Chapter 2

Background

This chapter presents some background for the research presented in this thesis. We start with a review, in Sections 2.1 and 2.2, of example-based classification and regression methods, which provides an important context for similarity learning. In Section 2.3 we discuss prior work on learning distances and similarities, and in Section 2.4 we review state-of-the-art algorithms for similarity-based retrieval. The background for vision applications in the thesis is not covered in this chapter, but rather presented in Chapters 4, 5 and 6.

2.1 Example-based classification

In a classification problem, the labels belong to a finite set of possible *identities*:

$$\mathcal{Y} \equiv \{1, \ldots, C\},$$

and the task consists of assigning a test example to one of the C classes. By far the most used example-based method is the K nearest neighbors (K-NN) classifier. Its operation is described in Algorithm 1. Setting $K = 1$ yields the nearest neighbor classification rule, perhaps the simplest and the most widely used in practice.

2.1.1 Properties of KNN classifiers

Despite its simplicity, the NN classifier very often achieves good performance, particularly for large data sets. The result by Cover and Hart [26] establishes a tight upper bound on the *asymptotic risk* R_∞ of the NN rule for C classes in terms of the Bayes risk R^*,

$$R_\infty \leq R^* \left(2 - \frac{C}{C-1} R^*\right). \tag{2.1}$$

Similar bounds can be established for the K-NN classifier, although they are more involved (see, e.g., [38].)

The bound in (2.1) describes the performance of the rule in the limit on an infinite amount of data, and has practical significance only in conjunction with a reasonable

Algorithm 1 Classification with K nearest neighbors (KNN).

Given: Training set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ with labels $\{y_1, \ldots, y_N\}$.
Given: A distance measure $\mathcal{D} : \mathcal{X} \to \mathbb{R}$.
Given: An integer $0 < K \leq N$.
Given: Test example $\mathbf{x}_0 \in \mathcal{X}$.
Output: Predicted label $\hat{y}_0^{(KNN)}$.

1: Let $i_1^*, \ldots, i_K^*$ be the indices of the K NN of $\mathbf{x}_0$ in X w.r.t. $\mathcal{D}$, i.e.

$$\mathcal{D}(\mathbf{x}_0, \mathbf{x}_{i_1^*}) \leq \ldots \leq \mathcal{D}(\mathbf{x}_0, \mathbf{x}_{i_K^*})$$

and

$$\mathcal{D}(\mathbf{x}_0, \mathbf{x}_{i_K^*}) \leq \mathcal{D}(\mathbf{x}_0, \mathbf{x}_i) \text{ for all } i \notin \{i_1^*, \ldots, i_K^*\}.$$

2: For each $y \in \mathcal{Y}$ let $X'_y = \{i_k^* \mid y_{i_k^*} = y,\ 1 \leq k \leq K\}$
3: Predict $\hat{y}_0^{(KNN)} = \text{argmax}_{y \in \mathcal{Y}} |X'_y|$, breaking ties randomly.

rate of convergence of the N-sample risk R_N to R_∞, as N grows large. The finite sample behavior of the NN rule has been extensively studied and shown to be difficult to characterize in a general form. Under various assumptions and approximations, the existing results describe the rates of convergence of R_N to R_∞ [25, 37]; unfortunately, it has been shown that such convergence may be arbitrarily slow. Some results exist regarding the bounds on R_N [48, 90], and means of calculating the risk for given data and distribution parameters [90, 109]. In addition, some analysis of the deviation $R_N - R_\infty$ is given in [50].

Despite the lack of guarantees for finite samples, the NN rule has been known to work well in very many practical cases, and often performs on par with much more sophisticated classifiers, provided enough training data (see, for instance, [30] for an extensive comparative study).

A major drawback of the NN rule is its computational complexity. With N examples in a D-dimensional input space, brute-force exhaustive search requires $O(DN)$ operations (assuming a single distance calculation costs $O(D)$ operations, and must be carried out for each reference point). Faster algorithms, which require as little as $O(\log N)$ time, also require $O\left(N^{D/2}\right)$ storage which for high D is prohibitively large. It is a common conjecture, supported by empirical evidence [18], that exact NN search algorithms are bound to suffer from this problem, due to the "curse of dimensionality". To alleviate this problem, practitioners often turn to approximate schemes, which trade off some of the accuracy guarantees in retrieval of neighbors for a significant increase in speed.

2.1.2 Approximate nearest neighbors

An ϵ-k-th NN of $\mathbf{x}_0$ is defined as a training example $\mathbf{x}_{i_k^\epsilon}$ such that

$$\mathcal{D}(\mathbf{x}_0, \mathbf{x}_{i_k^\epsilon}) \leq (1 + \epsilon)\mathcal{D}(\mathbf{x}_0, \mathbf{x}_{i_k^*}), \tag{2.2}$$

where $\epsilon > 0$ is an approximation parameter. In general, there will be more than one such *candidate* point and the particular selection strategy depends on the specific search algorithm used.

The asymptotic risk of the ϵ-NN rule can be easily established.[1] By the dominated convergence theorem [], we have

$$\begin{aligned} \text{if} \quad & \lim_{N\to\infty} \mathcal{D}(\mathbf{x}_0, \mathbf{x}_{i_k^*}) = 0 \quad \text{w.p.1}, & (2.3) \\ \text{then} \quad & \lim_{N\to\infty} \mathcal{D}(\mathbf{x}_0, \mathbf{x}_{i_k^\epsilon}) = 0 \quad \text{w.p.1}, & (2.4) \end{aligned}$$

following from (2.2), where $\mathcal{D}$ is the metric in the input space. The limit in (2.3) is proved, under mild assumptions, in [26], thus yielding the conclusion in (2.4). From here, one can closely follow the proof in [26] and obtain the R_∞ (i.e., the same asymptotic overall risk as for the exact NN rule) in the limit. As for the finite risk of the ϵ-NN classifier, and in particular its deviation from the corresponding risk of the exact NN, no definitive answers are known yet.

