Learning Task-Specific Similarity

by

Gregory Shakhnarovich

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2005

Author . . . .
Department of Electrical Engineering and Computer Science
September 30, 2005

Certified by. . . . .
Trevor J. Darrell
Associate Professor
Thesis Supervisor

Accepted by . . . .
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Learning Task-Specific Similarity
by
Gregory Shakhnarovich

Submitted to the Department of Electrical Engineering and Computer Science
on September 30, 2005, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

The right measure of similarity between examples is important in many areas of computer science. In particular it is a critical component in example-based learning methods. Similarity is commonly defined in terms of a conventional distance function, but such a definition does not necessarily capture the inherent meaning of similarity, which tends to depend on the underlying task. We develop an algorithmic approach to learning similarity from examples of what objects are deemed similar according to the task-specific notion of similarity at hand, as well as optional negative examples. Our learning algorithm constructs, in a greedy fashion, an encoding of the data. This encoding can be seen as an embedding into a space, where a weighted Hamming distance is correlated with the unknown similarity. This allows us to predict when two previously unseen examples are similar and, importantly, to efficiently search a very large database for examples similar to a query.

This approach is tested on a set of standard machine learning benchmark problems. The model of similarity learned with our algorithm provides and improvement over standard example-based classification and regression. We also apply this framework to problems in computer vision: articulated pose estimation of humans from single images, articulated tracking in video, and matching image regions subject to generic visual similarity.

Thesis Supervisor: Trevor J. Darrell
Title: Associate Professor

Acknowledgments

I would like to thank my advisor, Trevor Darrell, for his influential role in this thesis and in my graduate career. Being Trevor's student has been fun in many ways; he has struck the perfect balance in his role as an advisor. On the one hand, he has given me a great deal of independence in pursuing my ideas and lots of encouragement. On the other hand he has provided thorough advice (on things academic and not) and, at times, challenging the nonsense I would produce. Not less importantly, Trevor has succeeded in building a really great research group, where things go smoothly professionally and socially.

Bill Freeman, Mike Collins and Shimon Ullman have been great thesis readers, and their careful and challenging feedback has made the thesis much better than it would otherwise be. I am also thankful for their patience with my ever shifting deadlines. My gratitude also goes to Tommi Jaakkola for many helpful conversations about learning, and for the enriching experience of TA-ing his machine learning course, and to Piotr Indyk, whose work has inspired much of the work described in the thesis and who was always willing to meet and talk. John Fisher has been an excellent colleague and friend, and I thank him for his appreciation of my Soviet jokes (one in particular).

My work and thinking has also been benefited by a number of collaborations outside of MIT. A summer internship at MERL with Paul Viola taught me many things about vision, learning and scientific intuition. Paul has contributed some of the ideas in the core of this thesis, and has collaborated with Trevor and me on the work presented in Chapter 4. I have also benefited from collaboration with Baback Moghaddam, Liu Ren, Jessica Hodgins, Paul Viola and Hanspeter Pfister; Liu led much of the effort in the last stages of the project described in Section 4.5.

During the five years I have spent at MIT, I have learned a great deal from courses and talks, but probably more so from the many seemingly random conversations with fellow students, staff and faculty. Naturally, the most frequent victims have been my officemates. I am very grateful to Leonid Taycher, Kristen Grauman and Ariadna Quattoni for many interesting conversations, for being kind and helpful in all matters, and for humoring me by listening to my ramblings and tolerating the (often bad) jokes. The other members of Trevor's research group have made my time here more productive and enjoyable. I would like to especially mention Mario Christoudias, David Demirdjian, Louis-Philippe Morency, Ali Rahimi, Kate Saenko

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The need to automatically decide whether, and/or to what extent, two objects are similar arises in many areas of computer science. Sometimes it is explicit, for instance in nearest neighbor methods that rely on finding training instances similar to the input, or in information retrieval applications. In other cases, for instance in probabilistic models that use dissimilarity computations to derive model parameters, this need is implicit. The notion of similarity judgment has been also in the focus of a large body of research in cognitive science. It is known that people can perceive and judge similarity at different cognitive levels, and that the semantics of such judgments may depend on the task.

The basic idea explored in this thesis is that the notion of task-specific visual similarity can be, and should be, learned from examples of what is to be considered similar for a given task. Specifically, we develop an new approach that learns an *embedding* of the data into a metric space where a (possibly weighted) Hamming distance is highly faithful to the target similarity. A crucial practical advantage of this approach is that a search for examples similar to a given query is reduced to a standard search in the metric embedding space and thus may be done extremely quickly, leveraging an arsenal of randomized algorithms developed for that purpose. In some of the applications reported here we use our embedding approach in conjunction with locality sensitive hashing, and achieve state-of-the-art performance in sublinear time.

We develop a family of algorithms for learning such an embedding. The algorithms offer a trade-off between simplicity and speed of learning on the one hand and accuracy and flexibility of the learned similarity concept on the other hand. We then describe two applications of our similarity learning approach in computer vision: for a regression task of estimating articulated human pose from images and videos, and for a classification task of matching image regions by visual similarity. To our knowledge, this is the first example-based solution to these problems that affords a feasible implementation.

In the context of regression, the novelty of our approach is that it relies on learning an embedding that directly reflects similarity in the target space. We can use this embedding to retrieve training examples in which the target function with high probability has values similar to the value on the input point. We combine the embedding

with the search algorithm using randomized hashing and with a clustering step that allows for multi-modal estimation.

This Introduction is organized as follows. Section 1.1 gives defines more formally the task we are addressing. Section 1.2 outlines the basic ideas in our approach to learning similarity. The computer vision applications of this approach are briefly described in Section 1.3. Finally, Section 1.4 describes the organization of the remainder of the thesis.

1.1 Modeling equivalence

The central learning problem addressed in this thesis can be formulated as follows. Let $\mathcal{X}$ denote the *data space* in which the examples are represented. We will define an *equivalence similarity* concept as a binary relation $\mathcal{S}(\mathbf{x}, \mathbf{y}) \rightarrow \pm 1$, that specifies whether two objects $\mathbf{x} \in \mathcal{X}$ and $\mathbf{y} \in \mathcal{X}$ are similar (+1) or not (-1). We will assume, unless noted otherwise, that this relation is reflexive, i.e. $\mathcal{S}(\mathbf{x}, \mathbf{x}) = +1$, and symmetric, i.e. $\mathcal{S}(\mathbf{x}, \mathbf{y}) = \mathcal{S}(\mathbf{y}, \mathbf{x})$. However we will not require $\mathcal{S}$ to be transitive, and so it will not necessarily induce equivalence classes on $\mathcal{X}$.

We develop an approach to learning a model of such similarity relation *from examples* of pairs that would be labeled similar, and ones that would be labeled dissimilar, by $\mathcal{S}$. We also show how, under certain assumptions, such learning can be done in a scenario in which *only positive* examples are provided, in addition to some unlabeled data.

Our objective in learning similarity is dual:

- To develop a *similarity classifier*, that is, to build an estimator that given a novel pair of objects in $\mathcal{X}$ predicts, as accurately as possible, the label $\mathcal{S}$ would have assigned to it.

- To provide framework for a very efficient *similarity search*. Given a large database $\mathbf{x}_1, \ldots, \mathbf{x}_N$ of examples and a query $\mathbf{x}_0$ we would like to have a method for retrieving examples in the database that are similar (with respect to $\mathcal{S}$) to the query, *without* having to apply the similarity classifier to every possible pair $(\mathbf{x}_0, \mathbf{x}_i)$.

The embedding approach developed in this thesis allows us to achieve both of these goals in a single learning framework.

1.1.1 Other notions of similarity

Similarity can be defined at two additional levels of "refinement", which we do not address here. However we describe these notions of similarity below in order to clarify the distinction from the problem outlined above.

Ranking One can define a relative similarity: for two pairs of examples, $\mathbf{x}, \mathbf{y}$ and $\mathbf{z}, \mathbf{w}$ one can determine whether or not $\mathcal{S}(\mathbf{x}, \mathbf{y}) \leq \mathcal{S}(\mathbf{z}, \mathbf{w})$. In principle such similarity model defines a binary classification problem on *pairs of pairs* of examples.

Distance At the most refined level, $\mathcal{S}$ could produce a non-negative real number for any pair of examples; the smaller this number the more similar the two examples are. Such a regression mapping $\mathcal{X}^2 \to \mathbb{R}_+$ corresponds to the standard notion of a distance between pairs. The distance values of course induce a ranking relation, as well. It may also be possible to obtain a consistent set of distances from ranking, by methods like multidimensional scaling (Section 2.3.3) but in general the information available at this level is more rich than the other two.

In this thesis, unless otherwise noted the term "similarity" will refer to equivalence. At the end of the thesis we will discuss how the approach we develop could be extended to the ranking notion of similarity. As for learning a real-valued, distance notion of similarity, we will not pursue it here.

1.1.2 Example-based methods

In some cases, the goal of an application is explicitly to predict the similarity judgment on two examples $\mathbf{x}, \mathbf{y} \in \mathcal{X}$, under a particular similarity $\mathcal{S}$. This is a *classification* problem over the space of pairs $\mathcal{X} \times \mathcal{X}$. However, very often in machine learning the ability to automatically judge similarity of example pairs is important not in itself, but as part of an example-based method.

The distinction between model-based and example-based classification is often loose; here we attempt to frame it in terms of the manner in which training examples are used to predict the label of a new input.

Model-based methods use the training examples to build a model–of the class or of the target function. More often than not the model is parametric. A common example is to fit a parametric model of probability density to examples from each class; the very popular family of classification methods based on principal component analysis belongs to this kind. Sometimes it is non-parametric, for instance modeling the class-conditional density with a kernel estimate. The main defining characteristic of a model-based classification is that the input is not explicitly matched with (compared to) individual training examples but rather matched to the model. This is true whether the original training examples are kept around, like in the case of kernel-based non-parametric model, or are discarded after the model is constructed as in principal component analysis, or a mix of the two is used, as in a support vector machine (SVM) [103], where some of the training examples, namely the support vectors, are retained in addition to a set of parameters learned from the entire training set.

In contrast, in example-based methods classification or regression is based explicitly on comparing the input to individual training examples. Widely used example-based methods include nearest-neighbor classification and locally-weighted regression. A generic description of such a method is:

1. Store the training (sometimes called reference) data $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ and the associated labels $\ell_1, \ldots, \ell_n$.

2. Given a query $\mathbf{x}_0$, find examples $\mathbf{x}_{i_1}, \ldots, \mathbf{x}_{i_k}$ in X that are similar to $\mathbf{x}_0$.

3. Infer the label of the query ℓ_0 from $(\mathbf{x}_{i_1}, \ell_{i_1}), \ldots, (\mathbf{x}_{i_k}, \ell_{i_k})$.

The central computational task in the above description is *similarity search*: Given a query $\mathbf{x}_0 \in \mathcal{X}$ and a set of examples $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, find $\mathbf{x}_i$ such that $\mathcal{S}(\mathbf{x}_0, \mathbf{x}_i) = +1$. Technically, this may be equivalent to applying a similarity classifier on n pairs $(\mathbf{x}_0, \mathbf{x}_i)$, $i = 1, \ldots, n$. However, from a practical standpoint such a solution is unacceptable for large datasets, even with a relatively simple classifier. In order to make search feasible, it should be possible to complete in time sublinear in n.

We assume that no analytic expression exists for the "true" similarity concept $\mathcal{S}$, or that no access to such an expression is given to us. Thus we need to construct a *model* $\widehat{\mathcal{S}}$ of similarity, which will be used to predict the values of $\mathcal{S}$.

1.1.3   Why learn similarity?

Before we discuss the details of our approach to learning similarity from data, we briefly discuss some alternatives here, and a more detailed discussion is found in Chapter 2.

A reasonable approach may be to use a distance as a proxy for the desired similarity, namely,

$$\widehat{\mathcal{S}}_{\mathcal{D}}(\mathbf{x}, \mathbf{y}) \;=\; \begin{cases} +1 & \text{if } \mathcal{D}(\mathbf{x}, \mathbf{y}) \leq R, \\ -1 & \text{if } \mathcal{D}(\mathbf{x}, \mathbf{y}) > R. \end{cases} \tag{1.1}$$

The choice of the distance $\mathcal{D}$, and to some extent of the threshold R, may have critical impact on the success of such a model. The most commonly used distances are the L_p metrics, in particular the L_1 (or Manhattan) and the L_2 (Euclidean) distances. These distances account for a vast majority of example-based methods proposed in computer vision when the representation space $\mathcal{X}$ is a vector space of fixed dimension.[1] When the representation does not allow a meaningful application of L_p, the similarity is typically measured in one of two ways. One is to *embed* the data into a metric space and proceed using an L_p distance; the other is to apply a distance measure suitable for $\mathcal{X}$. For instance, when examples are sets of points in a vector space, a common distance to use is the Hausdorff distance [45] or the earth mover's distance [55]. Often one uses an embedding of $\mathcal{X}$ into another, usually higher-dimensional space, in which an L_p metric approximates the complex distance in the original space [5, 55].

