance Metric (VDM) [60] in order to measure distances. This metric will be discussed in some detail in Section 6.3.1 on symbolic attributes.

A second approach that is commonly used to speed up the approach is the concept of *instance selection* or *prototype selection* [27, 41, 72, 73, 81]. In these methods, a subset of instances may be pre-selected from the data, and the model is constructed with the use of these pre-selected instances. It has been shown that a good choice of pre-selected instances can often lead to *improvement* in accuracy, in addition to the better efficiency [72, 81]. This is because a careful pre-selection of instances reduces the noise from the underlying training data, and therefore results in better classification. The pre-selection issue is an important research issue in its own right, and we refer the reader to [41] for a detailed discussion of this important aspect of instance-based classification. An empirical comparison of the different instance selection algorithms may be found in [47].

In many rare class or cost-sensitive applications, the instances may need to be weighted differently corresponding to their importance. For example, consider an application in which it is desirable to use medical data in order to diagnose a specific condition. The vast majority of results may be normal, and yet it may be costly to miss a case where an example is abnormal. Furthermore, a nearest neighbor classifier (which does not weight instances) will be naturally biased towards identifying instances as normal, especially when they lie at the border of the decision region. In such cases, costs are associated with instances, where the cost associated with an abnormal instance is the same as the relative cost of misclassifying it (false negative), as compared to the cost of misclassifying a normal instance (false positive). The weights on the instances are then used for the classification process.

Another issue with the use of nearest neighbor methods is that it does not work very well when the dimensionality of the underlying data increases. This is because the quality of the nearest neighbor decreases with an increasing number of irrelevant attributes [45]. The noise effects associated with the irrelevant attributes can clearly degrade the quality of the nearest neighbors found, especially when the dimensionality of the underlying data is high. This is because the cumulative effect of irrelevant attributes often becomes more pronounced with increasing dimensionality. For the case of numeric attributes, it has been shown [2], that the use of fractional norms (i.e. L_p-norms for $p < 1$) provides superior quality results for nearest neighbor classifiers, whereas L_∞ norms provide the poorest behavior. Greater improvements may be obtained by designing the distance function

more carefully, and weighting more relevant ones. This is an issue that will be discussed in detail in later subsections.

In this context, the issue of distance-function design is an important one [50]. In fact, an entire area of machine learning has been focussed on distance function design. Chapter 18 of this book has been devoted entirely to distance function design, and an excellent survey on the topic may be found in [75]. A discussion of the applications of different similarity methods for instance-based classification may be found in [32]. In this section, we will discuss some of the key aspects of instance-function design, which are important in the context of nearest neighbor classification.

6.3.1 Handling Symbolic Attributes

Since most natural distance functions such as the L_p-norms are defined for numeric attributes, a natural question arises as to how the distance function should be computed in data sets in which some attributes are symbolic. While it is always possible to use a distance-function learning approach [75] for an arbitrary data type, a simpler and more efficient solution may sometimes be desirable. A discussion of several unsupervised symbolic distance functions is provided in [22], though it is sometimes desirable to use the class label in order to improve the effectiveness of the distance function.

A simple, but effective supervised approach is to use the value-difference-metric (VDM), which is based on the class-distribution conditional on the attribute values [60]. The intuition here is that similar symbolic attribute values will show similar class distribution behavior, and the distances should be computed on this basis. Thus, this distance function is clearly a supervised one (unlike the euclidian metric), since it explicitly uses the class distributions in the training data.

Let x_1 and x_2 be two possible symbolic values for an attribute, and $P(C_i|x_1)$ and $P(C_i|x_2)$ be the conditional probabilities of class C_i for these values, respectively. These conditional probabilities can be estimated in a data-driven manner. Then, the value different metric $VDM(x_1,x_2)$ is defined as follows:

$$VDM(x_1,x_2) = \sum_{i=1}^{k} (P(C_i|x_1) - P(C_i|x_2))^q \tag{6.1}$$

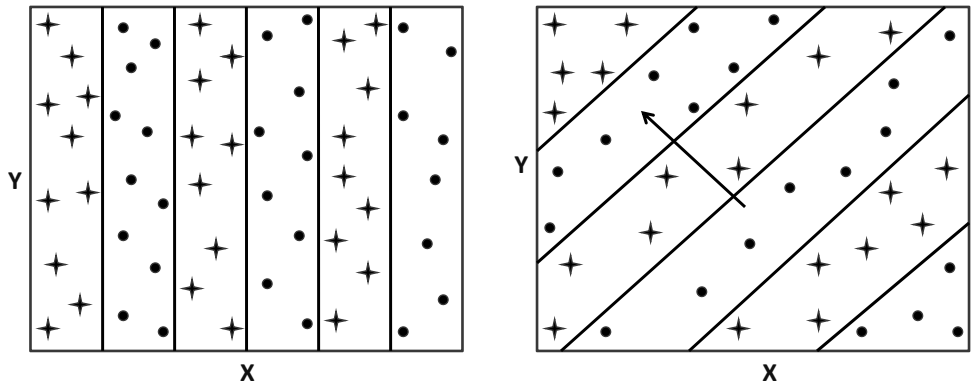
Here, the parameter q can be chosen either on an ad hoc basis, or in a data-driven manner. This choice of metric has been shown to be quite effective in a variety of instance-centered scenarios [36, 60]. Detailed discussions of different kinds of similarity functions for symbolic attributes may be found in [22, 30].

6.3.2 Distance-Weighted Nearest Neighbor Methods

The simplest form of the nearest neighbor method is when the the majority label among the k-nearest neighbor distances is used. In the case of distance-weighted neighbors, it is assumed that all nearest neighbors are not equally important for classification. Rather, an instance i, whose distance d_i to the test instance is smaller, is more important. Then, if c_i is the label for instance i, then the number of votes $V(j)$ for class label j from the k-nearest neighbor set S_k is as follows:

$$V(j) = \sum_{i:i\in S_k, c_i=j} f(d_i) \tag{6.2}$$

Here $f(\cdot)$ is either an increasing or decreasing function of its argument, depending upon when d_i represents similarity or distance, respectively. It should be pointed out that if the appropriate weight is used, then it is not necessary to use the k nearest neighbors, but simply to perform this average over the entire collection.



(a) Axis-Parallel (b) Arbitrarily Oriented

FIGURE 6.1: Illustration of importance of feature weighting for nearest neighbor classification.

6.3.3 Local Distance Scaling

A different way to improve the quality of the k-nearest neighbors is by using a technique, that is referred to as *local distance scaling* [65]. In local distance scaling, the distance of the test instance to each training example $\overline{X_i}$ is scaled by a weight r_i, which is specific to the training example $\overline{X_i}$. The weight r_i for the training example $\overline{X_i}$ is the largest distance from $\overline{X_i}$ such that it does not contain a training example from a class that is different from $\overline{X_i}$. Then, the new scaled distance $d(\hat{\overline{X}},\overline{X_i})$ between the test example $\overline{X}$ and the training example $\overline{X_i}$ is given by the following scaled value:

$$d(\hat{\overline{X}},\overline{X_i}) = d(\overline{X},\overline{X_i})/r_i \tag{6.3}$$

The nearest neighbors are computed on this set of distances, and the majority vote among the k nearest neighbors is reported as the class labels. This approach tends to work well because it picks the k nearest neighbor group in a noise-resistant way. For test instances that lie on the decision boundaries, it tends to discount the instances that lie on the noisy parts of a decision boundary, and instead picks the k nearest neighbors that lie away from the noisy boundary. For example, consider a data point $\overline{X_i}$ that lies reasonably close to the test instance, but even closer to the decision boundary. Furthermore, since $\overline{X_i}$ lies almost on the decision boundary, it lies extremely close to another training example in a different class. In such a case the example $\overline{X_i}$ should not be included in the k-nearest neighbors, because it is not very informative. The small value of r_i will often ensure that such an example is not picked among the k-nearest neighbors. Furthermore, such an approach also ensures that the distances are scaled and normalized by the varying nature of the patterns of the different classes in different regions. Because of these factors, it has been shown in [65] that this modification often yields more robust results for the quality of classification.