State-of-the-art methods allow finding an ϵ-NN or ϵ-R neighbors (see next section for definition) in time sublinear in N, and with mild storage requirements. In section 2.4.2 we describe in detail one such method: the locality sensitive hashing (LSH).

2.1.3 Near versus nearest

In the algorithms discussed above, the cutoff used in the search procedure is parametrized by the rank order K. An alternative criterion is to use a distance cutoff. Modifying step 1 accordingly leads to the R-near neighbor classifier (Algorithm 2.) The notion of approximate near neighbor is defined similarly to (2.2): For a given distance value R and the approximation factor ϵ, the ϵ-R neighbor of $\mathbf{x}_0$ is defined as any $\mathbf{x}$ for which

$$\mathcal{D}(\mathbf{x}_0, \mathbf{x}_{\epsilon,R}) \;\leq\; (1+\epsilon)R. \tag{2.5}$$

Algorithm 2 Classification with R-neighbors.

Given: Training set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ with labels $\{y_1, \ldots, y_N\}$.
Given: A distance measure $\mathcal{D} : \mathcal{X} \to \mathbb{R}$.
Given: A number $R > 0$.
Given: Test example $\mathbf{x}_0 \in \mathcal{X}$.
Output: Predicted label $\hat{y}_0^{(RN)}$.

1: Let $i_1^*, \ldots, i_K^*$ be the indices of R-neighbors of $\mathbf{x}_0$ in X w.r.t. $\mathcal{D}$, i.e.

$$\mathcal{D}(\mathbf{x}_0, \mathbf{x}_{i_k^*}) \;<\; R \qquad \text{for } k = 1, \ldots, K.$$

2: For each $y \in \mathcal{Y}$ let $X'_y = \{i_k^* \mid y_{i_k^*} = y,\ 1 \leq k \leq K\}$
3: Predict $\hat{y}_0^{(RN)} = \text{argmax}_{y\in\mathcal{Y}} |X'_y|$, breaking ties randomly.

[1] We are not aware of any previous publication of this observation.

There are important differences between the two algorithms. On the one hand, the search in K-NN is guaranteed to produce exactly K matches while for a fixed R the search in R-neighbor may fail to return any matches even with exact search algorithms, since there may simply be no appropriate matches in the database. Theoretical analysis of example-based methods based on near-neighbor retrieval appears to be harder, in particular it is difficult to show any "distribution-free" properties. On the other hand, setting the cutoff for Algorithm 2 may lead to more robust estimation, since it prevents cases in which some of the K-NN are too far to usefully contribute to the local model.[2] Overall, the choice of the specific formulation of neighborhood retrieval is a matter of design and should be decided for the task at hand. In most of the experiments in this thesis we used the K-NN (i.e., nearest) formulation.

2.1.4 Evaluation of retrieval accuracy

How good is a model $\widehat{\mathcal{S}}$? Since $\mathcal{S}$ defines a classification problem, a standard measure of accuracy for a classifier $\widehat{\mathcal{S}}$ is the *risk*

$$\mathcal{R}(\widehat{\mathcal{S}}, \mathcal{S}) = E_{(\mathbf{x},\mathbf{y})\in\mathcal{X}^2}\left[L\left(\widehat{\mathcal{S}}(\mathbf{x},\mathbf{y}), \mathcal{S}(\mathbf{x},\mathbf{y})\right)\right]$$

which depends on the *loss matrix* L that specifies the penalty for any combination of true and predicted similarity values. In practice, the expectation above can be estimated by calculating the average loss on a finite test set.

A more detailed measure is the combination of the *precision* of $\widehat{\mathcal{S}}$

$$\text{pre} \;=\; \frac{|\{(\mathbf{x},\mathbf{y}) :\; \mathcal{S}(\mathbf{x},\mathbf{y}) = +1 \text{ and } \widehat{\mathcal{S}}(\mathbf{x},\mathbf{y}) = +1\}|}{|\{(\mathbf{x},\mathbf{y}) :\; \widehat{\mathcal{S}}(\mathbf{x},\mathbf{y}) = +1\}|} \tag{2.6}$$

(i.e., out of pairs judged similar by $\widehat{\mathcal{S}}$, how many are really similar under $\mathcal{S}$), and its *recall*

$$\text{rec} \;=\; \frac{|\{(\mathbf{x},\mathbf{y}) :\; \mathcal{S}(\mathbf{x},\mathbf{y}) = +1 \text{ and } \widehat{\mathcal{S}}(\mathbf{x},\mathbf{y}) = +1\}|}{|\{(\mathbf{x},\mathbf{y}) :\; \mathcal{S}(\mathbf{x},\mathbf{y}) = +1\}|} \tag{2.7}$$

(out of pairs similar under $\mathcal{S}$, how many are correctly judged similar by $\widehat{\mathcal{S}}$.) A closely related terminology[3] refers to the *true positive*, or *detection* rate TP which is equal to the recall rate, and the *false positive*, or *false alarm* rate

$$\text{FP} \;=\; \frac{|\{(\mathbf{x},\mathbf{y}) :\; \mathcal{S}(\mathbf{x},\mathbf{y}) = -1 \text{ and } \widehat{\mathcal{S}}(\mathbf{x},\mathbf{y}) = +1\}|}{|\{(\mathbf{x},\mathbf{y}) :\; \mathcal{S}(\mathbf{x},\mathbf{y}) = -1\}|}. \tag{2.8}$$

Rather than specifying a single point in the precision/recall space, one can consider the entire range of the tradeoff between the two. Plotting the TP against FP as a function of changing parameters of the retrieval algorithm (in this case the threshold

[2] Although such spurious neighbors may be seen as outliers and could be dealt with by, say, robust local regression (Section 2.2.)