However, it is usually possible to provide *examples* of similarity values. The source of such examples depends on the circumstances in which similarity modeling is required. In some cases, similarity values for example pairs may be provided directly, either by manual labeling or via an automated data generation or gathering process. In colloquial terms, this means that a human has a particular concept of similarity in mind, such as "these two image patches look similar", or "these two people have a similar body pose", that allows him/her to serve as an oracle and provide values of $\mathcal{S}(\mathbf{x}, \mathbf{y})$ for some pairs $(\mathbf{x}, \mathbf{y}) \in \mathcal{X}^2$. These values are considered the "ground truth" and the goal of the similarity modeler is to construct an estimate, $\widehat{\mathcal{S}}$, that optimizes

---

[1]The distances under L_1 and L_2 will of course differ, however the ranking, and consequently the set of nearest neighbors, are typically very similar; see, e.g., [52] for a discussion.

the chosen measure of agreement with $\mathcal{S}$. In other words, the goal is to "uncover" the similarity judgment $\mathcal{S}$ used to assign the training labels.

On the other hand, in the context of example-based methods similarity between objects in $\mathcal{X}$ is in effect a "latent concept". Each training example $\mathbf{x}$ in $\mathcal{X}$ is associated with a *label* $\ell(\mathbf{x})$ in a target space $\mathcal{Y}$. Usually, a well-defined similarity $\mathcal{S}_{\mathcal{Y}}$ exists over $\mathcal{Y}$ and can usually be computed analytically. For instance, in a classification scenario $\mathcal{Y}$ is the finite set of class labels, and two labels are similar if they are identical. In a regression setting $\mathcal{Y}$ contains the values of the target function, and similarity may be defined by two values falling within a certain distance from each other. We suggest a natural protocol for defining a similarity over $\mathcal{X}^2$: two examples in $\mathcal{X}$ are considered to be similar under $\mathcal{S}$ if their labels are similar under $\mathcal{S}_{\mathcal{Y}}$. This provides us with a method for inferring values of $\mathcal{S}$ from the labels. The basic challenge remains unchanged: to be able to predict $\mathcal{S}(\mathbf{x}, \mathbf{y})$ without access to the labels $\ell(\mathbf{x}), \ell(\mathbf{y})$ and thus to the ground truth similarity.

A crucial property of similarity is that it can be *task-specific*: the same two examples may be judged similar for one purpose and dissimilar for another. This is illustrated by the following "toy" example. Consider a set of 2D points, with two different notions of similarity illustrated in Figure 1-1 (analyzed in more detail in Chapter 3.) Under the first similarity (top row), two points are similar if their Euclidean norms are close (within a given threshold). Under the second, two points are similar if the angles in their polar coordinates (modulo π) are close. Clearly, Euclidean norms, Manhattan or Mahalanobis distances are not adequate here. The proposed algorithm uses a few hundred examples of pairs similar under the relevant similarity and produces an embedding which recovers the target concept quite well, as shown on the right.

1.2   Learning embeddings that reflect similarity

In the most basic form, our approach can be summarized as follows. We construct an embedding of $\mathcal{X}$ into an M-dimensional space $\mathcal{H}$, each dimension m of which is given by a separate function h_m:

$$H \;:\; \mathbf{x} \in \mathcal{X} \;\rightarrow\; [\alpha_1 h_1(\mathbf{x}), \ldots, \alpha_M h_M(\mathbf{x})], \qquad h_m(\mathbf{x}) \in \{0, 1\}. \qquad (1.2)$$

The value of $\alpha_m > 0$ depends on the specific algorithm, but in all algorithms the h_m are chosen in such a way that the L_1 distance

$$\|H(\mathbf{x}) - H(\mathbf{y})\| \;=\; \sum_{m=1}^{M} |\alpha_m h_m(\mathbf{x}) - \alpha_m h_m(\mathbf{y})|.$$

reflect the underlying similarity. That is, the lower the distance $\|H(\mathbf{x}), H(\mathbf{y})\|$, the higher the certainty of $\mathcal{S}(\mathbf{x}, \mathbf{y}) = +1$. Thus, we follow the paradigm of distance as proxy for similarity (1.1), however the representation, the distance and the threshold R are explicitly chosen with the objective of maximizing the prediction accuracy.
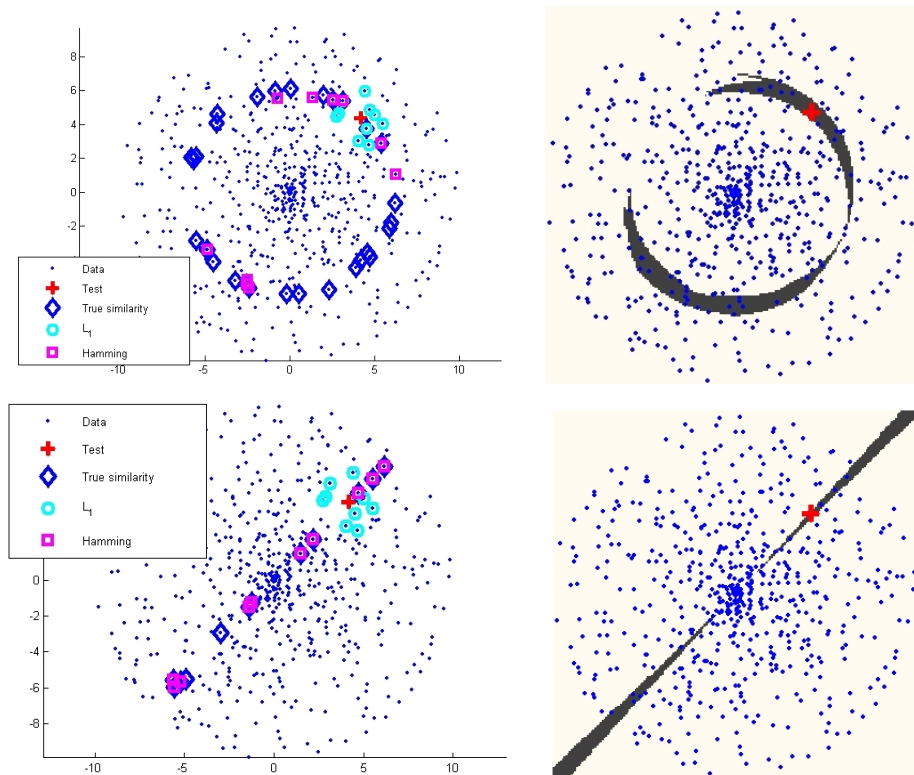
Figure 1-1: Illustration of task-specific similarity modeling on a toy 2D data set. Left: ground truth showing, for one query (cross), examples similar to it (diamonds). Examples found by the BoostPro (Chapter 3) algorithm are shown by squares. Right: similarity regions induced by the query and the embedding learned with BoostPro (200 bits), for a particular distance. Top row: norm similarity, bottom row: angle similarity.

1.2.1 Motivation: hashing and boosting

This approach is inspired by, and to a large extent has evolved from ideas developed in the last decade in two areas of research: randomized search algorithms in computational geometry and ensemble methods in machine learning. Here we briefly describe them, and a more detailed survey can be found in Chapters 2 and 3.

Locality sensitive hashing (LSH)   The LSH [65, 52, 31] is a scheme for approximate similarity search under the L_p metric for $p \in [0, 2]$. It works by indexing the data in a set of l hash tables with independently constructed randomized hash functions, each using a key of k bits. Each bit in the hashing key is computed by projecting the data onto a random vector and thresholding the value. With a suitable setting of parameters l and k, this hashing scheme finds, for a value R, a ϵ-R neighbor of $\mathbf{x}_0$, i.e., an example $\mathbf{x}$ such that $\|\mathbf{x}_0 - \mathbf{x}\| \leq (1 + \epsilon)R$. Its lookup time is $O\left(n^{1/(1+\epsilon)}\right)$, and arbitrarily high probability of success can be achieved. The building block of LSH which provides this guarantee is the notion of a *locality sensitive* hash function, under which the probability of collision is related to the distance in $\mathcal{X}$. When the L_p metric over $\mathcal{X}$ is used as a proxy for the underlying similarity $\mathcal{S}$, the LSH achieves our goal as formulated: the union of distinct bits used in the hash keys defines an embedding in which L_1 distance (in this case equivalent to the Hamming distance) reflects $\mathcal{S}$. A natural question, then, is how to extend the LSH framework to reflect the distance in the unknown embedding space. Our solution is, essentially, to *learn* the locality-sensitive bits and let the bits define the embedding.

Boosting   The idea of boosting [99, 23] is to create an ensemble classifier (or regressor) by greedily collecting simple classifiers that improve the ensemble performance. Each simple classifier only has to be better than chance, hence it is often referred to as a "weak" classifier. A number of variants of boosting have been published so far; in Chapter 3 we review the specific boosting algorithms relevant to our work. The general strategy shared by boosting methods is to assign weights to the training examples and manipulate these weights in order to steer the iterative greedy selection process towards improving the desired properties of the ensemble.

The learning approach in this thesis was inspired by these ideas, and has adapted them for the purpose of constructing a similarity-reflecting embedding. The algorithms outlined below and described in detail in Chapter 3. The order in which they are presented corresponds to the evolution of the underlying ideas and to trading off simplicity of learning for representational power of the resulting embeddings.

1.2.2 Similarity sensitive coding

The first algorithm[2] is essentially a modification of the original LSH approach in which the hashing bits correspond to axis-parallel decision stumps. The operating assumption behind it is that a reasonable approximation to $\mathcal{S}$ may be obtained by calculating the L_1 distance in the data space $\mathcal{X}$, when the following "corrections":

---

[2]Published in [105].

1. Some dimensions of $\mathcal{X}$ may be irrelevant for determining $\mathcal{S}$. These dimensions serve as noise when distance is computed, and are better ignored.

2. For a given dimension, some thresholds (decision stumps) are much more effective (i.e. similarity-sensitive–see Section 2.4.2) than others. Using these thresholds in constructing LSH keys will optimize the properties of the hashing scheme for a given size of the data structure (and thus for a given lookup time.)

3. The determination of the dimensions and thresholds described above is to be guided by the available training data in the form of similar and dissimilar pairs of points in $\mathcal{X}$. The training true positive (TP) rate correspond to the percentage of similar pairs in which both examples (projected on the dimension at hand) fall on the same side of the threshold. The false positive (FP) rate is evaluated similarly by looking at the dissimilar pairs.

This leads to the algorithm called similarity sensitive coding (SSC), first presented in [105] under the name of PSH (parameter-sensitive hashing). For each dimension of $\mathcal{X}$, SSC evaluates the thresholds and selects the ones with acceptable combination of TP and FP rate. The criteria of acceptability depend on the precision/recall rates appropriate for the application at hand, and are formulated as an upper bound on the FP and a lower bound on the TP rates. The data are then indexed by LSH, which uses only the selected stumps as hash key bits. This is equivalent to embedding $\mathcal{X}$ into a binary space

$$H^{\text{SSC}}(\mathbf{x}) \;=\; \left[h_1^{\text{SSC}}(\mathbf{x}), \dots, h_M^{\text{SSC}}(\mathbf{x})\right], \tag{1.3}$$

where each bit $h_m^{\text{SSC}}(\mathbf{x})$ is obtained by quantizing a single dimension i_m in $\mathcal{X}$ into a single bit by thresholding:

$$h_m^{\text{SSC}}(\mathbf{x}) \;=\; \begin{cases} 1 & \text{if } x_{i_m} \le T_m, \\ 0 & \text{if } x_{i_m} > T_m. \end{cases}$$

1.2.3 Boosting the embedding bits

Learning of the decision stumps in SSC is straightforward, and the algorithm has produced good results in the pose estimation domain [105, 35]. However, SSC leaves room for a major improvement: it ignores dependencies between the dimensions of $\mathcal{X}$. The second algorithm of Chapter 3 addresses these issues and employs a boosting algorithm (AdaBoost) which takes the dependencies into account. The boosting algorithm yields an ensemble classifier,

$$C^{\text{AB}}(\mathbf{x}, \mathbf{y}) = \text{sgn}\left[\sum_{m=1}^{M} \alpha_m \left(h_m^{\text{AB}}(\mathbf{x}) - 1/2\right)\left(h_m^{\text{AB}}(\mathbf{y}) - 1/2\right)\right] \tag{1.4}$$

where the single bit functions h_m^{AB} are of the same form as h_m^{SSC}. The resulting embedding is into a weighted binary space

$$H^{\text{AB}}(\mathbf{x}) \;=\; \left[\alpha_1 h_1^{\text{AB}}, \dots, \alpha_M h_M^{\text{AB}}\right]. \tag{1.5}$$

Interestingly, the L_1 (Hamming) distance in this space between $H^{\mathrm{AB}}(\mathbf{x})$ and $H^{\mathrm{AB}}(\mathbf{y})$ is proportional to the *margin* of the AdaBoost classifier,

$$\sum_{m=1}^{M} \alpha_m \left(h_m^{\mathrm{AB}}(\mathbf{x}) - 1/2 \right) \left(h_m^{\mathrm{AB}}(\mathbf{y}) - 1/2 \right).$$

In practice, this algorithms may outperform SSC for a number of reasons:

- The embedding is less redundant and more directly optimized for the underlying similarity prediction task.

- The weights produced by AdaBoost allow for an additional "tuning" of the embedding.

While in principle this is a straightforward application of AdaBoost, a number of interesting practical problems arise when the algorithm is applied to a large amount of data. In particular, under the assumption mentioned in Section 2.1.4 that similarity is a "rare event", the class distribution is very unbalanced. We discuss this issue in Chapter 3.