6.3.4 Attribute-Weighted Nearest Neighbor Methods

Attribute-weighting is the simplest method for modifying the distance function in nearest neighbor classification. This is closely related to the use of the Mahalanobis distance, in which arbitrary *directions* in the data may be weighted differently, as opposed to the actual attributes. The Mahalanobis distance is equivalent to the Euclidian distance computed on a space in which the different directions along an arbitrarily oriented axis system are "stretched" differently, according to the covariance matrix of the data set. Attribute weighting can be considered a simpler approach, in which the directions of stretching are parallel to the original axis system. By picking a particular weight

to be zero, that attribute is eliminated completely. This can be considered an implicit form of feature selection. Thus, for two d-dimensional records $\overline{X} = (x_1 \ldots x_d)$ and $\overline{Y} = (y_1 \ldots y_d)$, the feature weighted distance $d(\overline{X}, \overline{Y}, \overline{W})$ with respect to a d-dimensional vector of weights $\overline{W} = (w_1 \ldots w_d)$ is defined as follows:

$$d(\overline{X}, \overline{Y}, \overline{W}) = \sqrt{\sum_{i=1}^{d} w_i \cdot (x_i - y_i)^2}. \tag{6.4}$$

For example, consider the data distribution, illustrated in Figure 6.1(a). In this case, it is evident that the feature X is much more discriminative than feature Y, and should therefore be weighted to a greater degree in the final classification. In this case, almost perfect classification can be obtained with the use of feature X, though this will usually not be the case, since the decision boundary is noisy in nature. It should be pointed out that the Euclidian metric has a spherical decision boundary, whereas the decision boundary in this case is linear. This results in a bias in the classification process because of the difference between the shape of the model boundary and the true boundaries in the data. The importance of weighting the features becomes significantly greater, when the classes are not cleanly separated by a decision boundary. In such cases, the natural noise at the decision boundary may combine with the significant bias introduced by an unweighted Euclidian distance, and result in even more inaccurate classification. By weighting the features in the Euclidian distance, it is possible to elongate the model boundaries to a shape that aligns more closely with the class-separation boundaries in the data. The simplest possible weight to use would be to normalize each dimension by its standard deviation, though in practice, the class label is used in order to determine the best feature weighting [48]. A detailed discussion of different aspects of feature weighting schemes is provided in [70].

In some cases, the class distribution may not be aligned neatly along the axis system, but may be arbitrarily oriented along different directions in the data, as in Figure 6.1(b). A more general distance metric is defined with respect to a $d \times d$ matrix A rather than a vector of weights $\overline{W}$.

$$d(\overline{X}, \overline{Y}, A) = \sqrt{(\overline{X} - \overline{Y})^T \cdot A \cdot (\overline{X} - \overline{Y})}. \tag{6.5}$$

The matrix A is also sometimes referred to as a metric. The value of A is assumed to be the inverse of the covariance matrix of the data in the standard definition of the Mahalanobis distance for *unsupervised* applications. Generally, the Mahalanobis distance is more sensitive to the global data distribution and provides more effective results.

The Mahalanobis distance does not, however take the class distribution into account. In supervised applications, it makes much more sense to pick A based on the *class* distribution of the underlying data. The core idea is to "elongate" the neighborhoods along less discriminative directions, and to shrink the neighborhoods along more discriminative dimensions. Thus, in the modified metric, a small (unweighted) step along a discriminative direction, would result in relatively greater distance. This naturally provides greater importance to more discriminative directions. Numerous methods such as the linear discriminant [51] can be used in order to determine the most discriminative dimensions in the underlying data. However, the key here is to use a *soft* weighting of the different directions, rather than selecting specific dimensions in a hard way. The goal of the matrix A is to accomplish this. How can A be determined by using the distribution of the classes? Clearly, the matrix A should somehow depend on the within-class variance and between-class variance, in the context of linear discriminant analysis. The matrix A defines the shape of the neighborhood within a threshold distance, to a given test instance. The neighborhood directions with low ratio of inter-class variance to intra-class variance should be elongated, whereas the directions with high ratio of the inter-class to intra-class variance should be shrunk. Note that the "elongation" of a neighborhood direction is achieved by scaling that component of the distance by a larger factor, and therefore de-emphasizing that direction.

Let $\mathcal{D}$ be the full database, and $\mathcal{D}_i$ be the portion of the data set belonging to class i. Let $\overline{\mu}$ represent the mean of the entire data set. Let $p_i = |\mathcal{D}_i|/|\mathcal{D}|$ be the fraction of records belonging to class i, $\overline{\mu_i}$ be the d-dimensional row vector of means of $\mathcal{D}_i$, and Σ_i be the $d \times d$ covariance matrix of $\mathcal{D}_i$. Then, the scaled[1] within class scatter matrix S_w is defined as follows:

$$S_w = \sum_{i=1}^{k} p_i \cdot \Sigma_i. \tag{6.6}$$

The between-class scatter matrix S_b may be computed as follows:

$$S_b = \sum_{i=1}^{k} p_i (\overline{\mu_i} - \overline{\mu})^T (\overline{\mu_i} - \overline{\mu}). \tag{6.7}$$

Note that the matrix S_b is a $d \times d$ matrix, since it results from the product of a $d \times 1$ matrix with a $1 \times d$ matrix. Then, the matrix A (of Equation 6.5), which provides the desired distortion of the distances on the basis of class-distribution, can be shown to be the following:

$$A = S_w^{-1} \cdot S_b \cdot S_w^{-1}. \tag{6.8}$$

It can be shown that this choice of the metric A provides an excellent discrimination between the different classes, where the elongation in each direction depends inversely on ratio of the between-class variance to within-class variance along the different directions. The aforementioned description is based on the discussion in [44]. The reader may find more details of implementing the approach in an effective way in that work.

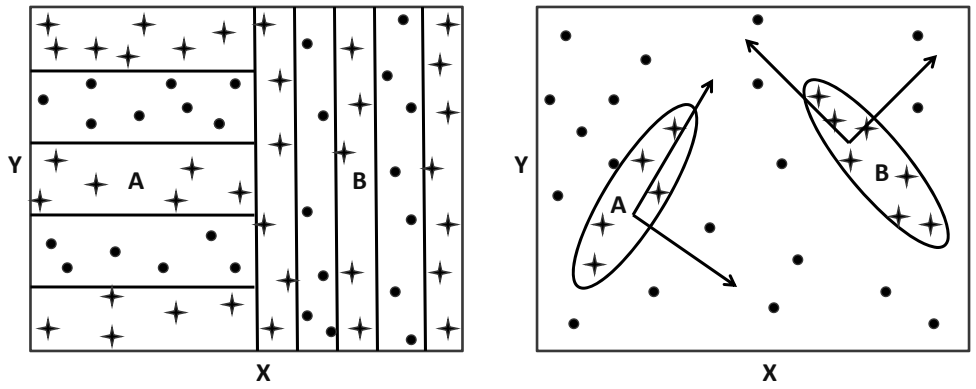
A few special cases of the metric of Equation 6.5 are noteworthy. Setting A to the identity matrix corresponds to the use of the Euclidian distance. Setting the non-diagonal entries of A entries to zero results in a similar situation to a d-dimensional vector of weights for individual dimensions. Therefore, the non-diagonal entries contribute to a rotation of the axis-system before the stretching process. For example, in Figure 6.1(b), the optimal choice of the matrix A will result in greater importance being shown to the direction illustrated by the arrow in the figure in the resulting metric. In order to avoid ill-conditioned matrices, especially in the case when the number of training data points is small, a parameter ε can be used in order to perform smoothing.

$$A = S_w^{-1/2} \cdot (S_w^{-1/2} \cdot S_b \cdot S_w^{-1/2} + \varepsilon \cdot I) \cdot S_w^{-1/2}. \tag{6.9}$$

Here ε is a small parameter that can be tuned, and the identity matrix is represented by I. The use of this modification assumes that any particular direction does not get infinite weight. This is quite possible, when the number of data points is small. The use of this parameter ε is analogous to *Laplacian smoothing* methods, and is designed to avoid overfitting.

Other heuristic methodologies are also used in the literature for learning feature relevance. One common methodology is to use cross-validation in which the weights are trained using the original instances in order to minimize the error rate. Details of such a methodology are provided in [10,48]. It is possible to also use different kinds of search methods such as Tabu search [61] in order to improve the process of learning weights. This kind of approach has also been used in the context of text classification, by learning the relative importance of the different words, for computing the similarity function [43]. Feature relevance has also been shown to be important for other domains such as image retrieval [53]. In cases where domain knowledge can be used, some features can be eliminated very easily with tremendous performance gains. The importance of using domain knowledge for feature weighting has been discussed in [26].

[1]The unscaled version may be obtained by multiplying S_w with the number of data points. There is no difference from the final result, whether the scaled or unscaled version is used, within a constant of proportionality.



(a) Axis-Parallel (b) Arbitrarily Oriented

FIGURE 6.2: Illustration of importance of local adaptivity for nearest neighbor classification.

The most effective distance function design may be performed at query-time, by using a weighting that is *specific* to the test instance [34,44]. This can be done by learning the weights only on the basis of the instances that are near a specific test instance. While such an approach is more expensive than global weighting, it is likely to be more effective because of local optimization specific to the test instance, and is better aligned with the spirit and advantages of lazy learning. This approach will be discussed in detail in later sections of this survey.

It should be pointed out that many of these algorithms can be considered rudimentary forms of distance function learning. The problem of distance function learning [3, 21, 75] is fundamental to the design of a wide variety of data mining algorithms including nearest neighbor classification [68], and a significant amount of research has been performed in the literature in the context of the classification task. A detailed survey on this topic may be found in [75]. The nearest neighbor classifier is often used as the prototypical task in order to evaluate the quality of distance functions [2, 3, 21, 45, 68, 75], which are constructed using distance function learning techniques.