[3] This terminology corresponds to the view of similarity learning as a detection task.

on the distance) yields the receiving-operating characteristic (ROC) curve.

When retrieval of similar examples is the goal in itself, the ROC curve provides the comprehensive measure of the method's performance by specifying the range of the trade-off between precision and recall. Furthermore, the area under ROC curve (AUC) provides a single number describing the performance. However, if a similarity model is used as a component in an example-based classification or regression, its success should be also measured by the accuracy of the resulting prediction: the average classification error, or the mean estimation error on a test set.

The choice of a loss matrix L as well as the desired precision/recall combination is typically influenced by the relative frequency of similar and dissimilar pairs in $\mathcal{X}^2$. It is often the case (and especially so in vision-related domains) that similarity is a "rare event": two randomly selected examples from $\mathcal{X}$ are much less likely to be similar than not. This asymmetry has consequences on all aspects of similarity learning. For instance, L may need to be skewed significantly, in the sense that the penalty for one "direction" of wrong prediction is much higher than the penalty for the opposite error. For many learning algorithms this poses a significant challenge. However, we will describe in Chapter 3 how it can be turned to our advantage.

2.2 Example-based regression

The task of regression consists of predicting, for a query point $\mathbf{x}_0$, the value of a real-valued *target function* g on a test point $\mathbf{x}_0$; the function is conveyed to the learned by a set of examples $\mathbf{x}_1, \ldots, \mathbf{x}_N$ labeled by the value of $y_i = g(\mathbf{x}_i)$, perhaps with some noise. When no global parametric model of g is available, example-based estimation is often an appropriate choice.

The simplest example-based approach to regression is to apply the K-NN rule [24], with the slight modification to account for the estimation goal: the predicted value of y_0 is set to the *average* of the values in the NN, rather then to the winner of a majority vote[4]. This estimation rule corresponds to a piecewise-constant approximation of the target function $g(\mathbf{x})$, with at most $\binom{N}{K}$ distinct values (one value for every feasible assignment of K nearest neighbors among the reference points). Similarly to the K-NN classifier, there are results on the asymptotic behavior of this estimator [24].

The family of *local polynomial regression* estimators [43] may provide more flexibility in modeling the higher order behavior of the target function g. Under the assumptions that g is well approximated by a polynomial of degree p within any small region of $\mathcal{X}$, and that the selected neighbors of the query point $\mathbf{x}_0$ fall within such a small region around it, a polynomial model fit to those neighbors and evaluated in $\mathbf{x}_0$ will produce an accurate estimate of $g(\mathbf{x}_0)$.

A more robust approach using the local modeling idea is to assign weights to the neighbors, in accordance with their similarity to the query $\mathbf{x}_0$. The closer $\mathbf{x}_{i_i^*}$ is to $\mathbf{x}_0$ the more influence should it exert on the $\hat{y}_0$. This leads to the *locally weighted regression* (LWR) idea, an excellent introduction to which is available in [7].

[4] If $\mathcal{Y} \in \mathbb{R}^d$, with probability 1 every value will appear exactly once.

In the presence of noise, the local regression procedure may still be vulnerable to the misleading influence of outliers introduced by function mismeasurement, labeling errors or spurious similarity judgments. The *robust LWR* [21, 22] addresses this by re-weighting the neighbors based on the residuals of the fitted model and re-fitting the model with the new weights. This process is repeated for a small number of iterations, and may considerably improve results on noisy data.

The regression approach outlined above is applicable when the underlying relationship between the data and the estimated quantity g is a function, that is, when specifying the $\mathbf{x}$ determines a *unique* value of $g(\mathbf{x})$. In some applications this may not be the case: multiple values of g correspond to the same value of $\mathbf{x}$. In other words, there is an *ambiguity* in $g(\mathbf{x})$. This of course makes the problem of estimating g ill-posed. There are two possible avenues for addressing this challenge. One focuses on representation: finding a data space $\mathcal{X}$ in which the ambiguity is resolved. For instance, using multiple silhouettes (Section 5.2) or stereo-based disparity images (Section 5.3) largely removes the ambiguity in the pose estimation context.

The other avenue is to address the problem at the estimation step. If the representation at hand does lead to ambiguity, simply ignoring it may cause severe estimation errors–for instance, in a K-NN regression, if there are two possible values of $g(\mathbf{x})$ and the labels of the retrieved neighbors are roughly equally distributed among these two values, naïve K-NN regression will yield a value which is the mean of the two, and may be quite far from both. Instead, we can introduce a *clustering* step whose objective is to detect the occurrence of such ambiguity and separate the distinct values. The regression procedure (e.g., averaging in the K-NN case) is then applied to each cluster of the labels. This results in multiple answers rather than a single prediction. Figure 2-1 illustrates this for the case of linear regression model. The query point (cross) matches a number of neighbors (circles), that correspond to two "modes" of the target function g. Clustering them according to the value of g is straightforward, and the final estimation is carried out separately on each cluster, producing two linear models shown by dashed lines.