1.2.4 BoostPro: boosting optimized projections

The final algorithm of Chapter 3, called BoostPro, further advances our approach towards making the embedding more flexible. We leave the realm of axis-parallel decision stumps, and instead propose to use arbitrary *projections* of the data. By a projection we mean any function $f : \mathcal{X} \to \mathbb{R}$; in all the experiments described in this thesis we have used polynomial projections,

$$f(\mathbf{x}) = \sum_{j=1}^{d} \theta_j x_{i_j}^{p_j}, \qquad p_j \in \{1, 2, \ldots\}, \qquad i_j \in \{1, \ldots, dim(\mathcal{X})\}.$$

In contrast to the algorithm outlined in the previous section (where the weak learners only select the threshold), BoostPro uses a gradient-based optimization procedure in the weak learners to improve projection coefficients as well as thresholds, given the current ensemble and the weights on the training data. Furthermore, we introduce a modification of AdaBoost algorithm for the special case of learning similarity, in which learning is done from positive examples only.

Figure 1-2 provides a cartoon illustration of the forms of embedding attainable with each of the algorithms.

1.2.5 Relationship to other similarity learning methods

In Chapter 2 we discuss in some detail the significant body of literature devoted to related topics. Here we attempt to broadly categorize the prior work and emphasize its main differences from the learning approach developed in this thesis.

| (a) SSC | (b) Boosted SSC | (c) BoostPro |
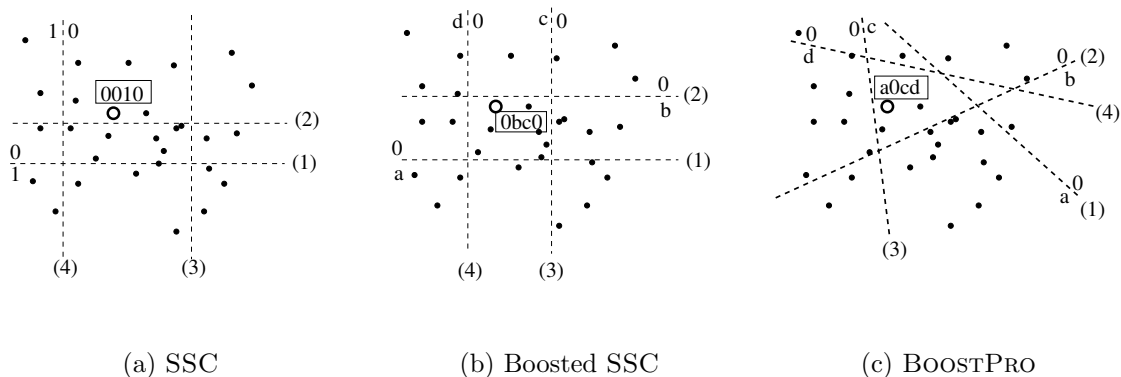
Figure 1-2: Illustration of embeddings obtained with the learning algorithms. Dotted lines show 0/1 boundaries for each bit. Letters correspond to weights, numbers in parenthesis to the order of the bits. Shown in the box is the embedding of the query point (circle). In (c), the case of linear projections is illustrated; for polynomial projections of higher order the boundaries would be nonlinear.

Metric learning   Much work has been done on learning a *metric* $\mathcal{D}$ on $\mathcal{X}$ that is optimized for the use in a particular learning machine, typically a NN classifier. The requirement that the distance be a metric (including transitivity and compliance with triangle inequality) stands as a major difference with our approach. Furthermore, typically the learned metric is constrained to a particular parametric form, usually described by a quadratic form [118, 53]. Thus the class of similarity concepts attainable by these methods is significantly more limited in comparison to our embeddings.

Optimal distance learning   For certain classification tasks there have been proposed algorithms that learn a distance (as a measure of dissimilarity) which is not necessarily a metric, optimized for a particular task—classification or clustering. Among recent work in this direction, [79] and [60, 61] are the closest in spirit to ours. However, the transitivity requirement is retained in these approaches, and it is not clear how to extend them effectively beyond problems with finite label sets.

Manifold learning   Many algorithms have been proposed for learning a low-dimensional structure in data, under the assumption that the data lie on a (possible non-linear) manifold: multidimensional scaling (MDS) [27], Isomap [112], local linear embedding [96] and others (see [12] for a unifying perspective on these and other manifold learning algorithms.) These algorithms usually obtain an embedding of the training data by manipulating the eigenvectors of the pairwise distance matrix. A related family of methods deals with embedding a graph, whose vertices represent examples and edges are weighted by (dis)similarity, in a space where the similarities are preserved.

In contrast to the manifold learning algorithms, our approach does not make an implicit assumption regarding structure in the data, nor does it limit the dimension-

ality of the embedding by the dimensionality of $\mathcal{X}$.[3] A more important difference, however, has to do with extending the embedding to new examples. The MDS and related algorithms do not yield a *mapping function*, which could be applied to a previously unseen example. While some extensions to out-of-sample examples have been proposed [12, 33], they typically rely on the (Euclidean) distance in $\mathcal{X}$ and the ability to find neighbors efficiently among training data–an undesirably circular dependency in the context we are considering here.

1.3 Applications in computer vision

1.3.1 Levels of visual similarity

Visual similarity can be defined at a number of perceptual levels, which differ in the amount of semantic complexity, the dependence on the task at hand, and the potential stages in the visual pathway at which they may be implemented in biological vision systems.

Low-level similarity Two image regions (*patches*) are considered visually similar if they correspond to similar physical scenes. A simple example of this occurs under small motions (translation and rotation) of a camera pointed at a given scene: in most cases, unless there is a discontinuity in appearance due, for examples, to sharp edges, images of the scene in subsequent frames will be similar. The framework developed in this thesis will be applied to learn such similarity – specifically, to predict when two image patches are transformed versions of each other. In essence, the goal is to obtain transformation-invariant similarity on top of non-invariant representation. The learning for this kind of similarity can occur with no human supervision: given a set of natural images, pairs of similar patches can be extracted automatically.

Mid-level similarity On a higher perceptual level (which may be associated with later stages in the visual pathway) visual elements are deemed similar if they share some simple semantic property. An example of such similarity that arises in the object categorization domain is the notion of *parts* - elements that are repeatable in a particular visual category, albeit with some appearance variation. This level of similarity may be important, in particular in an object classification architecture with multiple feature levels.

High-level similarity On an even higher perceptual level, similarity is defined primarily by semantics. These properties that make two objects similar are themselves not visual, but can be inferred (by human perception) from visual information. Two examples of such *task-specific* similarity that we consider in this thesis are object category (do the two objects belong to the same category?) and articulated human

---

[3]Although directly comparing dimensionalities is somewhat inappropriate, since our embedding space is (possibly weighted) binary, as opposed to Euclidean $\mathcal{X}$.
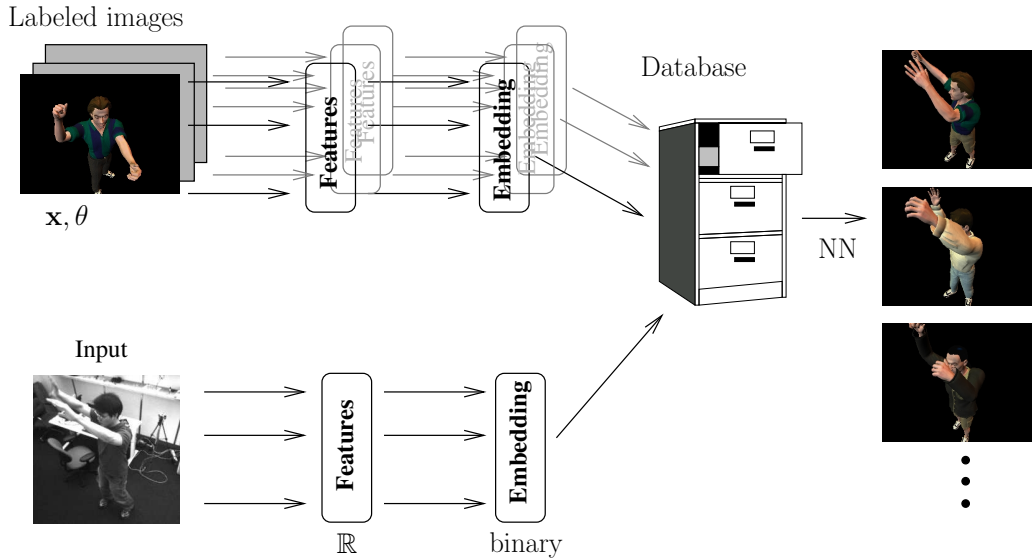
Figure 1-3: A cartoon of the example-based pose estimation approach. The embedding is learned to reflect similarity of the entire pose.

pose estimation (is the body configuration of the two human figures similar?). Note that in the latter case, there is an additional level of dependency on the exact task: two poses that may be judged similar if one only needs to classify a gesture (pointing versus raising one's hand) would not be considered similar if the goal is to recover the 3D location of every body joint with maximal precision.

1.3.2 Example-based pose estimation

Previous model-based approaches have shown that the task of modeling the global relationship between the image and the pose is very difficult. In the proposed approach, we instead model a simpler concept: similarity between the poses that appear in two images. This leads to an example-based estimation algorithm: given an image, find in a large database of images (labeled with the underlying articulated poses) examples classified as similar to the input. This scheme, illustrated in Figure 1-3, relies on the performance of the similarity classifier. Its high true positive (TP) rate provides that with high probability, the unknown pose is close to the poses in the retrieved examples. On the other hand, the low false positive (FP) rate means that not many spurious examples will be retrieved.

A preliminary work in this direction, using SSC, has been presented in [105]. In this thesis we present new experiments with a very large database of poses, obtained with motion capture system, using BoostPro to learn an embedding of images that reflects pose similarity.

1.3.3 Learning visual similarity of image regions

Comparing image regions is a basic task which arises in many computer vision problems: analysis of stereo, image denoising, scene recognition, object categorization etc. Recently, methods that operate by comparing image regions have established themselves as state-of-the-art in some of these problems. Conceptually, there are usually four steps in such methods that directly operate on image regions:

1. *Interest operator*: selecting a set of regions from the given image that are considered "interesting". This is an attention mechanism, and a number of such operators have been proposed. While some appear to be particularly successful in certain cases [77, 84], the choice of interest operator and even its utility is still far from obvious [80, 14], and we will remain agnostic regarding this issue.

2. *Descriptor* The next step is to compute the representation of the selected patches. Ideally, the representation should capture the features that are important to the application that uses the matching method, while being invariant to features that are unimportant. We will consider two representations, that have been the subject of much work in the vision community: the shift-invariant feature transform (SIFT) [77] and the sparse overcomplete codes [89].

3. *Matching* Once the descriptor for a region is computed, it is matched to the descriptors of regions in the database (the training data).

4. *Inference* Depending on the specific task and the method at hand, the results of the matching across the test image are combined to produce an answer.

The matching step clearly provides a natural grounds for applying our learning approach. In Chapter 6 we describe an experiment in which we learn to match patches obtained by transforming an image in certain ways (rotations and mild translations), and show how whereas standard distance-based similarity models fail, the embedding learned by our algorithm allows to detect similarity between transformed versions of the same patches.

1.4 Thesis organization

Chapter 2 provides the background for the thesis research. It describes the prior work in related areas, with particular emphasis on the two ideas that inspired our learning approach: locality sensitive hashing and the boosting. Chapter 3 describes the core machine learning contribution of the thesis–a family of algorithms that produce similarity-reflecting embeddings of the data. Armed with these algorithms we develop example-based approaches for two computer vision domains. In Chapter 4 we describe a method for estimating articulated pose of human figure from a single image, and in Chapter 6 a method for matching image regions based on visual similarity under certain class of transformations. Chapter 7 contains a discussion of the presented approach, and outlines the most important directions for future work.

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 $O(N^{D/2})$ 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 $\hat{y}_0^{(RN)} = \operatorname{argmax}_{y \in \mathcal{Y}} |X_y'|$, 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
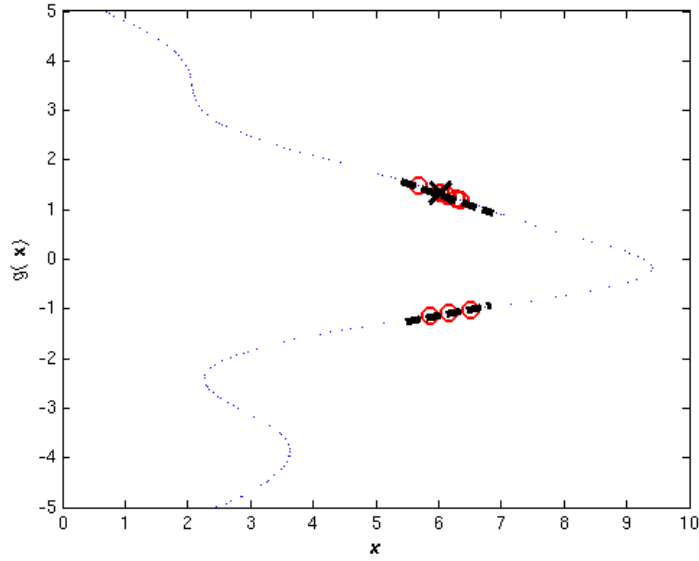
Figure 2-1: Illustration of the idea of regression disambiguation by clustering the labels. Cross: query point and its true label; circles: the neighbors; the dashed lines show the two models fit separately to the two clusters.