6.3.5 Locally Adaptive Nearest Neighbor Classifier

Most nearest neighbor methods are designed with the use of a *global* distance function such as the Euclidian distance, the Mahalanobis distance, or a discriminant-based weighted distance, in which a particular set of weights is used over the entire data set. In practice, however, the importance of different features (or data directions) is not local, but global. This is especially true for high dimensional data, where the relative importance, relevance, and noise in the different dimensions may vary significantly with data locality. Even in the unsupervised scenarios, where no labels are available, it has been shown [45] that the relative importance of different dimensions for finding nearest neighbors may be very different for different localities. In the work described in [45], the relative importance of the different dimensions is determined with the use of a contrast measure, which is independent of class labels. Furthermore, the weights of different dimensions vary with the data locality in which they are measured, and these weights can be determined with the use of genetic algorithm discussed in [45]. Even for the case of such an unsupervised approach, it has been shown that the effectiveness of a nearest neighbor classifier improves significantly. The reason for this behavior is that different features are noisy in different data localities, and the local feature selection process sharpens the quality of the nearest neighbors.

Since local distance functions are more effective in unsupervised scenarios, it is natural to explore whether this is also true of supervised scenarios, where even greater information in available

in the form of different labels in order to make inferences about the relevance of the different dimensions. A recent method for distance function learning [76] also constructs instance-specific distances with supervision, and shows that the use of locality provides superior results. However, the supervision in this case is not specifically focussed on the traditional classification problem, since it is defined in terms of similarity or dissimilarity constraints between instances, rather than labels attached to instances. Nevertheless, such an approach can also be used in order to construct instance-specific distances for classification, by transforming the class labels into similarity or dissimilarity constraints. This general principle is used frequently in many works such as those discussed by [34, 39, 44] for locally adaptive nearest neighbor classification.

Labels provide a rich level of information about the relative importance of different dimensions in different localities. For example, consider the illustration of Figure 6.2(a). In this case, the feature Y is more important in locality A of the data, whereas feature X is more important in locality B of the data. Correspondingly, the feature Y should be weighted more, when using test instances in locality A of the data, whereas feature X should be weighted more in locality B of the data. In the case of Figure 6.2(b), we have shown a similar scenario as in Figure 6.2(a), except that different *directions* in the data should be considered more or less important. Thus, the case in Figure 6.2(b) is simply an arbitrarily oriented generalization of the challenges faced in Figure 6.2(a).

One of the earliest methods for performing locally adaptive nearest neighbor classification was proposed in [44], in which the matrix A (of Equation 6.5) and the neighborhood of the test data point $\overline{X}$ are learned iteratively from one another in a local way. The major difference here is that the matrix A will depend upon the choice of test instance X, since the metric is designed to be locally adaptive. The algorithm starts off by setting the $d \times d$ matrix A to the identity matrix, and then determines the k-nearest neighbors N_k, using the generalized distance described by Equation 6.5. Then, the value of A is updated using Equation 6.8 *only on the neighborhood points N_k found in the last iteration*. This procedure is repeated till convergence. Thus, the overall iterative procedure may be summarized as follows:

1. Determine N_k as the set of the k nearest neighbors of the test instance, using Equation 6.5 in conjunction with the current value of A.

2. Update A using Equation 6.8 on the between-class and within-class scatter matrices of N_k.

At completion, the matrix A is used in Equation 6.8 for k-nearest neighbor classification. In practice, Equation 6.9 is used in order to avoid giving infinite weight to any particular direction. It should be pointed out that the approach is [44] is quite similar in principle to an unpublished approach proposed previously in [39], though it varies somewhat on the specific details of the implementation, and is also somewhat more robust. Subsequently, a similar approach to [44] was proposed in [34], which works well with limited data. It should be pointed out that linear discriminant analysis is not the only method that may be used for "deforming" the shape of the metric in an instance-specific way to conform better to the decision boundary. Any other model such as an SVM or neural network that can find important directions of discrimination in the data may be used in order to determine the most relevant metric. For example, an interesting work discussed in [63] determines a small number of k-nearest neighbors for each class, in order to span a linear subspace for that class. The classification is done based not on distances to prototypes, but the distances to subspaces. The intuition is that the linear subspaces essentially generate pseudo training examples for that class. A number of methods that use support-vector machines will be discussed in the section on Support Vector Machines. A review and discussion of different kinds of feature selection and feature weighting schemes are provided in [8]. It has also been suggested in [48] that feature weighting may be superior to feature selection, in cases where the different features have different levels of relevance.

6.3.6 Combining with Ensemble Methods

It is evident from the aforementioned discussion that lazy learning methods have the advantage of being able to optimize more effectively to the locality of a specific instance. In fact, one issue that is encountered sometimes in lazy learning methods is that the quest for local optimization can result in overfitting, especially when some of the features are noisy within a specific locality. In many cases, it may be possible to create *multiple models* that are more effective for different instances, but it may be hard to know which model is more effective without causing overfitting. An effective method to achieve the goal of simultaneous local optimization and robustness is to use ensemble methods that combine the results from multiple models for the classification process. Some recent methods [4,16,80] have been designed for using ensemble methods in order to improve the effectiveness of lazy learning methods.

An approach discussed in [16] samples random subsets of features, which are used for nearest neighbor classification. These different classifications are then combined together in order to yield a more robust classification. One disadvantage of this approach is that it can be rather slow, especially for larger data sets. This is because the nearest neighbor classification approach is inherently computationally intensive because of the large number of distance computations *for every instance-based classification*. This is in fact one of the major issues for all lazy learning methods. A recent approach *LOCUST* discussed in [4] proposes a more efficient technique for ensemble lazy learning, which can also be applied to the streaming scenario. The approach discussed in [4] builds inverted indices on discretized attributes of each instance in real time, and then uses efficient intersection of randomly selected inverted lists in order to perform real time classification. Furthermore, a limited level of feature bias is used in order to reduce the number of ensemble components for effective classification. It has been shown in [4] that this approach can be used for resource-sensitive lazy learning, where the number of feature samples can be tailored to the resources available for classification. Thus, this approach can also be considered an *anytime* lazy learning algorithm, which is particularly suited to the streaming scenario. This is useful in scenarios where one cannot control the input rate of the data stream, and one needs to continuously adjust the processing rate, in order to account for the changing input rates of the data stream.

6.3.7 Multi-Label Learning

In traditional classification problems, each class is associated with exactly one label. This is not the case for multi-label learning, in which each class may be associated with more than one class. The number of classes with which an instance is associated is unknown a-priori. This kind of scenario is quite common in many scenarios such as document classification, where a single document may belong to one of several possible categories.

The unknown number of classes associated with a test instance creates a challenge, because one now needs to determine not just the most relevant classes, but also the number of relevant ones for a given test instance. One way of solving the problem is to decompose it into multiple independent binary classification problems. However, such an approach does not account for the correlations among the labels of the data instances.

An approach known as ML-KNN is proposed in [78], where the Bayes method is used in order to estimate the probability that a label belongs to a particular class. The broad approach used in this process is as follows. For each test instance, its k nearest neighbors are identified. The statistical information in the label sets of these neighborhood instances is then used to determine the label set. The maximum a-posteriori principle is applied to determine the label set of the test instance.

Let $\mathcal{Y}$ denote the set of labels for the multi-label classification problem, and let n be the total number of labels. For a given test instance T, the first step is to compute its k nearest neighbors. Once the k nearest neighbors have been computed, the number of occurrences $\overline{C} = (C(1) \ldots C(n))$ of each of the n different labels is computed. Let $E^1(T,i)$ be the event that the test instance T contains

the label i, and $E^0(T,i)$ be the event that the test instance T does not contain the label i. Then, in order to determine whether or not the label i is included in test instance T, the maximum posterior principle is used:

$$b = \text{argmax}_{b\in\{0,1\}}\{P(E^b(T,i)|\overline{C})\}. \tag{6.10}$$

In other words, we wish to maximize between the probability of the events of label i being included or not. Therefore, the Bayes rule can be used in order to obtain the following:

$$b = \text{argmax}_{b\in\{0,1\}}\left\{\frac{P(E^b(T,i))\cdot P(\overline{C}|E^b(T,i))}{P(\overline{C})}\right\}. \tag{6.11}$$

Since the value of $P(\overline{C})$ is independent of b, it is possible to remove it from the denominator, without affecting the maximum argument. This is a standard approach used in all Bayes methods. Therefore, the best matching label may be expressed as follows:

$$b = \text{argmax}_{b\in\{0,1\}}\left\{P(E^b(T,i))\cdot P(\overline{C}|E^b(T,i))\right\}. \tag{6.12}$$

The prior probability $P(E^b(T,i))$ can be estimated as the fraction of the labels belonging to a particular class. The value of $P(\overline{C}|E^b(T,i))$ can be estimated by using the naive Bayes rule.

$$P(\overline{C}|E^b(T,i)) = \prod_{j=1}^{n} P(C(j)|E^b(T,i)). \tag{6.13}$$

Each of the terms $P(C(j)|E^b(T,i))$ can be estimated in a data driven manner by examining among the instances satisfying the value b for class label i, the fraction that contains exactly the count $C(j)$ for the label j. Laplacian smoothing is also performed in order to avoid ill conditioned probabilities. Thus, the correlation between the labels is accounted for by the use of this approach, since each of the terms $P(C(j)|E^b(T,i))$ indirectly measures the correlation between the labels i and j.