How these answers are used depends on the application at hand. An additional procedure aimed at resolving the ambiguity may be applied; an example of such approach is taken in the orientation estimation described in Section 5.2, where we produce two estimates of orientation that are subsequently refined based on temporal context. Alternatively, we may "propagate" the multiple answers, and defer the resolution of the ambiguity until later stages in the decision process, or even report them as the end result of the estimation.

2.2.1 Regression-induced similarity

The key underlying assumption in most example-based regression methods is that the target function behaves smoothly enough within any small enough region to be well modeled by a relatively simple (low order) model in that region. Thus the choice of the neighborhood is crucial for finite sample cases. This choice is typically tuned by comparing the prediction accuracy on the training data or, if the amount of available data allows that, in a cross-validation procedure, for a range of neighborhood-defining

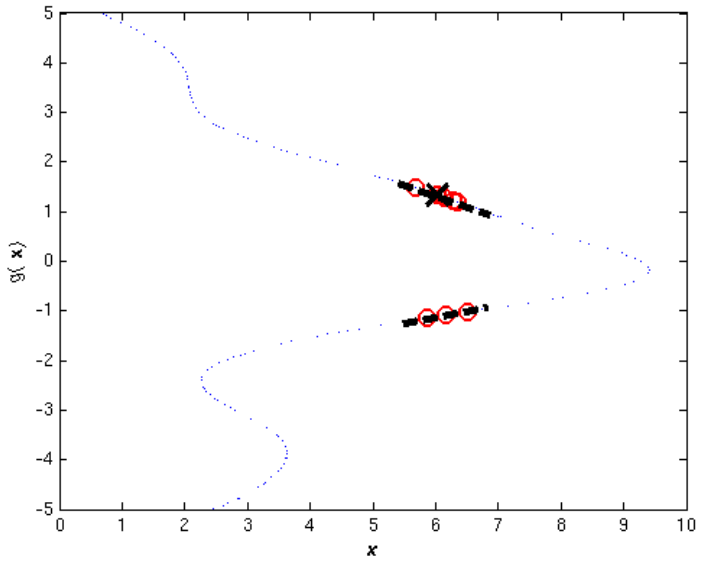

Figure 2-1: Illustration of the idea of regression disambiguation by clustering the labels. Cross: query point and its true label; circles: the neighbors; the dashed lines show the two models fit separately to the two clusters.

parameters: K for K-NN or R for R-neighbors.

Consider however a different notion of similarity: we will define two examples $\mathbf{x}$ and $\mathbf{y}$ in $\mathcal{X}$ to be similar if the values of the target function g are similar. The latter similarity is defined in a straightforward manner, depending on the application at hand, the precision required and the behavior of the function g. When the range $\mathcal{Y}$ of g is a metric space, a natural way to define such similarity is by setting a threshold r in $\mathcal{Y}$, and defining

$$\mathcal{S}_{g,r}(\mathbf{x}_0, \mathbf{x}) \triangleq \begin{cases} +1 & \text{if } \mathcal{D}_{\mathcal{Y}}(g(\mathbf{x}_0), g(\mathbf{x})) \leq r, \\ -1 & \text{otherwise.} \end{cases} \tag{2.9}$$

Figure 2-2 illustrates this definition. Note that if r is set so that errors within r can be reasonably tolerated in the application, and if we can accurately retrieve examples similar to $\mathbf{x}_0$ w.r.t. $\mathcal{S}_{g,r}$, we should achieve excellent regression results (subject to the noise level in the training labels.) Of course, the problem is that since the value $g(\mathbf{x}_0)$ is not known, the definition in (2.9) is ill-posed and can not be used directly. On the other hand, we can form a large number of pairs $(\mathbf{x}_i, \mathbf{x}_j)$ over the training examples such that they are similar, or dissimilar, under that definition. This naturally leads to the problem of learning similarity from examples.

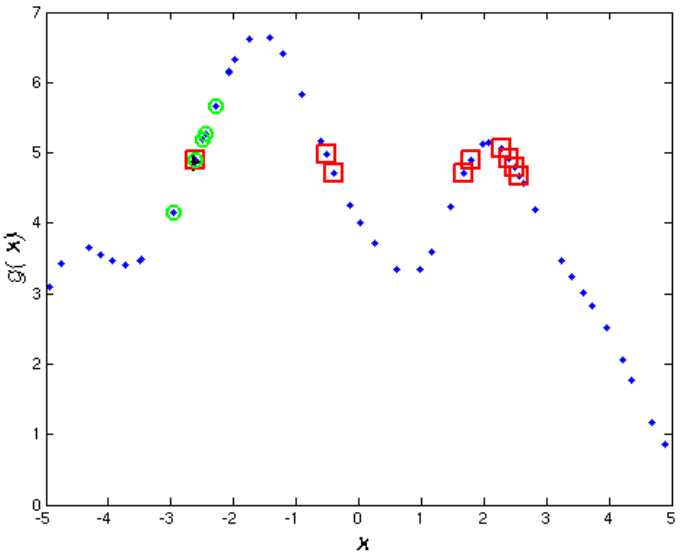

Figure 2-2: Regression-induced similarity. For a query point (black cross), shown are the .5-neighbors in $\mathcal{X}$(green circles) and the $\mathcal{S}_{g,.25}$-neighbors (red squares).

2.3 Learning Distances and Similarity

There exists a large body of literature, both in machine learning and in cognitive science, devoted to the idea of learning distances or similarities from examples and/or for a specific task. Below we review the prior work in some detail, pointing out relevance to the stated problem of learning an equivalence concept and the differences from our approach.