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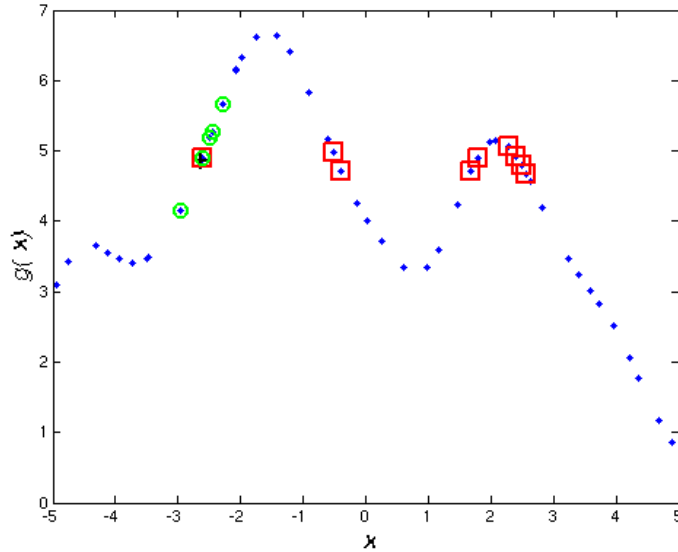
Figure 2-2: Regression-induced similarity. For a query point (black cross), shown are the .5-neighbors in $\mathcal{X}$(green circles) and the $\mathcal{S}_{g,.25}$-neighbors (red squares).

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:

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

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

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

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 $O\left(N \log N\right)$, using $O\left(N\right)$ 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* $\mathbf{x}, \mathbf{y} \in \mathcal{X}$,

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

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

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:

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

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

42

$\mathcal{H}$ (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 $\min\{\mathbf{x}_{(d)}\} \le v \le \max\{\mathbf{x}_{(d)}\}$.
 5:       Let h_i^j be the function $\mathcal{X} \to \{0, 1\}$ defined by

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

 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.
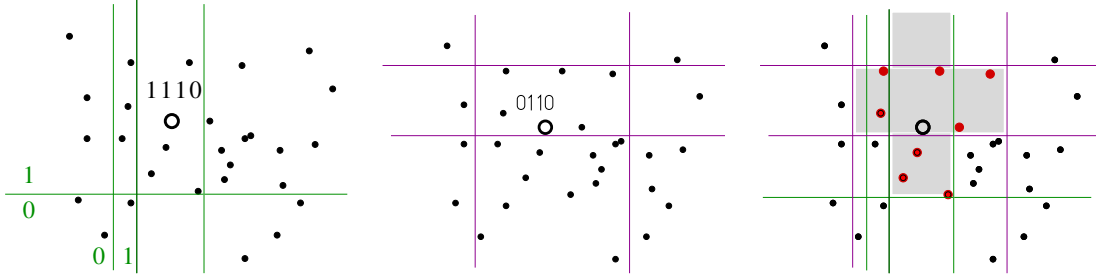
Figure 2-3: An illustration of LSH lookup. Left and center: two LSH tables with $k = 4$ bits based on axis-parallel stumps (assigning zero if a point falls to the left or below the threshold). The circle shows a query point, with the value of its hash key in each table. Right: the union of the two buckets 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 $1 \le p \le 2$; the basic underlying principles, including the exploitation of the gap between $p_1 - p_2$, remain the same. Besides having appealing

44

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 $h : \mathcal{X} \rightarrow \{0, 1\}$ is $(p_1, p_2, \mathcal{S})$-sensitive if, for any $h \in \mathcal{H}$,

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

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

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.

Chapter 3

Learning embeddings that reflect similarity

This chapter describes a family of algorithms for learning an embedding

$$H : \mathcal{X} \; \rightarrow \; [\alpha_1 h_1(\mathbf{x}), \dots, \alpha_M h_M(\mathbf{x})]$$

that is faithful to a task-specific similarity. This means that the lower the distance $\|H(\mathbf{x}) - H(\mathbf{y})\|$ is, the higher is the probability that $\mathcal{S}(\mathbf{x}, \mathbf{y}) = +1$. Consequently, there exists a range of values of R such that if $\mathcal{S}(\mathbf{x}, \mathbf{y}) = +1$ then with high probability $\|H(\mathbf{x}) - H(\mathbf{y})\| < R$, and if $\mathcal{S}(\mathbf{x}, \mathbf{y}) = -1$ then with high probability $\|H(\mathbf{x}) - H(\mathbf{y})\| > R$. For a practical application, such a relationship means that the task of matching a query to a database may be reduced to the task of search for K nearest neighbors or for R-neighbors of the query, embedded in H, among the database examples embedded in the same way.

The order in which the algorithms are presented corresponds to the evolution of this general approach, which in turn corresponds to the trade-off between the simplicity and cost of training and the flexibility, and accuracy, of the embedding. The similarity sensitive coding algorithm in Section 3.2 has evolved from the parameter sensitive hashing (PSH) published in [105]. It can be seen as an improvement of the LSH structure for a similarity measure which is not necessarily identical to an L_p norm in $\mathcal{X}$. The extension using AdaBoost (Section 3.3), published in [93] has considerably higher training complexity, but may greatly improve the efficiency of the embedding. However, it is still limited to a certain family of L_1-like similarities. This limitation is reduced by the third algorithm, BoostPro, presented in Section 3.4.

The embedding algorithms are designed to learn from examples of similar and dissimilar pairs of examples. Moreover, we extend the algorithms, subject to certain assumptions, to the semi-supervised case when only examples of similar pairs, plus some unlabeled data points, are available.

3.1 Preliminaries

The general form of the embedding constructed by our algorithms is

$$H(\mathbf{x}) = [\alpha_1 h_1(\mathbf{x}), \ldots, \alpha_M h_M(\mathbf{x})], \tag{3.1}$$

where each dimension m is produced by a function h_m, parametrized by a *projection* $f : \mathcal{X} \to \mathbb{R}$ and a threshold $T \in \mathbb{R}$:

$$h(\mathbf{x}; \, f, T) = \begin{cases} 1 & \text{if } f(\mathbf{x}) \leq T, \\ 0 & \text{if } f(\mathbf{x}) > T. \end{cases} \tag{3.2}$$

We will simply write $h(\mathbf{x})$ when the parametrization is clear from context. This form of H is motivated by two considerations. One is the simplicity of learning: the "modular" form of H affords simple, greedy algorithms. The other is the computational complexity of the search: the L_1 distance in H is in fact a Hamming distance (perhaps weighted by αs), and its calculation can be implemented with particular efficiency.

A function h in (3.2) naturally defines a classifier $c : \mathcal{X}^2 \to \{\pm 1\}$ on pairs of examples. We will refer to such c as *simple classifier*, and omit writing the parametrization unless necessary:

$$c(\mathbf{x}, \mathbf{y}; \, f, T) = \begin{cases} +1 & \text{if } h(\mathbf{x}; \, f, T) = h(\mathbf{y}; \, f, T), \\ -1 & \text{otherwise.} \end{cases} \tag{3.3}$$

3.1.1 Threshold evaluation procedure

The embedding algorithms in this chapter differ in the form of projections f used to derive the hs, and in the way the hs are chosen. One element they all share is a procedure for evaluating, for a fixed f, the empirical performance of the simple classifiers that correspond to a set of thresholds. This procedure is given in Algorithm 4. We assume that each training pair $(\mathbf{x}_{i,1}, \mathbf{x}_{i,2})$ is assigned a non-negative weight w_i; when an algorithm involves no such weights they can be simply assumed to be all equal.

Intuitively, the motivation behind the algorithm is as follows. Our goal is to estimate, for a given T, the expected true positive rate

$$\text{TP} \triangleq E_{\mathbf{x}, \mathbf{y} | \mathcal{S}(\mathbf{x}, \mathbf{y}) = +1} \left[\Pr(h(\mathbf{x}) = h(\mathbf{y})) \right] \tag{3.4}$$

and the true false positive rate

$$\text{FP} \triangleq E_{\mathbf{x}, \mathbf{y} | \mathcal{S}(\mathbf{x}, \mathbf{y}) = -1} \left[\Pr(h(\mathbf{x}) = h(\mathbf{y})) \right], \tag{3.5}$$

with the expectations taken with respect to the joint distribution of example pairs $p(\mathbf{x}, \mathbf{y})$. In the context of retrieval, when we are conceptually considering pairing the query with every example in the database, this means the product of marginal distributions $p(\mathbf{x})p(\mathbf{y})$.

As is normally the case in machine learning, we can only estimate these quantities from the available examples of similar and dissimilar pairs.[1] The straightforward approach that we will adopt for now, is to estimate TP by the percentage of similar pairs that are *not separated* by T, i.e. pairs for which the both values fall on the same side of T.[2] Similarly, FP is estimated by measuring the percentage of dissimilar pairs not separated by T.

An implicit assumption in this estimation is that the training pairs are distributed identically and independently according to a probability law that generates the data, and therefore are equally representative. Instead, it is possible that each pair have a weight, which may be interpreted as the probability of selecting that pair; such is the situation in the context of boosting algorithms later in this chapter. The weights are easily incorporated into our empirical estimation approach: instead of the percentage of pairs separated by T, we will calculate their cumulative weight.

Algorithm 4 describes in pseudocode an efficient procedure for such estimation of the TP and FP rates for all feasible thresholds. The technique used to do this in the single pass is simple: when we form the sorted array of projection values, we record for each element $p = 1, 2$ of a pair $(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)})$ the direction $d_{i,p}$ to its counterpart within the array; e.g., if $f(\mathbf{x}_i^{(1)}) > f(\mathbf{x}_i^{(1)})$ then, after sorting by the values of $f(\mathbf{x})$, $\mathbf{x}_i^{(1)}$ will appear *after* $\mathbf{x}_i^{(2)}$. Traversing the array from the lowest to the highest value we maintain and update the cumulative weights (which is equivalent to counts, when weights are all equal) of positive and negative pairs separated by the current threshold. This is illustrated with Figure 3-1 that shows the estimated TP and FP rates for a set of five similar and five dissimilar pairs.

The set of thresholds to consider is determined by the number of unique values among the projections of the data: any two thresholds for which no data point is projected between them are not distinguishable by the algorithm. Therefore, with N training pairs we have $n \le 2N + 1$ thresholds. The sorting step dominates the complexity, since after the values $v_{i,p}$ are sorted, all thresholds are evaluated in a single pass over the sorted $2N$ records. Thus the running time of the algorithm is $O(N \log N)$.

The first algorithm we propose in this thesis is the similarity sensitive coding (SSC). It uses the procedure presented above to construct an embedding of the data into a binary space, selecting the dimensions of the embedding independently based on the estimated gap.

3.2 Similarity sensitive coding

The idea underlying the SSC algorithm is to construct an embedding similar to the one achieved in LSH, but to explicitly maximize its sensitivity to the desired similarity

---

[1] In a notation shortcut we will henceforth write TP and FP to mean these estimates, rather than the unknown true values.

[2] Which side is not important, as long as both values are on the same side; consequently, note that h is not a classifier, while c is.
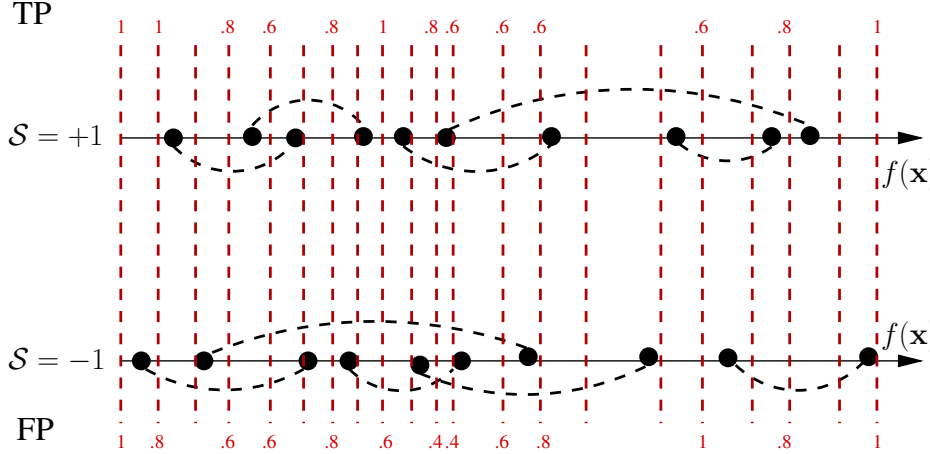
Figure 3-1: Illustration of the operation of Algorithm 4. Similar (top) and dissimilar (bottom) pairs are connected by dashed lines, and are assumed to all have equal weights of 1/10. All 21 distinct thresholds are shown; the TP (top) and FP (bottom) rates are shown only for some. The maximal attainable gap here is .4 (.e.g, with the ninth threshold from the left).

measure. The implicit assumption here is that the L_1 distance in $\mathcal{X}$ provides a reasonable foundation for modeling $\mathcal{S}$, that is in need of the following improvements:

- Some dimensions are more relevant to determining similarity than others, and thus should affect the distance more heavily.

- For a given dimension, some thresholds are more useful than others.

A pseudocode description for SSC is given in Algorithm 5. Recall the discussion in Section 2.4.2 on the role of the gap between the TP and FP rates of a binary function. SSC takes a parameter G that specifies a minimal acceptable value (lower bound) of this gap, and extracts, for each dimension of the data, all the thresholds for which the estimated TP-FP gap meets this bound.