This approach is often popularly understood in the literature as a nearest neighbor approach, and has therefore been discussed in the section on nearest neighbor methods. However, it is more similar to a local naive Bayes approach (discussed in Section 6.6) rather than a distance-based approach. This is because the statistical frequencies of the neighborhood labels are used for local Bayes modeling. Such an approach can also be used for the standard version of the classification problem (when each instance is associated with exactly one label) by using the statistical behavior of the neighborhood features (rather than label frequencies). This yields a lazy Bayes approach for classification [38]. However, the work in [38] also estimates the Bayes probabilities *locally* only over the neighborhood in a data driven manner. Thus, the approach in [38] sharpens the use of locality even further for classification. This is of course a tradeoff, depending upon the amount of training data available. If more training data is available, then local sharpening is likely to be effective. On the other hand, if less training data is available, then local sharpening is not advisable, because it will lead to difficulties in robust estimations of conditional probabilities from a small amount of data. This approach will be discussed in some detail in Section 6.6.

If desired, it is possible to combine the two methods discussed in [38] and [78] for multi-label learning in order to learn the information in both the features and labels. This can be done by using both feature *and* label frequencies for the modeling process, and the product of the label-based and feature-based Bayes probabilities may be used for classification. The extension to that case is straightforward, since it requires the multiplication of Bayes probabilities derived form two different methods. An experimental study of several variations of nearest neighbor algorithms for classification in the multi-label scenario is provided in [59].

6.4 Lazy SVM Classification

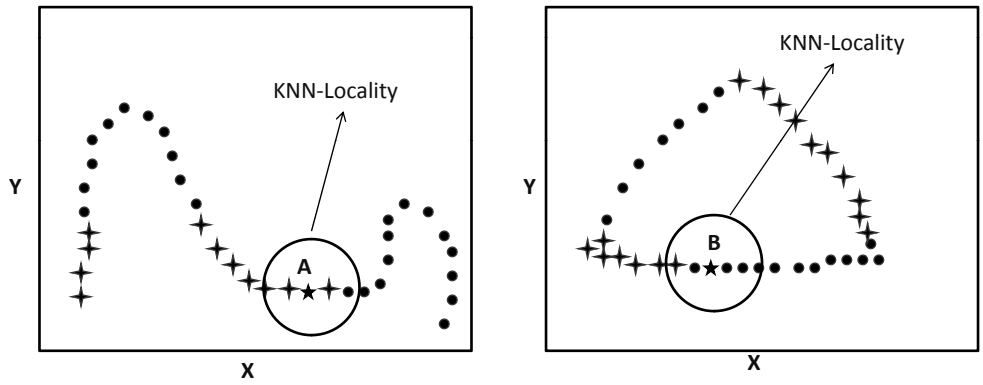
In an earlier section, it was shown that linear discriminant analysis can be very useful in adapting the behavior of the nearest neighbor metric for more effective classification. It should be pointed out that SVM classifiers are also linear models that provide directions discriminating between the different classes. Therefore, it is natural to explore the connections between SVM and nearest neighbor classification.

The work in [35] proposes methods for computing the relevance of each feature with the use of an SVM approach. The idea is to use a support vector machine in order to compute the gradient vector (local to test instance) of the rate change in class label in each direction, which is parallel to the axis. It has been shown in [35] that the support vector machine provides a natural way to estimate this gradient. The component along any direction provides the relevance R_j for that feature j. The idea is that when the class labels change at a significantly higher rate along a given direction j, then that feature should be weighted more in the distance metric. The actual weight w_j for the feature j is assumed to be proportional to $e^{\lambda \cdot R_j}$, where λ is a constant that determines the level of importance to be given to the different relevance values. A similar method for using SVM to modify the distance metric is proposed in [54], and the connections of the approach to linear discriminant analysis have also been shown in this work.

Note that the approaches in [35, 54] use a nearest neighbor classifier, and the SVM is only used to learn the distance function more accurately by setting the relevance weights. Therefore, the approach is still quite similar to that discussed in the last section. A completely different approach would be to use the nearest neighbors in order to isolate the locality around the test instance, and then *build an SVM that is optimized to that data locality*. For example, in the case of Figures 6.3(a) and (b) the data in the two classes is distributed in such a way that a global SVM cannot separate the classes very well. However, in the locality around the test instances A and B, it is easy to create a linear SVM that separates the two classes well. Such an approach has been proposed in [23, 77]. The main difference between the work in [23] and [77] is that the former is designed for the L_2-distances, whereas the latter can work for arbitrary and complex distance functions. Furthermore, the latter also proposes a number of optimizations for the purpose of efficiency. Therefore, we will describe the work in [77], since it is a bit more general. The specific steps that are used in this approach are as follows:

1. Determine the k-nearest neighbors of the test instance.
2. If all the neighbors have the same label, then that label is reported, otherwise all pairwise distances between the k-nearest neighbors are computed.
3. The distance matrix is converted into a kernel matrix using the kernel trick and a local SVM classifier is constructed on this matrix.
4. The test instance is classified with this local SVM classifier.

A number of optimizations such as caching have been proposed in [77] in order to improve the efficiency of the approach. Local SVM classifiers have been used quite successfully for a variety of applications such as spam filtering [20].



(a) Non-linear distribution (b) Piecewise linear

FIGURE 6.3: Advantages of instance-centered local linear modeling.

6.5 Locally Weighted Regression

Regression models are linear classifiers, which use a regression model in order to relate the attribute values to the class label. The key idea in locally weighted regression is that global regression planes are not well optimized to the local distributions in the data, as is evident from the illustrations in Figure 6.2. Correspondingly, it is desirable to create a test-instance specific local regression model. Such a method has been discussed in [15, 28]. The latter work [28] is also applicable to the multi-label case.

The core idea in locally weighted regression is to create a regression model that relates the feature values to the class label, as in all traditional regression models. The only difference is that the *k*-nearest neighbors are used for constructing the regression model. Note that this approach differs from the local discriminant-based models (which are also linear models), in that the votes among the *k* nearest neighbors are not used for classification purposes. Rather, a linear regression model is constructed in a lazy way, which is specific to the test instance. Several choices may arise in the construction of the model:

- The nature of the fit may be different for different kinds of data sets. For example, linear models may be more appropriate for some kinds of data, whereas quadratic models may be more appropriate for others.
- The choice of the error function may be different for different kinds of data. Two common error functions that are used are the least-squares error and the distance-weighted least squares error.

The primary advantage of locally weighted regression is when the data shows non-linearity in the underlying class distribution on a global scale, but can be approximated by linear models at the local scale. For example, in Figures 6.3(a) and (b) there are no linear functions that can approximate the discrimination between the two classes well, but it is easy to find local linear models that discriminate well within the locality of the test instances A and B, respectively. Since instance-based learning provides knowledge about the appropriate locality (by virtue of the laziness in the model construction process), it is possible to build a regression model that is well optimized to the corresponding data locality. An interesting local linear model with the use of the recursive least squares algorithm is proposed in [18]. In this approach, the number of nearest neighbors is also selected in a data-driven manner.

6.6 Lazy Naive Bayes

Lazy learning has also been extended to the case of the naive Bayes classifiers [38,79]. The work in [38] proposes a locally weighted naive Bayes classifier. The naive Bayes classifier is used in a similar way as the local SVM method [77], or the locally weighted regression method [15] discussed in previous sections. A local Bayes model is constructed with the use of a subset of the data in the neighborhood of the test instance. Furthermore, the training instances in this neighborhood are weighted, so that training instances that are closer to the test instance are weighted more in the model. This local model is then used for classification. The subset of data in the neighborhood of the test instance are determined by using the k-nearest neighbors of the test instance. In practice, the subset of k neighbors is selected by setting the weight of any instance beyond the k-th nearest neighbor to 0. Let D represent the distance of the test instance to the kth nearest neighbor, and d_i represent the distance of the test instance to the ith training instance. Then, the weight w_i of the ith instance is set to the following:

$$w_i = f(d_i/D). \tag{6.14}$$

The function $f(\cdot)$ is a monotonically non-increasing function, which is defined as follows:

$$f(y) = \max\{1-y, 0\}. \tag{6.15}$$

Therefore, the weight decreases linearly with the distance, but cannot decrease beyond 0, once the k-nearest neighbor has been reached. The naive Bayes method is applied in a standard way, *except that the instance weights are used in estimating all the probabilities for the Bayes classifier*. Higher values of k will result in models that do not fluctuate much with variations in the data, whereas very small values of k will result in models that fit the noise in the data. It has been shown in [38] that the approach is not too sensitive to the choice of k within a reasonably modest range of values of k. Other schemes have been proposed in the literature, which use the advantages of lazy learning in conjunction with the naive Bayes classifier. The work in [79] fuses a standard rule-based learner with naive Bayes models. A technique discussed in [74] lazily learns multiple Naive Bayes classifiers, and uses the classifier with the highest estimated accuracy in order to decide the label for the test instance.

The work discussed in [78] can also be viewed as an unweighted local Bayes classifier, where the weights used for all instances are 1, rather than the weighted approach discussed above. Furthermore, the work in [78] uses the frequencies of the *labels* for Bayes modeling, rather than the actual *features* themselves. The idea in [78] is that the other labels themselves serve as features, since the same instance may contain multiple labels, and sufficient correlation information is available for learning. This approach is discussed in detail in Section 6.3.7.