2.3.1 Metric learning

The most common way to model similarity is to assume that a distance $\mathcal{D}$ can serve as a reasonable "proxy" for the desired similarity $\mathcal{S}$. Many methods assume that $\mathcal{D}$ is a *metric*, complying with the following three properties:

$$\forall \mathbf{x} \in \mathcal{X}, \qquad \mathcal{D}(\mathbf{x}, \mathbf{x}) = 0; \tag{2.10}$$

$$\forall \mathbf{x}_1, \mathbf{x}_2 \in \mathcal{X}, \qquad \mathcal{D}(\mathbf{x}_1, \mathbf{x}_2) \geq 0; \tag{2.11}$$

$$\forall \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \in \mathcal{X}, \qquad \mathcal{D}(\mathbf{x}_1, \mathbf{x}_2) + \mathcal{D}(\mathbf{x}_2, \mathbf{x}_3) \geq \mathcal{D}(\mathbf{x}_1, \mathbf{x}_3). \tag{2.12}$$

Sometimes $\mathcal{D}$ is allowed to be a *pseudo-metric*, i.e. it may violate the triangle inequality (2.12).

The most common scenario in which this approach has been applied is a classification (or equivalently clustering) setup, where the objective is to produce a metric that minimizes label error with a specific classification or clustering algorithm. Notable examples of work in this direction include [78, 118, 79, 61, 53]. In these applications,

in addition to the metric assumptions in (2.10)-(2.12), it is typically assumed that the target equivalence concept on pairs in $\mathcal{X}^2$ induces equivalence classes in $\mathcal{X}$. The key reason for that is that the metric-learning methods usually depend on *transitivity* of similarity: they assume that if $\mathcal{S}(\mathbf{x}, \mathbf{y}) = +1$ and $\mathcal{S}(\mathbf{x}, \mathbf{z}) = +1$ than $\mathcal{S}(\mathbf{x}, \mathbf{z}) = +1$.

As we stated earlier, we would like to avoid such transitivity assumption. In particular, this assumption clearly does not hold in the context of regression-induced similarity defined in Section 2.2.1, such as the pose estimation domain described in Chapter 4. Neither does it hold in general matching problems, such as the image patch classification in Chapter 6. If a region in an image is repeatedly shifted by one pixel 100 times, most of the consecutive regions in the sequence will be visually similar, however it will hardly be the case for the first and the 100th regions.

Another important difference of our approach is in the class of attainable similarity measures. Metric learning methods are typically based on a particular parametric form, often a quadratic form of the data, whereas our approach is non-parametric.

2.3.2 Similarity as classification

A very different family of approaches take advantage of the duality between binary classification on pairs and similarity. Formulated as a classification problem, the task of learning similarity may be approached using the standard arsenal of techniques designed to learn classifiers. A number of such approaches have been proposed in the area of face analysis, where pairs of faces are to be classified as belonging to the same or different persons, either in a verification context or as part of a matching-based recognition. Typically it is done by modeling the differences between the two images in some parametric form, either probabilistic [85] or energy-based [20]. A different approach is taken in [69], where the classifier is obtained by boosting local features, in a way similar to our learning algorithms. However, none of that work is extended to learning an embedding.

The classification approach to modeling similarity face two major challenges. One is inherent in the nature of similarity in many domains: the classification task induced by similarity is typically extremely unbalanced. That is, similarity is a *rare event*: the prior probability for two examples to be similar may be very low. Consequently, the negative class is much larger and, in a sense, more diverse and more difficult to model. Although some solutions to such situations have been proposed, in particular in the context of detection of rate events [117, 114] this remains a difficulty for learning algorithms.

The other challenge is in the realm of practical applications of the learned similarity concept. Most of the classifiers are ill-suited for performing a search in a massive database; often the only available solution is to explicitly classify all the possible pairings of the query with the database examples, and that does not scale up.

Conceptually, these approaches are closely related to ours, since our algorithms described in Chapter 3 do rely on classification techniques. However, we use those as means to construct an embedding, and the similarity classification itself is done by means of thresholding a metric distance, an approach that easily scales up to high dimensions and large databases, in particular using methods reviewed in Section 2.4.

2.3.3 Embeddings and mappings

Finally, there exists a broad family of algorithms that learn a mapping of the data into a space where similarity is in some way more explicit. Our approach falls into this broad category as well, although it differs from the previously developed ones in important ways. Below we discuss the existing embedding and mapping methods, which can roughly be divided into two categories.

Multidimensional scaling

Multidimensional scaling (MDS) [27] is a family of techniques aimed at discovering and extracting low-dimensional structure of the data. The algorithms in this family expect as their input a set of examples $x_1, \ldots, x_N$ and a (perhaps partial) list of pairwise *dis*similarities δ_{ij} between x_i and x_j. The goal is to map the input examples into a space where Euclidean distance match, as well as possible, the given values of δ.

Let us denote by f the transformation that such a mapping induces on the dissimilarities (from the value of δ_{ij} to the distance between the images of x_i and x_j.) In *metric* MDS, f must be a continuous monotonic function. This form is most relevant to the distance model of similarity mentioned in Section 1.1, but less so to the boolean similarity case. More relevant is the *non-metric* MDS (NMDS), in which the transformation f can be arbitrary, and is only subject to monotonicity constraint: if $\delta_{ij} < \delta_{kl}$ then $f(\delta_{ij}) \leq f(\delta_{kl})$, i.e., it only must preserve the rank. Technically, NMDS may be directly applied to the problem of learning an equivalence similarity, in which case there are only two ranks since all $\delta_{ij} \in \{-1, 1\}$. However, NMDS does not learn an embedding in the sense our algorithms do: it finds the mapping of the training examples into a lower-dimensional Euclidean space, but does not provide a way to map a previously unseen example. Another difference is the assumption of low dimensionality of the embedding space, which is not explicitly present in our work.