An earlier version of this algorithm was published in [105], under the name of parameter sensitive hashing (PSH). The original name reflected the coupling of representation (a bit vector based on a set of axis-parallel stumps) and the LSH-based search, and also the implicit notion of similarity present only through the specification of pose parameters. An additional difference is in the criterion for selecting the embedding bits: in PSH, the criterion is formulated in terms of bounding the TP and FP rates separately rather than bounding the gap. Numerous experiments have confirmed since that the gap-based formulation is not only better justified theoretically but also superior in practice. Thus, SSC can be seen as a generalization and improvement of the original PSH algorithm.

In a practical implementation of Algorithm 5, one faces a number of design decision that may have a dramatic effect on the performance. Below we discuss these issues in the context of experimental evaluation on the UCI/Delve data sets. The focus here is

Algorithm 4 THRESHOLDRATE(P, f, W): Evaluation of projection thresholds given similarity-labeled examples.

Given: Set of labeled pairs $P = \{(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}, l_i)\}_{i=1}^N \subset \mathcal{X}^2 \times \{\pm 1\}$,
    where $l_i = \mathcal{S}\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}\right)$.
Given: A projection function $f : \mathcal{X} \to \mathbb{R}$.
Given: Weights $W = [w_1, \ldots, w_N]$.
Output: Set of triples $\{\langle T_t, \text{TP}_t, \text{FP}_t\rangle\}_{t=1}^n$, where TP_t and FP_t are the estimated TP and FP rates for threshold T_t.

1: Let $v_{i,p} := f\left(\mathbf{x}_i^{(p)}\right)$ for $i = 1, \ldots, N$ and $p = 1, 2$.
2: Let $u_1 < \ldots < u_{n-1}$ be the $n - 1$ unique values of $\{v_{i,p}\}$.
3: Let $\Delta_j := (u_{j+1} - u_j)/2$, for $j = 1, \ldots, n - 2$.
4: Let $T_1 := u_1 - \Delta_1$, and $T_{j+1} := u_j + \Delta_j$, for $j = 1, \ldots, n - 1$.
5: **for all** $i = 1, \ldots, N$ **do**
6:     Let $d_{i,1} := \begin{cases} +1 & \text{if } v_{i,1} \leq v_{i,2}, \\ -1 & \text{if } v_{i,1} > v_{i,2}. \end{cases}$
7:     Let $d_{i,2} := \begin{cases} +1 & \text{if } v_{i,1} > v_{i,2}, \\ -1 & \text{if } v_{i,1} \leq v_{i,2}. \end{cases}$
8: Sort records $\{\langle v_{i,p}, d_{i,p}, w_i, l_i\rangle\}_{i=1,\ldots,N,\, p=1,2}$ by the values of $v_{i,p}$.
9: Normalize w_i so that $\sum_{l_i=+1} w_i = 1$, $\sum_{l_i=-1} w_i = 1$.
10: **for all** $j = 1, \ldots, t$ **do**
11:     Let $i_j := \max\{i : v_i \leq T_j\}$
12:     $\text{TP}_j := 1 - \sum_{i \leq i_j, l_i=+1} w_i d_i$.
13:     $\text{FP}_j := 1 - \sum_{i \leq i_j, l_i=-1} w_i d_i$.

---

on questions arising directly in the implementation of SSC. Other important issues, such as how the similarity labels are obtained, are discussed elsewhere.

3.2.1   Benchmark data sets

Throughout this chapter we will refer to experiments on a number of data sets. The learning problems associated with these data sets are of the type for which we expect our algorithms to be particularly useful: regression or classification with a large number of classes.

The purpose of these experiments is two-fold. One is to illustrate the principles underlying the new algorithms. The data sets vary in size and difficulty, but most of them are small enough (both in number of examples and in dimension) to allow a rather thorough examination of the effect of various settings.

The other purpose is to evaluate the impact of our algorithms outside of the computer vision domain, on "generic" data sets, familiar to the machine learning community from their use as benchmarks. From a practitioner's perspective, this means evaluating what does one gain, if at all, from using a model of similarity learned for the task at hand, in comparison to the standard use of distances in the

---

Algorithm 5 SSC(P, g): Similarity sensitive coding by selecting thresholds on original dimensions.

Given: Set of similarity-labeled pairs $P = \{(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}), l_i\}_{i=1}^N \subset \mathcal{X}^2 \times \{\pm 1\}$,
    where $l_i = \mathcal{S}\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}\right)$.

Given: Lower bound on TP-FP gap $G \in (0, 1)$.

Output: Embedding $H^{\mathrm{SSC}} : \mathcal{X} \to \{0, 1\}^M$ (M to be determined by the algorithm).

1: Let $M := 0$.
2: Assign equal weights $W(i) = 1/N$ to all N pairs in P.
3: **for all** $d = 1, \ldots, dim(\mathcal{X})$ **do**
4:     Let $f(\mathbf{x}) \equiv \mathbf{x}_{(d)}$.
5:     Apply THRESHOLDRATE(P, f, W) to obtain a set of n thresholds $\{T_t^d\}_{t=1}^n$ and associated TP and FP rates $\{\mathrm{TP}_t^d\}, \{\mathrm{FP}_t^d\}$.
6:     **for all** $t = 1, \ldots, n$ **do**
7:        **if** $\mathrm{TP}_t^d - \mathrm{FP}_t^d \geq G$ **then**
8:           Let $M := M + 1$.
9:           $h_M(\mathbf{x}) \triangleq h(\mathbf{x}; f, T_t^d, 1)$ {as in (3.2).}
10: Let $H^{\mathrm{SSC}} \triangleq \mathbf{x} \to [h_1(\mathbf{x}), \ldots, h_M(\mathbf{x})]$

---

data space. Depending on the precise goals of an application, this effect can be measured in terms of ROC curve behavior, or in terms of the regression/classification error obtained by an example-based method that uses the similarity model.

The data sets are publicly available and come from a variety of domains. Below we give a brief description of each set; the important statistics are summarized in Table 3.1. Recall that r (given in the last column of Table 3.1) is the threshold used to define a label-induced similarity in our experiments, as explained in Section 2.2.1, on the distance in the labels, such that $\mathcal{D}_{\mathcal{Y}}(y_i, y_j) \leq r \Leftrightarrow \mathcal{S}(\mathbf{x}_i, \mathbf{x}_j) = +1$. For classification problems, a natural value of r is 0, i.e. two examples are similar if and only if they belong to the same class.

For regression the choice should be determined by the desired sensitivity of the estimator and by the effect on the resulting similarity model. In our experiments, we have set r based on a "rule of thumb" defined by two criteria: choose a value that does not exceed half of the mean error obtainable by the standard (published) regression algorithms, and that keeps the proportion of similar pairs out of all pairs below 10% (these two criteria "pull" the value of r in different directions.) A more thorough approach would involve optimizing the value of r by cross-validation or holdout procedure: repeating the entire experiment of learning an embedding and evaluating the NN estimator on this embedding, for a range of values of r. Such procedure would likely improve the results.

Auto-MPG   Predicting mileage per gallon of fuel from various mechanical characteristics of a vehicle.

Name	Source	Dimension	# of examples	Task	Label span	r
MPG	[16]	7	392	Regression	37.6	1
CPU	[1]	21	8192	Regression	99.0	1
Housing	[16]	13	506	Regression	45.0	1
Abalone	[16]	7	4177	Regression	28.0	1
Census	[1]	8	22784	Regression	5×10^5	500
Letter	[1]	16	20000	Classification	$1, \dots 26$	0
Isolet	[16]	617	3899	Classification	$1, \dots 26$	0

Table 3.1: Summary of the data sets used in the evaluation.

Machine CPU Regression: predicting time spent by a program in user CPU mode from process statistics: number of system calls, page faults, I/O etc.

Boston Housing Regression: predicting median value of housing in Boston neighborhoods as a function of various demographic and economic parameters.

Abalone Regression: predicting the age of abalone from physical measurements.

US Census Regression: predicting median price of housing based on neighborhood statistics.

Letter Classification of written letters from a set of image statistics; 26 classes (one per letter.)

Isolet Classification of spoken isolated letters (by a number of speakers) from a set of features of the recorded acoustic signal. There are 26 classes (one per letter.) Only half of the available 7797 examples were used to speed up experiments.

3.2.2 Performance and analysis of SSC

We have evaluated the performance of SSC on the seven data sets introduced in Section 3.2.1. The results were obtained using ten-fold cross-validation: each data set was randomly divided into ten disjoint parts roughly equal in size, and each part was used as a test set while the remaining 9/10 of the data served as training set. All the data were encoded using the SSC embedding learned on that training set, and then the prediction error was measured for the examples in the test set using the L_1 distance in the embedding space (with SSC embedding this is the same as Hamming distance) to determine similarity.

Two parameters have to be set in this process. One is the minimal gap G. We chose it from a range of values between 0.01 and 0.25 by leave-one-out cross validation on training data in each experiment. For each data set and in each "fold" of the ten-fold cross validation, we encode the training data (9/10 of the total data) using SSC with each gap value under consideration, and compute the mean absolute

training error of example-based estimation with that encoding. That is, we predict the value of the label for each training point using its neighbors (but not itself) in the embedded training set. We then select the gap value which produced the lowest training error, and use it to compute the testing error in that fold of cross validation. In our experiments we found that the gap value resulting from this tuning procedure is very stable, and typically is the same for most of the ten folds in any data set; these typical values are shown in the second to last column of Table 3.2.

The second parameter is the K (or R) in the eventual regression/classification algorithm. Virtually all published results on these data sets refer to K-NN algorithms, hence we also used K-NN, choosing K from a range between 1 and 300 by a procedure identical to the one for setting g.[3]

As a baseline, we compare the results obtained with SSC to those obtained with the standard nearest-neighbor regression estimation, using L_1 distance between the examples as a proxy for similarity. Tables 3.3 and 3.5 show the results of this comparison for regression databases. In terms of the mean absolute error (MAE), there is a general trend of SSC outperforming the L_1. On two datasets the differences between the means are farther than two standard deviations apart, while for others the difference is less significant. In terms of the mean squared error, the two methods achieve qualitatively similar performances. This suggests that the error with SSC is often smaller than that with L_1, but occasionally it becomes very high due to a spurious match. The performance of SSC on classification data sets, compared to the L_1, is similarly good, as evident from Table 3.4.

As mentioned in Chapter 1, another measure of the performance of a similarity model is its direct effect on retrieval accuracy. Figures 3-6-3-12 show the plots of the ROC curves for L_1 and SSC on the seven benchmark datasets. In six out of seven datasets, the curves for SSC (blue, dashed) are clearly above that for L_1 (black, dotted). The average gain in the area under curve (AUC) is between .05 and .1. The only data set in which no gain was recorded is Isolet. The dimension of that data set is significantly higher than the dimensions of the remaining six, and we believe that this fact partially accounts for the difficulty of SSC. There is a very high number of thresholds in general for this data set (i.e., the length of the unary encoding is very high, see Table 3.2) and of the thresholds that attain the desired gap value, in particular. Thus, in training SSC, we randomly selected 4,000 out of more than 250,000 thresholds with the gap above 0.1. That step, dictated by computational necessity, may have removed significant some useful thresholds from the code and hampered its retrieval accuracy.

Distribution of the TP-FP gap

An immediate effect of the value of G is on the value M, the number of selected bits. Setting G too high will result in failure to construct an embedding; setting it too high will result in an embedding with a huge number of bits, not only not efficient but also

---

[3]More precisely, we optimized g and K jointly, by evaluating on the training data a range of K for each embedding obtained with a particular g, and choosing the "winning" combination for each of the ten cross-validation folds.

impractical due to the required storage space. Figure 3-2 shows, for four datasets, how the number of accepted thresholds (pooled over all dimensions) declines as the lower bound on the gap increases.
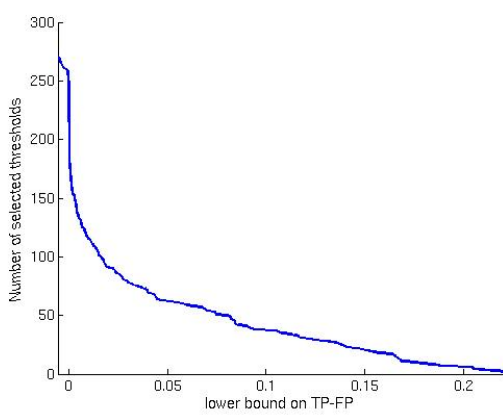
(a) Letter, dimension 4
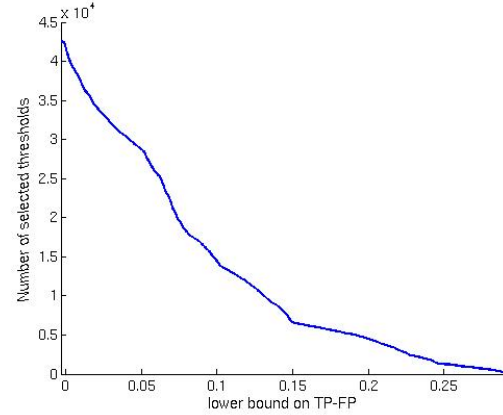
(b) CPU, dimension 11
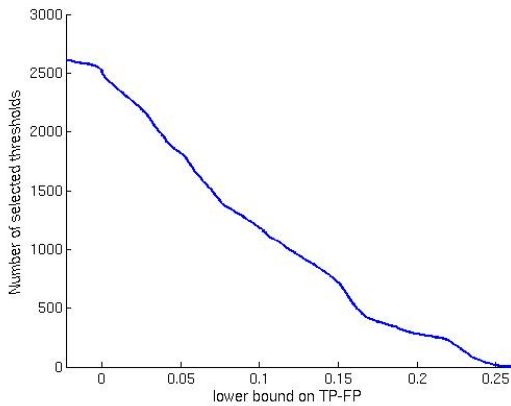
(c) Boston housing, dimension 10
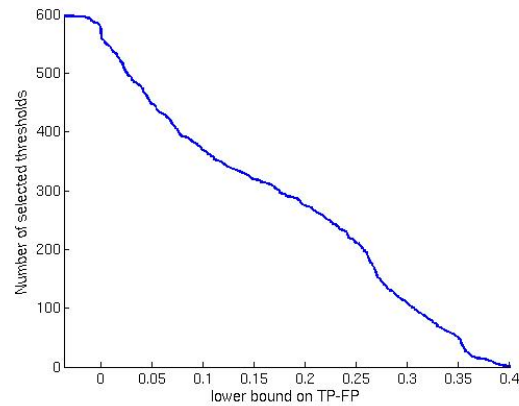
(d) Auto-MPG, dimension 3

Figure 3-2: The distribution of TP-FP gap values for four data sets (pooled over all dimensions.)