6.7 Lazy Decision Trees

The work in [40] proposes a method for constructing lazy decision trees, which are specific to a test instance, which is referred to as *LazyDT*. In practice, only a *path* needs to be constructed, because a test instance follows only one path along the decision tree, and other paths are irrelevant from the perspective of lazy learning. As discussed in a later section, such paths can also be interactively explored with the use of visual methods.

One of the major problems with decision trees is that they are *myopic*, since a split at a higher level of the tree may not be optimal for a specific test instance, and in cases where the data contains

many relevant features, only a small subset of them may be used for splitting. When a data set contains N data points, a decision tree is allowed only $O(\log(N))$ (approximately balanced) splits, and this may be too small in order to use the best set of features for a particular test instance. Clearly, the knowledge of the test instance allows the use of more relevant features for construction of the appropriate decision path at a higher level of the tree construction.

The *additional knowledge of the test instance* helps in the recursive construction of a path in which only relevant features are used. One method proposed in [40] is to use a split criterion, which successively reduces the size of the training associated with the test instance, until either all instances have the same class label, or the same set of features. In both cases, the majority class label is reported as the relevant class. In order to discard a set of irrelevant instances in a particular iteration, any standard decision tree split criterion is used, and only those training instances, that satisfy the predicate in the same way as the test instance will be selected for the next level of the decision path. The split criterion is decided using any standard decision tree methodology such as the normalized entropy or the gini-index. The main difference from the split process in the traditional decision tree is that *only the node containing the test instance is relevant in the split*, and the information gain or gini-index is computed on the basis of this node. One challenge with the use of such an approach is that the information gain in a single node can actually be negative if the original data is imbalanced. In order to avoid this problem, the training examples are re-weighted, so that the aggregate weight of each class is the same. It is also relatively easy to deal with missing attributes in test instances, since the split only needs to be performed on attributes that are present in the test instance. It has been shown [40] that such an approach yields better classification results, because of the additional knowledge associated with the test instance during the decision path construction process.

A particular observation here is noteworthy, since such decision paths can also be used to construct a robust any-time classifier, with the use of principles associated with a random forest approach [25]. It should be pointed out that a random forest translates to a random path created by a random set of splits from the instance-centered perspective. Therefore a natural way to implement the instance-centered random forest approach would be to discretize the data into ranges. A test instance will be relevant to exactly one range from each attribute. A random set of attributes is selected, and the intersection of the ranges provides one possible classification of the test instance. This approach can be repeated in order to provide a very efficient lazy ensemble, and the number of samples provides the tradeoff between running time and accuracy. Such an approach can be used in the context of an any-time approach in resource-constrained scenarios. It has been shown in [4] how such an approach can be used for efficient any-time classification of data streams.

6.8 Rule-Based Classification

A method for lazy rule-based classification has been proposed in [36], in which a unification has been proposed between instance-based methods and rule-based methods. This system is referred to as *Rule Induction from a Set of Exemplars*, or *RISE* for short. All rules contain at most one condition for each attribute on the left-hand side. In this approach, no distinction is assumed between instances and rules. An instance is simply treated as a rule in which all interval values on the left hand side of the rule are degenerate. In other words, an instance can be treated as a rule, by choosing appropriate conditions for the antecedent, and the class variable of the instance as the consequent. The conditions in the antecedent are determined by using the values in the corresponding instance. For example, if an attribute value for the instance for x_1 is numeric valued at 4, then the condition is assumed to be

$4 \leq x_i \leq 4$. If x_1 is symbolic and its value is a, then the corresponding condition in the antecedent is $x_1 = a$.

As in the case of instance-centered methods, a distance is defined between a test instance and a rule. Let $R = (A_1 \ldots A_m, C)$ be a rule with the m conditions $A_1 \ldots A_m$ in the antecedent, and the class C in the consequent. Let $\overline{X} = (x_1 \ldots x_d)$ be a d-dimensional example. Then, the distance $\Delta(\overline{X}, R)$ between the instance $\overline{X}$ and the rule R is defined as follows.

$$\Delta(\overline{X}, R) = \sum_{j=1}^{m} \delta(i)^s. \tag{6.16}$$

Here s is a real valued parameter such as 1,2,3, etc., and $\delta(i)$ represents the distance on the ith conditional. The value of $\delta(i)$ is equal to the distance of the instance to the nearest end of the range for the case of a numeric attribute and the value difference metric (VDM) of Equation 6.1 for the case of a symbolic attribute. This value of $\delta(i)$ is zero, if the corresponding attribute value is a match for the antecedent condition. The class label for a test instance is defined by the label of the nearest rule to the test instance. If two or more rules have the same accuracy, then the one with the greatest accuracy on the training data is used.

The set of rules in the RISE system are constructed as follows. RISE constructs good rules by using successive generalizations on the original set of instances in the data. Thus, the algorithm starts off with the training set of examples. RISE examines each rule one by one, and finds the nearest example of the same class that is not already covered by the rule. The rule is then generalized in order to cover this example, by expanding the corresponding antecedent condition. For the case of numeric attributes, the ranges of the attributes are increased minimally so as to include the new example, and for the case of symbolic attributes, a corresponding condition on the symbolic attribute is included. If the effect of this generalization on the global accuracy of the rule is non-negative, then the rule is retained. Otherwise, the generalization is not used and the original rule is retained. It should be pointed out that even when generalization does not improve accuracy, it is desirable to retain the more general rule because of the desirable bias towards simplicity of the model. The procedure is repeated until no rule can be generalized in a given iteration. It should be pointed out that some instances may not be generalized at all, and may remain in their original state in the rule set. In the worst case, no instance is generalized, and the resulting model is a nearest neighbor classifier.

A system called *DeEPs* has been proposed in [49], which combines the power of rules and lazy learning for classification purposes. This approach examines how the frequency of an instance's subset of features varies among the training classes. In other words, patterns that sharply differentiate between the different classes for a particular test instance are leveraged and used for classification. Thus, the specificity to the instance plays an important role in this discovery process. Another system, *HARMONY*, has been proposed in [66], which determines rules that are optimized to the different training instances. Strictly speaking, this is not a lazy learning approach, since the rules are optimized to training instances (rather than test instances) in a pre-processing phase. Nevertheless, the effectiveness of the approach relies on the same general principle, and it can also be generalized for lazy learning if required.

6.9 Radial Basis Function Networks: Leveraging Neural Networks for Instance-Based Learning

Radial-basis function networks (RBF) are designed in a similar way to regular nearest neighbor classifiers, except that a set of N *centers* are learned from the training data. In order to classify a test instance, a distance is computed from the test instance to each of these centers $x_1 \ldots x_N$, and a density

function is computed at the instance using these centers. The combination of functions computed from each of these centers is computed with the use of a neural network. Radial basis functions can be considered three-layer feed-forward networks, in which each hidden unit computes a function of the form:

$$f_i(x) = e^{-||x-x_i||^2/2\cdot\sigma_i^2}. \tag{6.17}$$

Here σ_i^2 represents the local variance at center x_i. Note that the function $f_i(x)$ has a very similar form to that commonly used in kernel density estimation. For ease in discussion, we assume that this is a binary classification problem, with labels drawn from $\{+1,-1\}$, though this general approach extends much further, even to the extent of regression modeling. The final function is a weighted combination of these values with weights c_i.

$$f^*(x) = \sum_{i=1}^{N} w_i \cdot f_i(x). \tag{6.18}$$

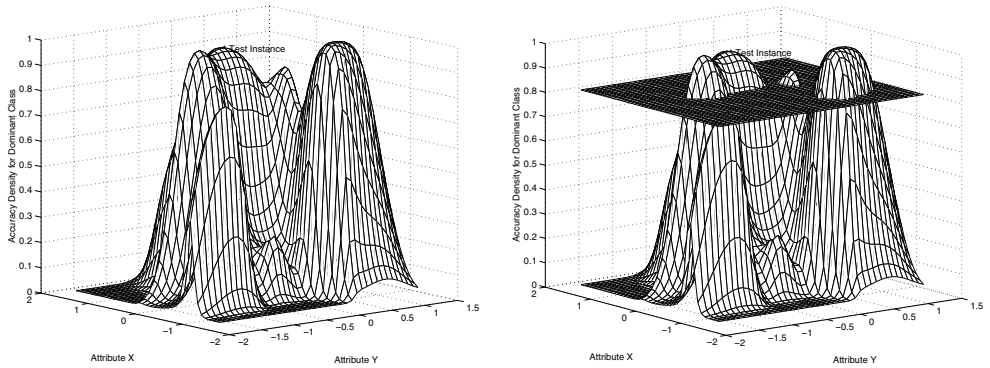
Here $x_1 \ldots x_N$ represent the N different centers, and w_i denotes the weight of center i, which is learned in the neural network. In classical instance-based methods, each data point x_i is an individual training instance, and the weight w_i is set to $+1$ or -1, depending upon its label. However, in RBF methods, the weights are *learned* with a neural network approach, since the centers are derived from the underlying training data, and do not have a label directly attached to them.