In addition to a large set of classical MDS techniques [27], notable methods that fit this description include, Isomap [112] and local linear embedding [96]. Some recent work aimed at extending these techniques to unseen points is discussed in [12], along with a unifying perspective on these and other methods. The focus there is, however, on metric MDS in the context of manifold learning and clustering. In general, approaches to extending the embedding in MDS-style methods to new examples proceed by finding the NN of $\mathbf{x}_0$ in $\mathcal{X}$ and combining their embeddings (for example, averaging) to produce an estimated embedding of $\mathbf{x}_0$. In contrast, our approach avoids such dependence on the original distance in $\mathcal{X}$, that can be detrimental when there is a significant departure of $\mathcal{S}$ from that distance.

Embedding of known distance

Many methods have been developed for constructing a *low-distortion embedding* of the original data space into a space where L_1 can be used to measure similarity. They assume that the underlying distance is known, but expensive to compute. The embedding is used either to approximate the true distance [44, 55, 56] or to apply a

filter-and-refine approach [42, 5] in which the embedding space is used for fast pruning of the database followed by exact distance calculations in the original space. Two main differences of these algorithms from MDS and related methods is that the dimension of the embedding is usually very high, often many times higher than the dimension of the input space, and that the construction of the embedding is usually guided by analytically known properties of the underlying distance rather than learned by the data. A recent review of other related methods can be found in [62], however many of them are better categorized as search algorithms rather than learning similarity.

2.4 Algorithms for search and retrieval

In this section we discuss the state of the art in search and retrieval, decoupled from the problem of learning and representing similarity. All of these algorithms assume that the dissimilarity is expressed by a distance (almost always a metric, usually Euclidean or L_1. This is generally not the case in the problems we are concerned with in this thesis, however, we can rely on these methods to allow, after an embedding has been learned, for efficient search under L_1 distance in the embedding space. Indeed this is one of the main motivations for our embedding approach. The dimension of our embedding space may be quite high, as we will see in the following chapters, and a method of choice must handle high-dimensional spaces well.

The most straightforward method is the linear scan, often referred to as *brute force* search: inspect all the examples in the database and measure the distance between them and the query. This is the simplest solution, but for a very large number of high-dimensional examples it quickly becomes infeasible. We will therefore focus on methods that allow some speedup relative to the brute force search.

2.4.1 kd-trees

In the kd-tree approach [13, 32], the space $\mathcal{X}$ is partitioned by a multidimensional binary tree. Each vertex represents a (possibly unbounded) region in the space, which is further partitioned by a hyperplane passing through the vertex and perpendicular to one of the coordinate axes of $\mathcal{X}$. The partition is done so that the set of points that belong to the region represented by a vertex is roughly equally divided between that vertex's children. Furthermore, the orientation of the partitioning hyperplanes alternates through the levels in the tree. That is, the first hyperplane is perpendicular to X_1, the two hyperplanes in the second level are perpendicular to X_2 and so on, starting over again with the first dimension at the level $dim(\mathcal{X}) + 1$.

The standard way of querying the kd-tree is by specifying a region of interest; any point in the database that falls into that region is to be returned by the lookup. The search algorithm for kd-tree proceeds by traversing the tree, and only descending into subtrees whose region of responsibility, corresponding to the partitions set at construction time, intersects the region of interest.

When $dim(\mathcal{X})$ is considered a constant, the kd-tree for a data set of size N can be constructed in $O(N \log N)$, using $O(N)$ storage [32]. Its lookup time has been

shown to be bounded by $O\left(N^{1-1/dim(\mathcal{X})}\right)$. Unfortunately, this means that kd-trees do not escape the curse of dimensionality mentioned in Section 2.1.1: for very high dimensions the worst case performance of kd-tree search may deteriorate towards the linear scan. Nevertheless, in more than three decades, *kd*-trees have been successfully applied to many problems. As a rule of thumb, their average performance is typically very good for dimensions below 10, and reasonable for dimensions up to 20; however, for hundreds of dimensions kd-trees are often impractical.

A number of modifications of the kd-tree scheme have been aimed at reducing the lookup time. For the classification scenario, a method has been proposed in [75] for NN classification, using a data structure similar to the kd-trees but with overlapping partitions; the insight of that approach is that in order to predict the majority vote among the NN it may not be necessary to explicitly retrieve the neighbors themselves. In a more general setting, a number of modifications have been proposed that change the order in which the tree is traversed [10] or randomization by early pruning [4].

We now turn to another approach to approximate similarity search, that is provably efficient even in very high dimensional spaces and that has seen a lot of attention in the recent years. Besides its utility for the search problems we will encounter, this approach has provided inspiration to some of the central ideas in this thesis.

2.4.2 Locality sensitive hashing

LSH [65, 52] is a randomized hashing scheme, developed with the primary goal of ϵ-R neighbor search. The main building block of LSH is a family of *locality sensitive* functions. A family $\mathcal{H}$ of functions $h : \mathcal{X} \to \{0, 1\}$ is (p_1, p_2, r, R)-sensitive if, for *any* $\mathbf{x}, \mathbf{y} \in \mathcal{X}$,

$$\Pr_{h \sim U[\mathcal{H}]}(h(\mathbf{x}) = h(\mathbf{y}) \mid \|\mathbf{x} - \mathbf{y}\| \leq r) \;\geq\; p_1, \tag{2.13}$$

$$\Pr_{h \sim U[\mathcal{H}]}(h(\mathbf{x}) = h(\mathbf{y}) \mid \|\mathbf{x} - \mathbf{y}\| \geq R) \;\leq\; p_2. \tag{2.14}$$

The probabilities are over a random choice of $h \in \mathcal{H}$; more precisely, the functions are assumed to be parametrized with a bounded range of parameter values, and the notation $U[\mathcal{H}]$ denotes uniform sampling of those values. A family $\mathcal{H}$ is of course useful only when $r < R$, and when there is a *gap* between p_1 and p_2, i.e. when $p_1 > p_2$. This notion of a gap is very important, and will inspire our approach to learning similarity.