Figure 3-3 shows some typical examples of the behavior of TP and FP rates and the gap between them (for the same cases used in Figure 3-2.) As may be expected, the general trend is that a threshold with higher TP rate typically will also have a higher FP. This is because thresholds with high TP rates simply lie close to the median of the projection (dimension) values, and thus are likely to separate many pairs–similar and dissimilar. In that way, the selection procedure is guided by the statistics of the data.

One other observation from Figure 3-3 is that the false positive rates appear to be bounded from below at around 1/2. We will discuss this phenomenon and its implications in Section 3.2.4.
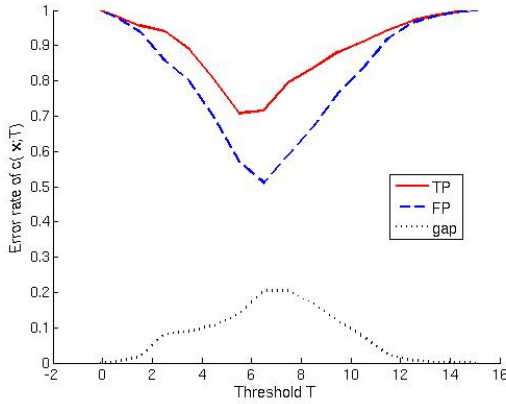
(a) Letter, dimension 4
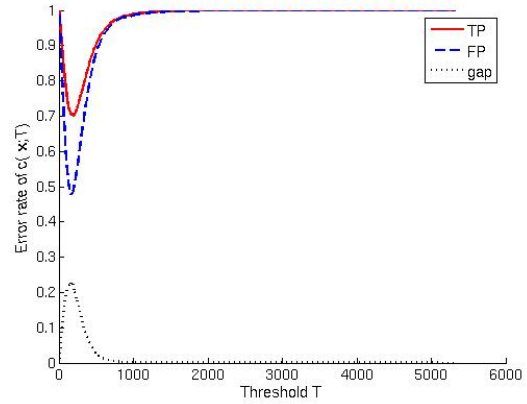
(b) CPU, dimension 11
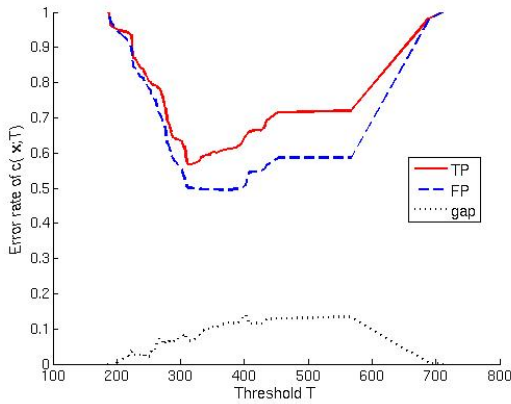
(c) Boston housing, dimension 10
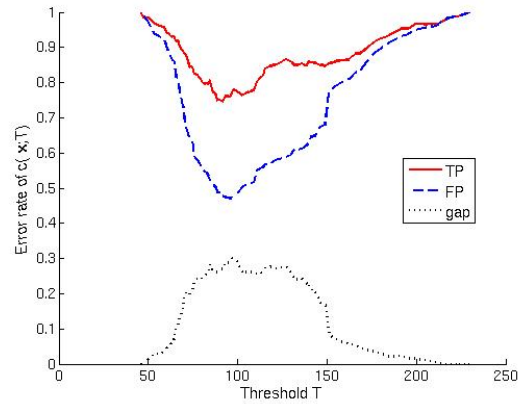
(d) Auto-MPG, dimension 3

Figure 3-3: The distribution of TP and FP rates for a single dimension, for four data sets. Solid (red): TP; dashed (blue): FP; dotted (black): the gap.

3.2.3 The coding perspective on SSC

From a machine learning standpoint, SSC can be seen as a mechanism of directly selecting *features* from a very large but finite pool, consisting of all the distinct functions h (i.e., all the bits in the unary encoding of the data). In terms customary in machine learning literature, this is a *filter* selector: the criteria for selecting or rejecting a feature are based on the feature's parameters–the performance of the associated simple classifier. That is in contrast to *wrapper* selection, whereby the features are evaluated by "plugging them in" to the classifier.[4]

The embedding H^{SSC} can be also interpreted as *encoding* examples in $\mathcal{X}$ with an M-bit code, which is constructed with the objective to retain maximum information relevant to similarity between examples. (A similar interpretation of similarity fea-

---

[4]The greedy algorithm presented in Section 3.3 is an example of a wrapper feature selection.

Data set	Optimized	Nominal	Unary	M	(gap G)	Compression
MPG	39	152	4672	371 ± 7	.1	0.9206
CPU	220	625	7136969	6864 ± 308	.15	0.9990
Housing	91	385	4612045	673 ± 57	.15	0.9999
Abalone	64	224	12640	3007 ± 25	.1	0.7621
Census	107	256	58564303	3438 ± 1481	.1	0.9999
Letter	64	128	240	37 ± 0	.1	0.8458
Isolet	6844	19744	5096373	178116 ± 6948	.15	0.9651

Table 3.2: Comparison of the SSC length M to original representation. Optimized: number of bits necessary to encode the unique values. Nominal: number of bits necessary to encode $N \times \dim(\mathcal{X})$ values in a N-point data set with no compression. Unary: length of unary encoding after conversion of the data to integers (see footnote 5). Compression: the percentage of the unary encoding bits effectively eliminated by SSC.

tures has been discussed in [97], in the context of binary classification problems.) It is interesting to compare M to the length of the original representation. In terms of the "nominal" number of dimensions, M is typically higher(as evident in Table 3.2) than $\dim(\mathcal{X})$. However, the effective representation that SSC is implicitly compressing is the unary encoding (see discussion in Section 2.4.2.) With respect to the unary encoding,[5] SSC is achieving considerable compression, as shown in the right column of Table 3.2. The numbers refer to the percentage of unary encoding bits that are left out of the SSC encoding (i.e., 90% compression means 90% reduction in encoding length.) The selection procedure in SSC can therefore be seen as a *dimensionality reduction* in the unary encoding space, with the objective to preserve the dimensions most relevant to similarity judgments.

Besides examining the number of bits in the code, of course, we must also look at the redundancy. It should come as no surprise that the code obtained with SSC is terribly redundant. Figure 3-4 visualizes the covariance matrices for the SSC bits for three of the data sets (these are typical covariance matrices), with red corresponding to higher values. One source for this redundancy is trivial: if two thresholds T_1 and T_2 are close (relative to the span of $f(\mathbf{x})$), the values of $h(\mathbf{x}; f, T_1)$ and $h(\mathbf{x}; f, T_2)$ will be highly correlated. A less trivial source of correlation is the structure in the data, which may include various dependencies between values and carry on to the thresholded projections.

3.2.4 Semi-supervised learning

In Section 3.2.2 we noted that the false positive rate of the stumps in our experiments appears to be bounded from below by 1/2. This has the following explanation. Suppose that similarity is a very rare event, in the sense that for two random examples

---

[5] Recall that for integers, the length of the unary encoding is simply the span of the values. For a set of values $v_1, \ldots, v_n$ some of which are non-integers, it was calculated as $\max\{v_i\} \cdot 1/\min_{i,j}\{|v_i - v_j|\}$.

(a) CPU	(b) Letter	(c) Housing
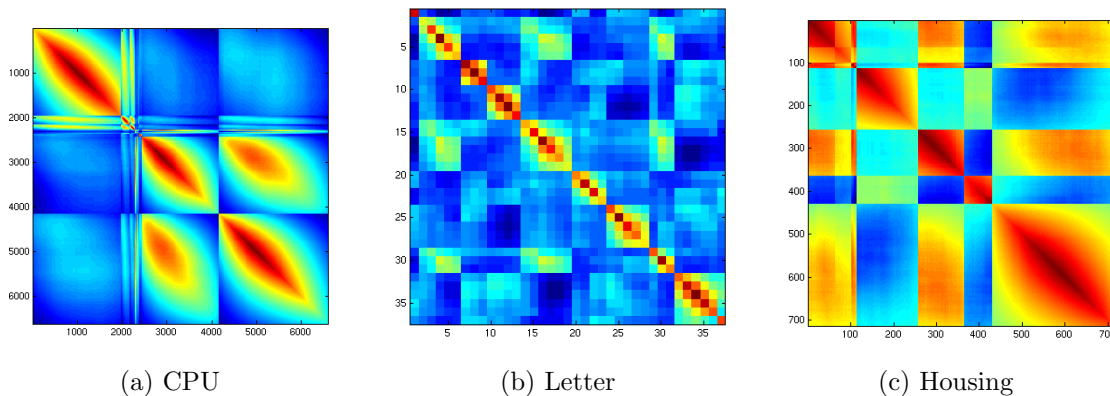
Figure 3-4: Covariances of the SSC bits for three of the data sets. Red values are high, blue values are low. The large blocks correspond to the original dimensions of the data; the peaks of covariance values are around the medians of projections. Refer to Tables 3.1 and 3.2 for details about the data sets and the embeddings.

drawn from the domain at hand, the probability of them being similar is low. This is certainly the case for many interesting applications in computer vision. For instance, two randomly selected images of people are unlikely to contain similar articulated poses, and two regions randomly extracted from natural images are unlikely to be visually similar.[6] This is in fact the case in the UCI/Delve data sets used in our experiments; the average *similarity rate* (the probability of a random pair to be similar) ranges from 0.03 to 0.1 (with the exception of Abalone for which it is 0.3.)

Let us consider the distribution of the values of $f(\mathbf{x})$ for similar and dissimilar pairs of examples in $\mathcal{X}$. The underlying assumption of our approach is that, if f is a "useful" projection, there is a structure in the distribution of these values, namely, that similar pairs tend to have similar values of f. On the other hand, under our assumption that the similarity rate is significantly lower than $1/2$, the set of all dissimilar pairs is close to simply the set of *all* pairs in $\mathcal{X}^2$. That means that

$$p(f(\mathbf{x}_1), f(\mathbf{x}_2) \mid \mathcal{S}(\mathbf{x}_1, \mathbf{x}_2) = -1) \approx p(f(\mathbf{x}_1), f(\mathbf{x}_2)) = p(f(\mathbf{x}_1))p(f(\mathbf{x}_2)), \quad (3.6)$$

i.e. the joint distribution of the pairs of projections $(f(\mathbf{x}), f(\mathbf{y}))$ for $\mathcal{S}(\mathbf{x}, \mathbf{y}) = -1$ is close to the unconditional joint distribution over all pairs.[7] (The second equality is due to the assumption that examples are provided to us i.i.d.)

We can then model the process that generates negative examples for similarity learning by the following trivial procedure: *take a random pair of examples and label it as dissimilar.* This of course will produce some noise in the labels - at the rate equal to similarity rate of the data set. In fact the natural procedure to create a set

---

[6]This may not be true if the notion of similarity is defined coarsely, e.g. if any two people standing upright are considered in similar pose. But we will assume that the similarity is sufficiently fine, as seems to be the case in most interesting problems.

[7]It should be clear that we are referring to distribution of $f(\mathbf{x})$, not that of $\mathbf{x}$.

of dissimilar pairs, and the one we used in all the experiments reported in this thesis, is in fact almost as described above, with the additional pass to remove any spurious similar pairs.

The consequence of (3.6) is that, for a low similarity rate ρ, the FP rate of a feature $h(\mathbf{x}; f, T)$ is bounded from below by a value close to $1/2$. The following proof has been given in [70], and is augmented here to take into account the correction by ρ. Suppose that we draw a random pair of examples $(\mathbf{x}_1, \mathbf{x}_2)$ from the data distribution $p(\mathbf{x})$ and project them using f. Let π_T be the probability mass of $f(\mathbf{x})$ below the threshold T:

$$\pi_T = \Pr(f(\mathbf{x}) \le T)$$

Since the randomly constructed pair $(\mathbf{x}_1, \mathbf{x}_2)$ is assumed to be dissimilar, a "bad" event, from the perspective of classifying similarity, occurs when $f(\mathbf{x}_1)$ and $f(\mathbf{x}_2)$ are on the same side of T on the line $f(\mathbf{x})$. The probability of such an event is $\pi_T^2 + (1 - \pi_T)^2$. By definition of ρ the random pair $(\mathbf{x}_1, \mathbf{x}_2)$ is dissimilar with probability $1 - \rho$; therefore, the expected FP rate of $h(\mathbf{x}; f, T)$ is

$$\mathrm{FP}(f, T) = (1 - \rho) \left(\pi_T^2 + (1 - \pi_T)^2 \right). \tag{3.7}$$

Note that π_T (cdf of a scalar random variable) can be easily and robustly estimated from the data, even with a relatively modest number of examples. This means that in order to estimate the FP rate of a threshold, we do not need explicit examples of dissimilar pairs if we have access to a set of unlabeled (single, not paired) data points. We will refer to such a setup as *semi-supervised*.[8] The threshold evaluation procedure in Algorithm 4 is easily modified for the semi-supervised case, as described in Algorithm 6.