The N centers $x_1 \ldots x_N$ are typically constructed with the use of an unsupervised approach [19, 37, 46], though some recent methods also use supervised techniques for constructing the centers [71]. The unsupervised methods [19, 37, 46] typically use a clustering algorithm in order to generate the different centers. A smaller number of centers typically results in smaller complexity, and greater efficiency of the classification process. Radial-basis function networks are related to sigmoidal function networks (SGF), which have one unit for each instance in the training data. In this sense sigmoidal networks are somewhat closer to classical instance-based methods, since they do not have a first phase of cluster-based summarization. While radial-basis function networks are generally more efficient, the points at the different cluster boundaries may often be misclassified. It has been shown in [52] that RBF networks may sometimes require ten times as much training data as SGF in order to achieve the same level of accuracy. Some recent work [71] has shown how supervised methods even at the stage of center determination can significantly improve the accuracy of these classifiers. Radial-basis function networks can therefore be considered an evolution of nearest neighbor classifiers, where more sophisticated (clustering) methods are used for prototype selection (or re-construction in the form of cluster centers), and neural network methods are used for combining the density values obtained from each of the centers.

6.10 Lazy Methods for Diagnostic and Visual Classification

Instance-centered methods are particularly useful for diagnostic and visual classification, since the role of a diagnostic method is to find diagnostic characteristics that *are specific to that test-instance*. When an eager model is fully constructed as a pre-processing step, it is often not optimized to finding the best diagnostic characteristics that are specific to a particular test instance. Therefore, instance-centered methods are a natural approach in such scenarios. The *subspace decision path* method [6] can be considered a lazy and interactive version of the decision tree, which provides insights into the specific characteristics of a particular test instance.

Kernel density estimation is used in order to create a visual profile of the underlying data. It is assumed that the data set $\mathcal{D}$ contains N points and d dimensions. The set of points in $\mathcal{D}$ are denoted



(a) Accuracy Density Profile Isolation of Data

FIGURE 6.4: Density profiles and isolation of data.

by $X_1 \ldots X_N$. Let us further assume that the k classes in the data are denoted by $\mathcal{C}_1 \ldots \mathcal{C}_k$. The number of points belonging to the class $\mathcal{C}_i$ is n_i, so that $\sum_{i=1}^{k} n_i = N$. The data set associated with the class i is denoted by $\mathcal{D}_i$. This means that $\cup_{i=1}^{k} \mathcal{D}_i = \mathcal{D}$. The probability density at a given point is determined by the sum of the smoothed values of the kernel functions $K_h(\cdot)$ associated with each point in the data set. Thus, the density estimate of the data set $\mathcal{D}$ at the point x is defined as follows:

$$f(x, \mathcal{D}) = \frac{1}{n} \cdot \sum_{X_i \in \mathcal{D}} K_h(x - X_i). \tag{6.19}$$

The kernel function is a smooth unimodal distribution such as the Gaussian function:

$$K_h(x - X_i) = \frac{1}{\sqrt{2\pi} \cdot h} \cdot e^{-\frac{||x - X_i||^2}{2h^2}}. \tag{6.20}$$

The kernel function is dependent on the use of a parameter h, which is the level of smoothing. The accuracy of the density estimate depends upon this width h, and several heuristic rules have been proposed for estimating the bandwidth [6].

The value of the density $f(x, \mathcal{D})$ may differ considerably from $f(x, \mathcal{D}_i)$ because of the difference in distributions of the different classes. Correspondingly, the *accuracy density* $\mathcal{A}(x, \mathcal{C}_i, \mathcal{D})$ for the class $\mathcal{C}_i$ is defined as follows:

$$\mathcal{A}(x, \mathcal{C}_i, \mathcal{D}) = \frac{n_i \cdot f(x, \mathcal{D}_i)}{\sum_{i=1}^{k} n_i \cdot f(x, \mathcal{D}_i)}. \tag{6.21}$$

The above expression always lies between 0 and 1, and is simply an estimation of the Bayesian posterior probability of the test instance belonging to class $\mathcal{C}_i$. It is assumed that the a-priori probability of the test instance belonging to class $\mathcal{C}_i$, (without knowing any of its feature values) is simply equal to its fractional presence n_i/N in class $\mathcal{C}_i$. The conditional probability density after knowing the feature values x is equal to $f(x, \mathcal{D}_i)$. Then, by applying the Bayesian formula for posterior probabilities,the condition of Equation 6.21 is obtained. The higher the value of the accuracy density, the greater the relative density of $\mathcal{C}_i$ compared to the other classes.

Another related measure is the *interest density* $I(x, \mathcal{C}_i, \mathcal{D})$, which is the *ratio* of the density of the class $\mathcal{C}_i$ to the overall density of the data.

$$I(x, \mathcal{C}_i, \mathcal{D}) = \frac{f(x, \mathcal{D}_i)}{f(x, \mathcal{D})}. \tag{6.22}$$

The class $\mathcal{C}_i$ is over-represented at x, when the interest density is larger than one. The *dominant class* at the coordinate x is denoted by $\mathcal{CM}(x, \mathcal{D})$, and is equal to $\text{argmax}_{i \in \{1,\ldots k\}} I(x, \mathcal{C}_i, \mathcal{D})$. Correspondingly, the maximum interest density at x is denoted by $I\mathcal{M}(x, \mathcal{D}) = \max_{i \in \{1,\ldots k\}} I(x, \mathcal{C}_i, \mathcal{D})$. Both the interest and accuracy density are valuable quantifications of the level of dominance of the different classes. The interest density is more effective at comparing among the different classes at a given point, whereas the accuracy density is more effective at providing an idea of the absolute accuracy at a given point.

So far, it has been assumed that all of the above computations are performed in the full dimensional space. However, it is also possible to project the data onto the subspace E in order to perform this computation. Such a calculation would quantify the discriminatory power of the subspace E at x. In order to denote the use of the subspace E in any computation, the corresponding expression will be superscripted with E. Thus the density in a given subspace E is denoted by $f^E(\cdot, \cdot)$, the accuracy density by $\mathcal{A}^E(\cdot, \cdot, \cdot)$, and the interest density by $I^E(\cdot, \cdot, \cdot)$. Similarly, the dominant class is defined using the subspace-specific interest density at that point, and the accuracy density profile is defined for that particular subspace. An example of the accuracy density profile (of the dominant class) in a 2-dimensional subspace is illustrated in Figure 6.4(a). The test instance is also labeled in the same figure in order to illustrate the relationship between the density profile and test instance.

It is desired to find those projections of the data in which the interest density value $I\mathcal{M}^E(t, \mathcal{D})$ is the maximum. It is quite possible that in some cases, different subspaces may provide different information about the class behavior of the data; these are the difficult cases in which a test instance may be difficult to classify accurately. In such cases, the user may need to isolate particular data localities in which the class distribution is further examined by a hierarchical exploratory process. While the density values are naturally defined over the continuous space of quantitative attributes, it has been shown in [6] that intuitively analogous values can be defined for the interest and accuracy densities even when categorical attributes are present.

For a given test example, the end user is provided with unique options in exploring various characteristics that are indicative of its classification. A subspace determination process is used on the basis of the highest interest densities at a given test instance. Thus, the subspace determination process finds the appropriate *local* discriminative subspaces for a given test example. These are the various possibilities (or branches) of the decision path, which can be utilized in order to explore the regions in the locality of the test instance. In each of these subspaces, the user is provided with a visual profile of the accuracy density. This profile provides the user with an idea of which branch is likely to lead to a region of high accuracy for that test instance. This visual profile can also be utilized in order to determine which of the various branches are most suitable for further exploration. Once such a branch has been chosen, the user has the option to further explore into a particular region of the data that has high accuracy density. This process of data localization can quickly isolate an arbitrarily shaped region in the data containing the test instance. This sequence of data localizations creates a path (and a locally discriminatory combination of dimensions) that reveals the underlying classification causality to the user.

In the event that a decision path is chosen that is not strongly indicative of any class, the user has the option to backtrack to a higher level node and explore a different path of the tree. In some cases, different branches may be indicative of the test example belonging to different classes. These are the "ambiguous cases" in which a test example could share characteristics from multiple classes. Many standard modeling methods may classify such an example incorrectly, though the subspace decision path method is much more effective at providing the user with an intensional knowledge of the test example because of its exploratory approach. This can be used in order to understand the causality behind the ambiguous classification behavior of that instance.