Algorithm 3 gives a concise description of the LSH construction algorithm for a particularly simple case, when the distance of interest is L_1. The family $\mathcal{H}$ in this case contains axis-parallel stumps, i.e. a value of an $h \in \mathcal{H}$ is obtained by taking a single dimension $d \in \{1, \ldots, dim(\mathcal{X})\}$ and thresholding it with some T:

$$h^{\text{LSH}} \;=\; \begin{cases} 1 & \text{if } x_d \leq T, \\ 0 & \text{otherwise.} \end{cases} \tag{2.15}$$

An LSH function $g : \mathcal{X} \to \{0, 1\}^k$ is formed by independently k function $h_1, \ldots, h_k \in$

$\mathcal{H}$ (which in this case means uniform sampling of a dimension d and a threshold T on that dimension). Applied on an example $\mathbf{x} \in \mathcal{X}$, it produces a k-bit hash key

$$g(\mathbf{x}) = [h_1(\mathbf{x}), \ldots, h_k(\mathbf{x})].$$

This process is repeated l times and produces l independently constructed hash functions $g_1, \ldots, g_l$. The available reference (training) data X are indexed by each one of the l hash functions, producing l hash tables.

Algorithm 3 LSH construction (from [52])

Given: Data set $X = [\mathbf{x}_1, \mathbf{x}_N]$, $\mathbf{x}_i \in \mathbb{R}^{\dim(\mathcal{X})}$.
Given: Number of bits k, number of tables l.
Output: A set of

1: **for all** $j = 1, \ldots, l$ **do**
2: **for all** $i = 1, \ldots, k$ **do**
3: Randomly (uniformly) draw $d \in \{1, \ldots, dim(\mathcal{X})\}$.
4: Randomly (uniformly) draw $\min\{\mathbf{x}_{(d)}\} \leq v \leq \max\{\mathbf{x}_{(d)}\}$.
5: Let h_i^j be the function $\mathcal{X} \to \{0, 1\}$ defined by

$$h_i^j(\mathbf{x}) = \begin{cases} 1 & \text{if } \mathbf{x}_{(d)} \leq v, \\ 0 & \text{otherwise.} \end{cases}$$

6: The j-th LSH function is $g_j = [h_1^j, \ldots, h_k^j]$.

Once the LSH data structure has been constructed it can be used to perform a very efficient search for approximate neighbors, in the following way. When a query $\mathbf{x}_0$ arrives, we compute its key for each hash table j, and record the examples $C_= \{\mathbf{x}_1^l, \ldots, \mathbf{x}_{n_l}^l\}$ resulting from the lookup with that key. In other words, we find the training examples (if there any) that fell in the same "bucket" of the l-th hash table to which $\mathbf{x}_0$ would fall. These l lookup operations produce a set of *candidate matches*, $C = \bigcup_{j=1}^{l} C_j$. If this set is empty, the algorithm reports that and stops. Otherwise, the distances between the candidate matches and $\mathbf{x}_0$ are explicitly evaluated, and the examples that match the search criteria, i.e. that are closer to $\mathbf{x}_0$ than $(1 + \epsilon)R$, are reported.[5] This is illustrated in Figure 2-3.

LSH is considered to fail on a query $\mathbf{x}_0$ if there exists at least one R-neighbor of $\mathbf{x}_0$ in X, but the algorithm fails to find any $(1 + \epsilon)R$-neighbor; any other outcome it's a success. It was shown in [65, 52] that the probability of success can be made arbitrarily high by suitable choice of k and l; at the same time, there is a trade-off between this probability and the expected running time, which is dominated by the explicit distance calculations for the candidate set C.

[5]The roles of r and R seem somewhat arbitrary: one could ostensibly define R to be the desired distance and r to be $R/(1 + \epsilon)$. However, the actual values are important since they determine p_1 and p_2. They also define the event of success: if there are no points at distance r but there exists a point at distance R, the algorithm is not required to find it.

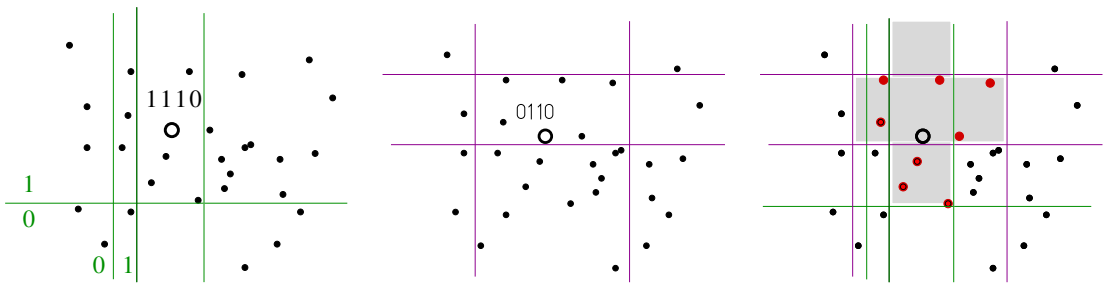

Figure 2-3: An illustration of LSH lookup. Left and center: two LSH tables with $k = 4$ bits based on axis-parallel stumps (assigning zero if a point falls to the left or below the threshold). The circle shows a query point, with the value of its hash key in each table. Right: the union of the two buckets is shaded. Points in the candidate set C shown enlarged; only these candidates are explicitly compared to the query.