In the remainder of this thesis, we will consider both supervised and semi-supervised setup when discussing the embedding algorithms.

3.2.5   Limitations of SSC

In the experiments described above, we have seen that SSC is able to improve over the "off-the-shelf" distance measure, both in terms of the prediction accuracy with example-based methods that rely on it and in terms of the accuracy of similarity detection, as expressed in the ROC curves. However, we also have pointed to a number of problems with the embeddings constructed with SSC. These problems are rooted in two main sources:

Constrained geometry   SSC provides a refinement on the L_1 distance better tuned to the target similarity, but the reliance on axis-parallel projections limits the resulting similarity concept to the class of hyper-rectangles in the unary encoding space.

---

[8]This may seem somewhat different from the common use of the term "semi-supervised" to mean that only part of the available data is labeled. To reconcile that with our use, consider that with N examples, we essentially operate on the set of $N(N - 1)/2$ pairs, only a small fraction of which are labeled (all positive), and the rest are given implicitly with no labels.

Algorithm 6 Semi-supervised procedure for evaluating threshold. See Section 3.2.4 for details.

Given: Data set $X = [\mathbf{x}_1, \ldots, \mathbf{x}_N] \subset \mathcal{X}$.

Given: Set of similar pairs $P^+ = \{(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)})\}_{i=1}^{N_p} \subset \mathcal{X}^2$.

Given: Projection function $f : \mathcal{X} \to \mathbb{R}$

Given: Weights on pairs $W = [w_1, \ldots, w_{N_p}]$, such that $\sum_i w_i = 1$.

Given: Weights on points $S = [s_1, \ldots, s_N]$, such that $\sum_j s_j = 1$.

Output: Set of triples $\{\langle T_t, \mathrm{TP}_t, \mathrm{FP}_t \rangle\}_{t=1}^n$, where TP_i and FP_i are the estimated TP and FP rates for threshold T.

1: Let $u_1 < \ldots < u_{n-1}$ be the unique values of $f\{x_i\}_{i=1}^N$.
2: Set thresholds $T_1 < \ldots < T_n$ based on $\{u_i\}$, like in Algorithm 4.
3: **for all** $i = 1, \ldots, N$ **do**
4:     Obtain list of records $\{\langle v_{i,p}, d_{i,p}, w_i \rangle\}_{i=1,\ldots,N_p,\,p=1,2}$ sorted by $v_{i,p}$, like in Algorithm 4, but using only similar pairs in P^+
5:     **for all** $j = 1, \ldots, n$ **do**
6:         Let $i_j := \max\{i : v_i \le T_j\}$
7:         $\mathrm{TP}_j := 1 - \sum_{i \le i_j} w_i d_i$.
8:         Let $\pi_j = \sum_{i:\, f(\mathbf{x}_i) \le T_j} s_i$.
9:         $\mathrm{FP}_j := \pi_j^2 + (1 - \pi_j)^2$.

Ignoring dependencies Treating features h individually leads to redundancy in the embedding, sometimes at the cost of performance. Although some ad-hoc methods for alleviating this (such as checking for correlation with already selected thresholds) may help, we would like to have a more direct method to limit unnecessary dependencies and to optimize the entire embedding rather than individual dimensions.

These issues are addressed in the improved versions of this basic similarity embedding algorithm, which we present next. The first of them enhances SSC by replacing independent selection of embedding bits with a greedy, sequential optimization procedure based on boosting.

3.3 Ensemble embedding with AdaBoost

Recall that for each thresholded projection h (3.2) there is a dual classifier of example pairs c (3.3). Let us now consider the M-bit SSC embedding $H = [h_1, \ldots, h_M]$, and suppose that for some $\mathbf{x}, \mathbf{y} \in \mathcal{X}$ the distance $\|H(\mathbf{x}) - H(\mathbf{y})\| = R$. Since each embedding dimension contributes either 0 or 1 to the distance, this means that values at exactly R positions in the two embeddings are different. Consequently, exactly R associated classifiers would assign $\widehat{\mathcal{S}}(\mathbf{x}, \mathbf{y}) = -1$. Generally, we can write

$$\|H(\mathbf{x}) - H(\mathbf{y})\| = \frac{M}{2} - \sum_{m=1}^M \frac{1}{2} c_m(\mathbf{x}, \mathbf{y}), \qquad (3.8)$$

so that the distance assumes values between 0 and M.

In the more general form, the contribution of a thresholded projection h_m to the distance is weighted and is either 0 or α_m. This corresponds to assigning a *vote* of $\alpha_m/2$ to the classifier c_m in (3.8). Together, the M thresholded projections form the similarity classifier

$$C(\mathbf{x}, \mathbf{y}) \;=\; \text{sgn}\left(\sum_{m=1}^{M} \alpha_m c_m(\mathbf{x}, \mathbf{y})\right). \tag{3.9}$$

This is an *ensemble classifier*.[9] A feasible strategy for constructing an embedding H is therefore to construct an ensemble C coupled with the threshold τ by a procedure that minimizes the empirical risk on the training pairs. We will follow this strategy and use the boosting approach [99, 23]. Boosting is essentially a procedure for greedy assembly of C in a way that reduces the training error. It has also been shown to yield excellent generalization performance. Before we describe how the boosting framework can be applied in our task, we review it in the next section.

3.3.1 Boosting

We will follow the generalized view of AdaBoost, given in [100], since it will simplify the transition to improved versions of our algorithm. Let $X = \mathbf{x}_1, \ldots, \mathbf{x}_N$ be the N training examples labeled by $l_1, \ldots, l_N \in \{\pm 1\}$. In boosting it is assumed that there exists a *weak learner* that can, given a set of labeled training examples and a *distribution* (set of non-negative weights that sum to one) W, obtain a weak hypothesis $c(\mathbf{x})$ whose training error on X, weighted by W, is better than chance $(1/2)$. The goal of boosting is to construct an ensemble classifier

$$H(\mathbf{x}) = \text{sgn}\left(\sum_{m=1}^{M} \alpha_m c_m(\mathbf{x})\right), \tag{3.10}$$

that minimizes training error. Note that (3.10) implicitly assumes thresholding at zero (i.e. classifying by a weighted majority). A different threshold may be introduced post-training and set to reach the desirable ROC point.[10]

Finding the ensemble that attains the global minimum of training error is computationally infeasible. Instead, AdaBoost gives an iterative greedy algorithm that adds weak classifiers c_m with an appropriate vote α_m one at a time. Throughout the iterations AdaBoost maintains a distribution W; we will denote by $W_m(i)$ the weight on the i-th example before iteration m. The distribution is updated so that, intuitively, examples classified correctly in an iteration have their weight reduced, and those misclassified have their weight increased (thus "steering" the weak learner towards themselves by increasing the cost of further misclassifying them).

The magnitude of change in iteration m is determined by the vote α_m; the update

---

[9]Instead of thresholding the sum of votes at zero in (3.9), a different value of the threshold may be introduced by adding a "dummy" classifier which always outputs, say, +1, and setting its vote to the desired threshold value.

[10]Or, alternatively, by including a fixed-output weak classifier in the ensemble, similarly to the "bias" input cell in neural networks.

rule in AdaBoost is

$$W_{m+1}(i) := W_m(i) \exp(-\alpha_m l_m c_m(\mathbf{x}_i)) / Z_m^{AB}, \tag{3.11}$$

with division by the normalization constant

$$Z_m^{AB} \triangleq \sum_{i=1}^{N} W_m(i) \exp(-\alpha_m l_m c_m(\mathbf{x}_i)) \tag{3.12}$$

ensuring that W_{m+1} remains a distribution in the sense defined above.

In addition to Z_m^{AB}, another key quantity in the analysis of boosting is the weighted correlation of labels with predictions

$$r_m^{AB} \triangleq \sum_{i=1}^{N} W_m(i) l_i c_m(\mathbf{x}_i). \tag{3.13}$$

It can be shown [100] that a reasonable objective of the weak learner at iteration m is to maximize r_m^{AB}. Furthermore, the training error of H after m iterations can be shown to be bounded from above by $\prod_{t=1}^{m} Z_t^{AB}$; minimizing Z_m^{AB} in each iteration is therefore a reasonable objective of the greedy algorithm. Once the weak classifier c_m has been selected, Z_m^{AB} is affected only by α_m, so that this objective is translated to setting α appropriately. When the range of c_m is $[-1, +1]$, the rule

$$\alpha_m := \frac{1}{2} \log \frac{1 + r_m^{AB}}{1 - r_m^{AB}} \tag{3.14}$$

can be shown to achieve that goal of minimizing Z_m^{AB} [100]. In a more general framework, the optimal α can be found by numerical optimization of (an easy procedure since Z can be shown to be convex and have a unique minimum.)

3.3.2   Supervised boosted SSC

Algorithm 7 is a straightforward application of AdaBoost to the problem of classifying pairs for similarity. Namely, the training examples in our case are *pairs*, and the weak classifiers here are thresholded projections that assign a positive or negative labels to a pair. The true label l_i of a pair $(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)})$ correspond to the underlying similarity $\mathcal{S}(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)})$.

To calculate the objective in iteration m, we collect the positive terms in (3.13), $\mathrm{TP}_d^j + W^n - \mathrm{FP}_d^j$, and the negative terms $-(\mathrm{FP}_d^j + W^p - \mathrm{TP}_d^j)$; summation of these yields the expression in step 7 of Algorithm 7. The calculation of α_m in step 9 is done by minimizing the Z_m^{AB}, following the bisection search procedure suggested in [100].[11]

---

[11]Briefly, we start with an initial guess for an interval that contains the optimal α, and evaluate the derivative $\partial Z_m / \partial \alpha_m$ at the endpoints as well as in the middle; since the derivative does not change the sign, and we are looking for its single zero-crossing, we then repeat, recursively, on the half of the interval that has opposing signs of $\partial Z_m / \partial \alpha_m$ at its endpoints.

Algorithm 7 Boosted SSC (supervised). Note: this is a direct application of the AdaBoost algorithm.

Given: A set of pairs $P\{\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}\}_{i=1}^N$, labeled by $l_i = \mathcal{S}(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)})$.
Output: A set of functions $h_m : \mathcal{X} \to \{0, \alpha_m\}$, $m = 1, \ldots, M$.
 1: Set initial set of weights W_1, $w_1(i) = 1/N$.
 2: **for all** $m = 1, \ldots, M$ **do**
 3:     Let $W^p := \sum_{i:l_i=+1} W_m(i)$, $W^n := \sum_{i:l_i=-1} W_m(i)$.
 4:     **for all** $d = 1, \ldots, dim(\mathcal{X})$ **do**
 5:         Let $f_d(\mathbf{x}) \equiv \mathbf{x}_{(d)}$.
 6:         For each feasible threshold T_d^j on f_d, $j = 1, \ldots, n_d$, compute TP_d^j and FP_d^j using THRESHOLDRATE(P, f_d, W_m).
 7:         Let $r_m^{(AB)}(T_d^j) := 2(\mathrm{TP}_d^j - \mathrm{FP}_d^j) + W^n - W^p$.
 8:     Select $T_m := \mathrm{argmax}_{d,j}\, r_m^{(AB)}(T_d^j)$.
 9:     Set α_m to minimize $Z_m(\alpha)$ (see text).
10:     If $\alpha_m \le 0$, stop.
11:     Update weights according to (3.11)

The boosted version differs from the original SSC algorithm in a number of ways. First, it replaces the exhaustive collection of features with large TP-FP gap in SSC by an optimization step that selects, at iteration m, a single feature maximizing r_m. Second, it incorporates the votes α_m, so that the embedding it produces is $H(\mathbf{x}) = [\alpha_1 h_1(\mathbf{x}), \ldots, \alpha_m h_m(\mathbf{x})]$. As a result, the embedding space becomes a weighted Hamming space: the L_1 distances there are measured by

$$\|H(\mathbf{x}) - H(\mathbf{y})\| = \sum_{m=1}^M \alpha_m |h_m(\mathbf{x}) - h_m(\mathbf{y})|$$

It is interesting to note the interaction of the type of weak learner we have chosen and the specific nature of the task. The objective r_m of the weak learner, expressed in (3.13), can be decomposed into two terms. One term, $\sum_{i:l_i=+1} W_m(i)c_m(\mathbf{x}_i)$ penalizes any positive pair divided by h_m. The influence of this term "pulls" the thresholds, for any projection f, away from the median of that projection, since that reduces the probability of crossing any positive pairs.