The overall algorithm for decision path construction is illustrated in Figure 6.5. The details of the subroutines in the procedure are described in [6], though a summary description is provided here. The input to the system is the data set $\mathcal{D}$, the test instance t for which one wishes to find the

Algorithm *SubspaceDecisionPath(Test Example: t, Data Set: $\mathcal{D}$,*
MaxDimensionality: l, MaxBranchFactor: b_{max}, Minimum Interest Ratio: ir_{min})
begin
PATH= $\{\mathcal{D}\}$;
while *not(termination)* **do**
begin
Pick the last data set $\mathcal{L}$ indicated in PATH;
$\mathcal{E} = \{E_1 \ldots E_q\} =$ *ComputeClassificationSubspaces*($\mathcal{L}, t, l, b_{max}, ir_{min}$);
for each E_i *ConstructDensityProfile(E_i, $\mathcal{L}$, t)*;
{ " Zooming in" refers to further subspace exploration }
if (*zoom-in (user-specified)*) **then**
begin
User specifies choice of branch E_i;
User specifies accuracy density threshold λ for zoom-in;
{ $p(\mathcal{L}', \mathcal{C}_i)$ is accuracy significance of class
$\mathcal{C}_i$ in $\mathcal{L}'$ with respect to $\mathcal{L}$ }
$(\mathcal{L}', p(\mathcal{L}', \mathcal{C}_1) \ldots p(\mathcal{L}', \mathcal{C}_k)) =$ *IsolateData($\mathcal{L}, t, \lambda$)*;
Add $\mathcal{L}'$ to the end of PATH;
end;
else begin (retreat))
User specifies data set $\mathcal{L}'$ on PATH to backtrack to;
Delete all data set pointers occuring after $\mathcal{L}'$ on PATH; (backtrack)
end;
{ Calculate cumulative dominance of each class $\mathcal{C}_i$ along PATH
in order to provide the user a measure of its significance }
for each class $\mathcal{C}_i$ **do**
Output $CD(PATH, \mathcal{C}_i) = 1 - \pi_{(\mathcal{L} \in PATH, \mathcal{L} \neq \mathcal{D})}(1 - p(\mathcal{L}, \mathcal{C}_i))$ to user;
end;
end

FIGURE 6.5: The subspace decision path method.

diagnostic characteristics, a maximum branch factor b_{max}, and a minimum interest density ir_{min}. In addition, the maximum dimensionality l of any subspace utilized in data exploration is utilized as user input. The value of $l = 2$ is especially interesting because it allows for the use of visual profile of the accuracy density. Even though it is natural to use 2-dimensional projections because of their visual interpretability, the data exploration process along a given path reveals a higher dimensional combination of dimensions, which is most suitable for the test instance. The branch factor b_{max} is the maximum number of possibilities presented to the user, whereas the value of ir_{min} is the corresponding minimum interest density of the test instance in any subspace presented to the user. The value of ir_{min} is chosen to be 1, which is the break-even value for interest density computation. This break-even value is one at which the interest density neither under-estimates nor over-estimates the accuracy of the test instance with respect to a class. The variable PATH consists of the pointers to the sequence of successively reduced training data sets, which are obtained in the process of interactive decision tree construction. The list PATH is initialized to a single element, which is the pointer to the original data set $\mathcal{D}$. At each point in the decision path construction process, the subspaces $E_1 \ldots E_q$ are determined, which have the greatest interest density (of the dominant class) in the locality of the test instance t. This process is accomplished by the procedure *ComputeClassificationSubspaces*. Once these subspaces have been determined, the density profile is constructed for each of them by

the procedure *ConstructDensityProfile*. Even though one subspace may have higher interest density at the test instance than another, the true value of a subspace in separating the data locality around the test instance is often a subjective judgement that depends both upon the interest density of the test instance and the spatial separation of the classes. Such a judgement requires human intuition, which can be harnessed with the use of the visual profile of the accuracy density profiles of the various possibilities. These profiles provide the user with an intuitive idea of the class behavior of the data set in various projections. If the class behavior across different projections is not very consistent (different projections are indicative of different classes), then such a node is not very revealing of valuable information. In such a case, the user may choose to backtrack by specifying an earlier node on PATH from which to start further exploration.

On the other hand, if the different projections provide a consistent idea of the class behavior, then the user utilizes the density profile in order to isolate a small region of the data in which the accuracy density of the data in the locality of the test instance is significantly higher for a particular class. This is achieved by the procedure *IsolateData*. This isolated region may be a cluster of arbitrary shape depending upon the region covered by the dominating class. However, the use of the visual profile helps to maintain the interpretability of the isolation process in spite of the arbitrary contour of separation. An example of such an isolation is illustrated in Figure 6.4(b), and is facilitated by the construction of the visual density profile. The procedure returns the isolated data set $\mathcal{L}'$ along with a number of called the *accuracy significance* $p(\mathcal{L}', \mathcal{C}_i)$ of the class $\mathcal{C}_i$. The pointer to this new data set $\mathcal{L}'$ is added to the end of PATH. At that point, the user decides whether further exploration into that isolated data set is necessary. If so, the same process of subspace analysis is repeated on this node. Otherwise, the process is terminated and the most relevant class label is reported. The overall exploration process also provides the user with a good diagnostic idea of how the local data distributions along different dimensions relate to the class label.

6.11 Conclusions and Summary

In this chapter, we presented algorithms for instance-based learning and lazy-learning. These methods are usually computationally more expensive than traditional models, but have the advantage of using information about the test instance at the time of model construction. In many scenarios, such methods can provide more accurate results, when a pre-processed model cannot fully capture the underlying complexities of the data. The other advantage of instance-based methods is that they are typically very simple and easy to implement for arbitrary data types, because the complexity is usually abstracted in determining an appropriate distance function between objects. Since distance function design has been widely studied in the literature for different data types, this allows the use of this methodology for arbitrary data types. Furthermore, existing model-based algorithms can be generalized to local variations in order to transform them to instance-learning methods. Such methods have great diagnostic value for specific test instances, since the model is constructed local to the specific test instance, and also provides knowledge to the analyst that is test-instance specific. Lazy methods are highly flexible and can be generalized to streaming and other temporal scenarios, where the laziness of the approach can be used in order to learn the appropriate temporal horizon, in which lazy learning should be performed.

Bibliography

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

[2] C. Aggarwal, A. Hinneburg, and D. Keim. On the surprising behavior of distance metrics in high dimensional space, *Data Theory—ICDT Conference*, 2001; *Lecture Notes in Computer Science*, 1973:420–434, 2001.

[3] C. Aggarwal. Towards systematic design of distance functions for data mining applications, *Proceedings of the KDD Conference*, pages 9–18, 2003.

[4] C. Aggarwal and P. Yu. Locust: An online analytical processing framework for high dimensional classification of data streams, *Proceedings of the IEEE International Conference on Data Engineering*, pages 426–435, 2008.

[5] C. Aggarwal and C. Reddy. *Data Clustering: Algorithms and Applications*, CRC Press, 2013.

[6] C. Aggarwal. Toward exploratory test-instance-centered diagnosis in high-dimensional classification, *IEEE Transactions on Knowledge and Data Engineering*, 19(8):1001–1015, 2007.

[7] C. Aggarwal, J. Han, J. Wang, and P. Yu. A framework for on demand classification of evolving data streams. In *IEEE TKDE Journal*, 18(5):577–589, 2006.

[8] D. Aha, P. Clark, S. Salzberg, and G. Blix. Incremental constructive induction: An instance-based approach, *Proceedings of the Eighth International Workshop on Machine Learning*, pages 117–121, 1991.

[9] D. Aha. *Lazy learning*. Kluwer Academic Publishers, 1997.

[10] D. Aha. Tolerating noisy, irrelevant and novel attributes in instance-based learning algorithms, *International Journal of Man-Machine Studies*, 36(2):267–287, 1992.

[11] D. Aha, D. Kibler, and M. Albert. Instance-based learning algorithms, *Machine Learning*, 6(1):37–66, 1991.

[12] D. Aha. Lazy learning: Special issue editorial, *Artificial Intelligence Review*, 11:7–10, 1997.

[13] F. Angiulli. Fast nearest neighbor condensation for large data sets classification, *IEEE Transactions on Knowledge and Data Engineering*, 19(11):1450–1464, 2007.

[14] M. Ankerst, G. Kastenmuller, H.-P. Kriegel, and T. Seidl. Nearest Neighbor Classification in 3D Protein Databases, *ISMB-99 Proceedings*, pages 34–43, 1999.

[15] C. Atkeson, A. Moore, and S. Schaal. Locally weighted learning, *Artificial Intelligence Review*, 11(1–5):11–73, 1997.

[16] S. D. Bay. Combining nearest neighbor classifiers through multiple feature subsets. *Proceedings of ICML Conference*, pages 37–45, 1998.

[17] J. Beringer and E Hullermeier. Efficient instance-based learning on data streams, *Intelligent Data Analysis*, 11(6):627–650, 2007.

[18] M. Birattari, G. Bontempi, and M. Bersini. Lazy learning meets the recursive least squares algorithm, *Advances in neural information processing systems*, 11:375–381, 1999.

[19] C. M. Bishop. Improving the generalization properties of radial basis function neural networks, *Neural Computation*, 3(4):579–588, 1991.

[20] E. Blanzieri and A. Bryl. Instance-based spam filtering using SVM nearest neighbor classifier, *FLAIRS Conference*, pages 441–442, 2007.

[21] J. Blitzer, K. Weinberger, and L. Saul. Distance metric learning for large margin nearest neighbor classification, *Advances in neural information processing systems*: 1473–1480, 2005.

[22] S. Boriah, V. Chandola, and V. Kumar. Similarity measures for categorical data: A comparative evaluation. *Proceedings of the SIAM Conference on Data Mining*, pages 243–254, 2008.

[23] L. Bottou and V. Vapnik. Local learning algorithms, *Neural COmputation*, 4(6):888–900, 1992.