The analysis of LSH in [52] is based on the following concept of *unary* encoding, which we describe below since it is relevant to our task as well. Suppose that all components of all the examples in a given data set are integers, and that the values of dimension d for all examples lie in between 0 and U_d. This does not cause loss of generality since one can always preprocess any finite data set represented with finite precision to adhere to these assumptions, by shifting and scaling the data. For each dimension d of $\mathcal{X}$, we write out U_d bits $u_d^1, \ldots, u_d^{U_d}$, where

$$u_d^j \triangleq \begin{cases} 0 & \text{if } \mathbf{x}_{(d)} < j, \\ 1 & \text{if } \mathbf{x}_{(d)} >= j. \end{cases} \tag{2.16}$$

The unary encoding of $\mathbf{x}$ is obtained by concatenating these bits for all the dimensions. For example, suppose $\mathcal{X}$ has two dimensions, and the span of the dimensions is $0, \ldots, 4$ and $0, \ldots, 7$ respectively (we are following the assumption above according to which the values are all integers). Then, the unary encoding for the example $[2, 4]$ will be

$$[\ \underbrace{1,1,0,0}_{\text{1st dimension}}\ ,\underbrace{1,1,1,1,0,0,0}_{\text{2nd dimension}}\ ].$$

The unary code has length $\sum_d U_d$, and is of course extremely wasteful (see Table 3.2 for some examples of unary encoding lengths for real data.) Fortunately, one does not need to actually compute the code in order to use LSH.

A later version of LSH [31] expands the locality-sensitive family defined in (2.13) to include arbitrary linear projections (namely, dot products with random vectors in $\mathcal{X}$), and uses quantization into more than two values. That is, the line $f(\mathbf{x}) = \mathbf{x} \cdot \mathbf{r}$, where $\mathbf{r}$ is the random vector defining the projection f, is divided into m regions, and the hash key is formed by concatenating k numbers from 1 to m, rather than bits. This version of the scheme, called E^{2}LSH, extends the guarantees of locality-similar hashing to L_p norms for $1 \leq p \leq 2$; the basic underlying principles, including the exploitation of the gap between $p_1 - p_2$, remain the same. Besides having appealing

theoretical properties, LSH has already been successful in practical applications, in particular in computer vision problems where the ability to do fast lookup in large databases is crucial. Some examples include [49, 55, 51] and also the work in [105] and [93], which is part of this thesis.

We can now make a connection between the idea of LSH and the learning framework developed in the next chapter. Each bit in the unary encoding is a *feature* of the input, and LSH is randomly selecting a set of kl (not necessarily distinct) features. This works since that specific family of features is locality-sensitive, with respect to L_1 norm. However, no such guarantee exists for general similarity concepts, that may not adhere to any metric. Moreover, in general the similarity is not known analytically, and therefore it is not possible to analytically design an LSH family and prove its properties. We therefore are interested in a method that would *learn* a set of locality-sensitive functions entirely from data.[6]

Consequently, we will have to change the notion of locality-sensitive set of functions from (2.13) to the following definition of a *similarity sensitive* family. Let p_1, p_2 be probability values and $\mathcal{S}$ be a similarity (equivalence) concept. A family $\mathcal{H}$ of functions $h : \mathcal{X} \to \{0, 1\}$ is $(p_1, p_2, \mathcal{S})$-sensitive if, for any $h \in \mathcal{H}$,

$$\Pr_{\mathbf{x},\mathbf{y} \sim\ p(\mathcal{X})^2}(h(\mathbf{x}) = h(\mathbf{y}) \mid \mathcal{S}(\mathbf{x}, \mathbf{y}) = +1) \ \geq \ p_1, \tag{2.17}$$

$$\Pr_{\mathbf{x},\mathbf{y} \sim\ p(\mathcal{X})^2}(h(\mathbf{x}) = h(\mathbf{y}) \mid \mathcal{S}(\mathbf{x}, \mathbf{y}) = -1) \ \leq \ p_2. \tag{2.18}$$

An important difference between this definition and (2.13) is in the placement of qualifiers. In (2.13) it is assumed that the data are fixed, and that the distance of interest is L_p. In our case, the roles are interchanged: we are interested in finding deterministic functions that are expected (i.e. have high probability) to be sensitive to similarity under $\mathcal{S}$ for a random input. Thus, the probabilities in (2.17) are taken with respect to randomly drawn data $\mathbf{x}, \mathbf{y}$, and not random functions.

2.5 Summary

In this chapter we have reviewed the main example-based methods for regression and estimation, in the context of which we develop our learning approach. The central computational task in these methods is the search in a labeled database for examples similar to a query. For cases where the similarity underlying this task is well represented by an analytically defined distance, there exist methods that allow for efficient solution, exact or approximate. However, the nature of similarity underlying this task is often defined by the task at hand, lacks known analytical form, and it is often beneficial to learn it from examples. We have discussed a number of approaches that have been proposed to this and related problem, some of which have inspired the work presented in this thesis. In the next chapter we develop a new approach

[6] As in any learning approach, we of course will also use certain amount of information not contained in the data per se, such as a hypothesis regarding the suitable parametric form of the projections.

that combines some of the ideas behind LSH, similarity classification and learning embeddings in one learning framework.