[24] L. Breiman, J. Friedman, R. Olshen, and C. Stone. *Classification and Regression Trees*, Wadsworth, 1984.

[25] L. Breiman. Random forests, *Machine Learning*, 48(1):5–32, 2001.

[26] T. Cain, M. Pazzani, and G. Silverstein. Using domain knowledge to influence similarity judgements, *Proceedings of the Case-Based Reasoning Workshop*, pages 191–198, 1991.

[27] C. L. Chang. Finding prototypes for nearest neighbor classifiers: *IEEE Transactions on Computers*, 100(11):1179–184, 1974.

[28] W. Cheng and E. Hullermeier. Combining instance-based learning and logistic regression for multilabel classification, *Machine Learning*, 76 (2–3):211–225, 2009.

[29] J. G. Cleary and L. E. Trigg. K^*: An instance-based learner using an entropic distance measure, *Proceedings of ICML Conference*, pages 108–114, 1995.

[30] S. Cost and S. Salzberg. A weighted nearest neighbor algorithm for learning with symbolic features, *Machine Learning*, 10(1):57–78, 1993.

[31] T. Cover and P. Hart. Nearest neighbor pattern classification, *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[32] P. Cunningham, A taxonomy of similarity mechanisms for case-based reasoning, *IEEE Transactuions on Knowledge and Data Engineering*, 21(11):1532–1543, 2009.

[33] B. V. Dasarathy. *Nearest Neighbor (NN) Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press, 1990.

[34] C. Domeniconi, J. Peng, and D. Gunopulos. Locally adaptive metric nearest-neighbor classification, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(9):1281–1285, 2002.

[35] C. Domeniconi and D. Gunopulos. Adaptive nearest neighbor classification using support vector machines, *Advances in Neural Information Processing Systems*, 1: 665–672, 2002.

[36] P. Domingos. Unifying instance-based and rule-based induction, *Machine Learning*, 24(2):141–168, 1996.

[37] G. W. Flake. Square unit augmented, radially extended, multilayer perceptrons, *In Neural Networks: Tricks of the Trade*, pages 145–163, 1998.

[38] E. Frank, M. Hall, and B. Pfahringer. Locally weighted Naive Bayes, *Proceedings of the Nineteenth Conference on Uncertainty in Artificial Intelligence*, pages 249–256, 2002.

[39] J. Friedman. *Flexible Nearest Neighbor Classification*, Technical Report, Stanford University, 1994.

[40] J. Friedman, R. Kohavi, and Y. Yun. Lazy decision trees, *Proceedings of the National Conference on Artificial Intelligence*, pages 717–724, 1996.

[41] S. Garcia, J. Derrac, J. Cano, and F. Herrera. Prototype selection for nearest neighbor classification: Taxonomy and empirical study, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(3):417–436, 2012.

[42] D. Gunopulos and G. Das. Time series similarity measures. In *Tutorial notes of the sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 243–307, 2000.

[43] E. Han, G. Karypis, and V. Kumar. Text categorization using weight adjusted k-nearest neighbor classification, *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 53–65, 2001.

[44] T. Hastie and R. Tibshirani. Discriminant adaptive nearest neighbor classification, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(6):607–616, 1996.

[45] A. Hinneburg, C. Aggarwal, and D. Keim. What is the nearest neighbor in high dimensional spaces? *Proceedings of VLDB Conference*, pages 506–515, 2000.

[46] Y. S. Hwang and S. Y. Bang. An efficient method to construct a radial basis function neural network classifier, *Neural Networks*, 10(8):1495–1503, 1997.

[47] N. Jankowski and M. Grochowski, Comparison of instance selection algorithms, I algorithms survey, *Lecture Notes in Computer Science*, 3070:598–603, 2004.

[48] R. Kohavi, P. Langley, and Y. Yun. The utility of feature weighting in nearest-neighbor algorithms, *Proceedings of the Ninth European Conference on Machine Learning*, pages 85–92, 1997.

[49] J. Li, G. Dong, K. Ramamohanarao, and L. Wong. Deeps: A new instance-based lazy discovery and classification system, *Machine Learning*, 54(2):99–124, 2004.

[50] D. G. Lowe. Similarity metric learning for a variable-kernel classifier, *Neural computation*, 7(1):72–85, 1995.

[51] G. J. McLachlan. *Discriminant analysis and statistical pattern recognition*, Wiley-Interscience, Vol. 544, 2004.

[52] J. Moody and C. Darken. Fast learning in networks of locally-tuned processing units, *Neural computation*, 1(2), 281–294, 1989.

[53] J. Peng, B. Bhanu, and S. Qing. Probabilistic feature relevance learning for content-based image retrieval, *Computer Vision and Image Understanding*, 75(1):150–164, 1999.

[54] J. Peng, D. Heisterkamp, and H. Dai. LDA/SVM driven nearest neighbor classification, *Computer Vision and Pattern Recognition Conference*, 1–58, 2001.

[55] J. R. Quinlan. Combining instance-based and model-based learning. *Proceedings of ICML Conference*, pages 236–243, 1993.

[56] G. Salton and M. J. McGill. *Introduction to modern information retrieval.* McGraw-Hill, 1983.

[57] H. Samet. *Foundations of multidimensional and metric data structures*, Morgan Kaufmann, 2006.

[58] M. Sonka, H. Vaclav, and R. Boyle. *Image Processing, Analysis, and Machine Vision.* Thomson Learning, 1999.

[59] E. Spyromitros, G. Tsoumakas, and I. Vlahavas. An empirical study of lazy multilabel classification algorithms, In *Artificial Intelligence: Theories, Models and Applications*, pages 401–406, 2008.

[60] C. Stanfil and D. Waltz. Toward memory-based reasoning, *Communications of the ACM*, 29(12):1213–1228, 1986.

[61] M. Tahir, A. Bouridane, and F. Kurugollu. Simultaneous feature selection and feature weighting using Hybrid Tabu Search in K-nearest neighbor classifier, *Pattern Recognition Letters*, 28(4):483–446, 2007.

[62] P. Utgoff. Incremental induction of decision trees, *Machine Learning*, 4(2):161–186, 1989.

[63] P. Vincent and Y. Bengio. K-local hyperplane and convex distance nearest algorithms, *Neural Information Processing Systems*, pages 985–992, 2001.

[64] D. Volper and S. Hampson. Learning and using specific instances, *Biological Cybernetics*, 57(1–2):57–71, 1987.

[65] J. Wang, P. Neskovic, and L. Cooper. Improving nearest neighbor rule with a simple adaptive distance measure, *Pattern Recognition Letters*, 28(2):207–213, 2007.

[66] J. Wang and G. Karypis. On mining instance-centric classification rules, *IEEE Transactions on Knowledge and Data Engineering*, 18(11):1497–1511, 2006.

[67] I. Watson and F. Marir. Case-based reasoning: A review, *Knowledge Engineering Review*, 9(4):327–354, 1994.

[68] K. Weinberger, J. Blitzer, and L. Saul. Distance metric learning for large margin nearest neighbor classification, *NIPS Conference*, MIT Press, 2006.

[69] D. Wettschereck, D. Aha, and T. Mohri. A review and empirical evaluation of feature weighting methods for a class of lazy learning algorithms, *Artificial Intelligence Review*, 11(1–5):273–314, 1997.

[70] D. Wettschereck and D. Aha. Weighting features. *Case-Based Reasoning Research and Development*, pp. 347–358, Springer Berlin, Heidelberg, 1995.

[71] D. Wettschereck and T. Dietterich. Improving the performance of radial basis function networks by learning center locations, *NIPS*, Vol. 4:1133–1140, 1991.

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

[73] D. Wilson and T. Martinez. An integrated instance-based learning algorithm, *Computers and Intelligence*, 16(1):28–48, 2000.

[74] Z. Xie, W. Hsu, Z. Liu, and M. L. Lee. Snnb: A selective neighborhood based naive Bayes for lazy learning, *Advances in Knowledge Discovery and Data Mining*, pages 104–114, Springer, 2002.

[75] L. Yang. Distance Metric Learning: A Comprehensive Survey, 2006. `http://www.cs.cmu.edu/~liuy/frame_survey_v2.pdf`

[76] D.-C. Zhan, M. Li, Y.-F. Li, and Z.-H. Zhou. Learning instance specific distances using metric propagation, *Proceedings of ICML Conference*, pages 1225–1232, 2009.

[77] H. Zhang, A. Berg, M. Maire, and J. Malik. SVM-KNN: Discriminative nearest neighbor classification for visual category recognition, *Computer Vision and Pattern Recognition*, pages 2126–2136, 2006.

[78] M. Zhang and Z. H. Zhou. ML-kNN: A lazy learning approach to multi-label learning, *Pattern Recognition*, 40(7): 2038–2045, 2007.

[79] Z. Zheng and G. Webb. Lazy learning of Bayesian rules. *Machine Learning*, 41(1):53–84, 2000.

[80] Z. H. Zhou and Y. Yu. Ensembling local learners through multimodal perturbation, *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 35(4):725–735, 2005.

[81] X. Zhu and X. Wu. Scalable representative instance selection and ranking, *Proceedings of IEEE International Conference on Pattern Recognition*, Vol 3:352–355, 2006.