The second term $-\sum_{i:l_i=-1} W_m(i)c_m(\mathbf{x}_i)$ penalizes the negative pairs that are *not* divided, and its influence is exactly opposite: it encourages thresholds as close to the median as possible, since then minimal number of negative pairs are misclassified (and that still is about one half). This situation is different from typical classification tasks, where the classes "work together" to optimize the decision boundaries. In addition, the examples in the negative class are significantly more difficult to classify consistently: a positive pair is likely to be repeatedly labeled correctly by the weak classifiers, while a negative pair is likely to get misclassified with high probability in any given iteration.[12] The training error rates on the two classes in a typical run of

---

[12]Yet another insight into this behavior can be obtained by realizing that it is trivial to produce

63

the algorithm reflect this: the training error on the similar pairs rapidly goes down and usually reaches zero after relatively few iterations, while the training rate on the negative examples goes up and eventually reaches 1. This makes it important to find the correct threshold on the Hamming distance in H, based on the ROC curve obtained on training data (or, if possible, on a held out validation set).

Nevertheless, this algorithm may be successfully used for complicated problems such as the task of learning similarity of human silhouettes, as described in Chapter 5.

3.3.3 Boosting in a semi-supervised setup

When only examples of similar pairs are specified in addition to the unlabeled data, as describe in Section 3.2.4, the boosting algorithm needs a modification, which is described in this section.

We maintain a distribution $W_m(i)$ for $i = 1, \ldots, N_p$ where N_p is the number of positive pairs. W_m plays essentially the same role as it did in AdaBoost, and is updated in the usual way, except that the normalization constant Z_m is set to make $\sum_i W_{m+1}(i) = 1/2$.

We also maintain a second distribution $S_m(j)$ on the unlabeled examples $\mathbf{x}_j$, $j = 1, \ldots, N$. Before we present the update rule for S_j, let us consider the role played by the unlabeled examples. Intuitively, an example $\mathbf{x}_j$ serves as a *representative* of all the possible pairs $(\mathbf{x}_j, \mathbf{y})$ that can be constructed. As we have seen in Section 3.2.4, if the similarity rate is low we may assume that most of these pairs are dissimilar, and at least half (usually much more) of these pairs will be misclassified by any $c_m(\mathbf{x}, \mathbf{y}; f, T)$. That number as we have seen depends on the probability mass $\pi_m = \Pr(f(\mathbf{x}) \le T)$. Specifically, the probability of a random pair formed with $\mathbf{x}_j$ to be misclassified by a threshold T on a projection f is

$$P_j \triangleq h(\mathbf{x}_j; f, T)\pi_m + (1 - h(\mathbf{x}_j; f, T))(1 - \pi_m). \tag{3.15}$$

The expected value returned by the classifier c_m on a pair formed with $\mathbf{x}_j$ is therefore

$$P_j \cdot (+1) + (1 - P_j) \cdot (-1) = 2P_j - 1.$$

Consequently, we change the definition of r_m from (3.13):

$$
\begin{aligned}
r_m &\triangleq \sum_{i=1}^{N_p} W_m(i)c_m(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}) - \sum_{j=1}^{N} S_m(j)E_{\mathbf{y}}\left[c_m(\mathbf{x}_j, \mathbf{y})\right] \\
&= \sum_{i=1}^{N_p} W_m(i)c_m(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}) - \sum_{j=1}^{N} S_m(2P_j - 1).
\end{aligned}
\tag{3.16}
$$

The update rule for S_j changes accordingly; instead of having a deterministically

---

a threshold that will classify *all* positive training examples correctly, but as we have shown it is impossible to do much better than chance on the negative examples.

computed value of c_m in the exponent, we use the expected value, which yields

$$S_{m+1}(j) := S_m(j) \cdot \exp\left(\alpha_m(2P_j - 1)\right) / Z_s \tag{3.17}$$

with the normalization constant $Z_s = \sum_j S_m(j) \exp(\alpha_m(2P_j - 1))$. This implies that

- When $f(\mathbf{x}_j)$ falls on the side of the threshold with small probability mass, its weight goes down.

- When $f(\mathbf{x}_j)$ falls on the side with large probability mass, its weight goes up. Intuitively this encourages the algorithm to choose next threshold which will place this example on the "good" side (with small probability mass).

- If π_i is 1/2, the weights do not change (that is the "ideal threshold").

3.4   BoostPro: boosting general projections

The learning framework presented above has been thus far limited to selection and combination of features from a finite set: axis-parallel stumps (we have shown that this is equivalent to selection of bit features from the unary encoding). This makes the learning simple, but at the same time may limit the power of the resulting encoding.

The following "toy" example clearly demonstrates the limits imposed by a commitment to axis-parallel features. Consider the 2D Euclidean space, in which we have two similarity concepts. The first concept, $\mathcal{S}_A$, the *angle similarity*, is determined by the slopes of straight lines passing through the origin and the points; if the angle between the two lines is less than 5 degrees, the two points are similar. The second concept $\mathcal{S}_N$, the *norm similarity*, relies on the Euclidean norm of the points (i.e., their distance from the origin): if the L_2 norms of two points differ by less than 1/4, they are considered similar under $\mathcal{S}_N$. Figure 3-5 illustrates this, by showing, for a fixed reference data set and two query points denoted by circles, the set of reference points similar to the queries under each of the two concepts. The figure also shows the *similarity region*: the set of all points on the 2D plane that would be judged similar to a query. While empirical performance of a similarity model is determined in terms of the precision/recall measured on a particular data set, its generalization performance may be evaluated by measuring the overlap between the correct similarity region and the region estimated under the model.

The performance of L_1 distance as a proxy for either of the two similarities is quite poor, not surprisingly. In particular, the threshold on the distance necessary to achieve reasonable precision corresponds to an ROC point with a very low recall. It seems obvious that no subset of the features inherently limited to axis-parallel stumps will do much better in this case.

In hindsight (given what we know about the target similarities in each case), the best solution is of course to simply extract the parameter which directly affects the similarity. This would mean simply converting the Euclidean coordinated to polar coordinates and using the phase (modulo π) and magnitude as an embedding of the data for, respectively, $\mathcal{S}_A$ and $\mathcal{S}_N$. Of course, normally we do not have such knowledge
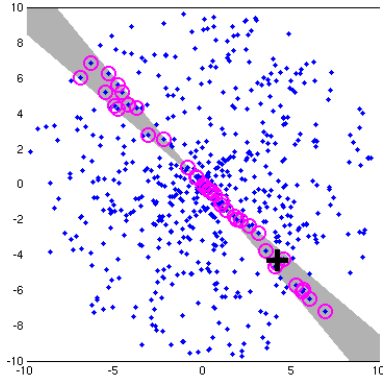
(a) Angle similarity $\mathcal{S}_A$ (b) Norm similarity $\mathcal{S}_N$
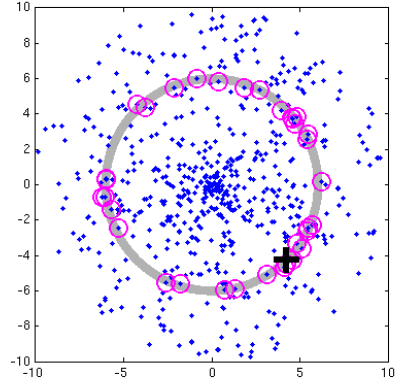
Figure 3-5: Toy 2D data set, with examples of angle similarity and norm similarity. $\mathcal{S}_A(\mathbf{x}, \mathbf{y}) \sim |\text{atan}(\mathbf{x}) - \text{atan}(\mathbf{y})|$, and $\mathcal{S}_N(\mathbf{x}, \mathbf{y}) \sim |\|\mathbf{x}\| - \|vy\||$. Circles: examples similar, under each of the two concepts, to the query shown by the cross. Shaded area: the similarity region (see text.)

of the functional form of the target $\mathcal{S}$, and so we must rely on a learning algorithm with a rather generic set of features that will allow us to reasonably approximate it.

3.4.1   Embedding with generic projections

We are now extending the family of the projection functions used to form the embedding. We will consider all generalized linear projections of the form

$$f(\mathbf{x}; \theta) \triangleq \sum_{j=1}^{D} \theta_j \phi_j(\mathbf{x}). \tag{3.18}$$

This still leaves the choice of ϕ unspecified. In this thesis, we will limit our attention to polynomial projections, in which

$$\phi_j(\mathbf{x}) = \mathbf{x}_{(d_j^1)} \cdots \mathbf{x}_{(d_j^{o_j})}, \tag{3.19}$$

that is, each term ϕ_j in (3.18) is a product of o_j components of $\mathbf{x}$ (not necessarily distinct). In our experiments, we have used projection with o_j bounded either by 1 (linear projections) or 2 (quadratic projections).

This is a fairly broad family (that of course includes the axis-parallel projections used so far), and the framework developed in this section does not necessarily assume any further constraints. The specific choice of the projections is a matter of design, and should probably be guided by two considerations. One is domain knowledge– for instance, in our toy example it is pretty clear that quadratic projections should

66

be appropriate for the task. The second consideration is computational resources: since learning with such projections involves optimization, increasing the number of parameters will increase the time required to learn an embedding.

3.4.2 The weak learner of projections

Until now the weak learner in boosting was essentially ranking all the features based on the current weights on the examples. Transition to an infinite set of projections requires a weak learner capable of searching the space of features in order to optimize the objective function in a current iteration of boosting. Below we define a differentiable objective function aimed at maximizing r_m, and describe a gradient ascent procedure for that function.

In order to have a differentiable objective, we need a differentiable expression for the classifier. Therefore, we replace the "hard" step functions in (3.2) with a differentiable approximation via the logistic function:

$$\tilde{h}(\mathbf{x}; f, T) \triangleq \frac{1}{1 + \exp\left(\gamma(f(\mathbf{x}) - T)\right)}. \tag{3.20}$$

This introduces the parameter γ, the value of which can affect the behavior of the learning algorithm.[13] We suggest the following heuristic to set a reasonable γ:

$$\gamma = \frac{log((1 - .999)/.999)}{\min\left(|\min_i\{f(\mathbf{x}_i) - T\}|, |\max_i\{f(\mathbf{x}_i) - T\}|\right)},$$

which means that the lowest value of $\tilde{h}$ on the available data is at most 0.001, and the highest value is at least 0.999.[14]

We also change the definition of the classifier associated with $\tilde{h}$ from (3.3) to

$$\tilde{c}(\mathbf{x}, \mathbf{y}) \triangleq 4\left(h(\mathbf{x}) - 1/2\right)\left(h(\mathbf{y}) - 1/2\right). \tag{3.21}$$

Note that the response of so defined $\tilde{c}$ is a continuous variable in the range $[-1, 1]$, that can be thought of as a confidence rated prediction: if both $f(\mathbf{x})$ and $f(\mathbf{y})$ are far from the threshold on different sides, then $\tilde{c}(\mathbf{x}, \mathbf{y})$ will be close to $+1$, and if they are very close to the threshold the response will be close to zero.

To calculate the gradient, we need to compute the partial derivatives of the objective function with respect to the projection parameters $\theta_1, \ldots, \theta_D, T$. Below we do that for two cases: the fully supervised case and the semi-supervised one.

---

[13]In principle the same role of determining the shape of $\tilde{h}$ can be played by the parameters θ_j, however we found that using γ, in particular for data with vastly different ranges for different dimensions, improves both the numerical stability and the speed of convergence of the learning.

[14]In principle the objective may be explicitly optimized with respect to the value of γ as well, however we have not pursued that direction.

Fully supervised case

To simplify notation, let us denote the parameter with respect to which we differentiate by θ. Recall that when total N of positive and negative pairs are available, labeled by l_i, the objective function is given by

$$\tilde{r}_m \triangleq \sum_{i=1}^{N} W_m(i) l_i \tilde{c}(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}). \tag{3.22}$$

The partial derivative of (3.22) is

$$\frac{\partial}{\partial \theta} \tilde{r}_m = \sum_{i=1}^{N} W_m(i) l_i \frac{\partial}{\partial \theta} \tilde{c}(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}). \tag{3.23}$$

Now, from definition of $\tilde{c}$

$$\frac{\partial}{\partial \theta} \tilde{c}(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}) = 4 \left[\frac{\partial}{\partial \theta} \tilde{h}(\mathbf{x}_i^{(1)}) \left(h(\mathbf{x}_i^{(2)}) - \frac{1}{2} \right) + \frac{\partial}{\partial \theta} \tilde{h}(\mathbf{x}_i^{(2)}) \left(h(\mathbf{x}_i^{(1)}) - \frac{1}{2} \right). \right] \tag{3.24}$$

Next, we can take the derivative of the soft threshold $\tilde{h}$. Denoting $f_T(\mathbf{x}) \equiv f(\mathbf{x}) - T$ for simplicity, we get

$$\frac{\partial}{\partial \theta} \tilde{h}(\mathbf{x}) = \frac{\gamma \exp(-\gamma f_T(\mathbf{x}))}{(1 + \exp(-\gamma f_T(\mathbf{x})))^2} \frac{\partial}{\partial \theta} f_T(\mathbf{x}). \tag{3.25}$$

Finally, we can take the derivative of the projection. For the coefficients θ_q, $q = 1, \ldots, D$ this will yield

$$\frac{\partial}{\partial \theta_q} f_T(\mathbf{x}) = \phi_D(\mathbf{x}), \tag{3.26}$$

and the derivative with respect to the threshold is simply -1. Plugging the equations (3.24)-(3.26) back into (3.23) produces the partial derivative of $\tilde{r}_m$ w.r.t. the projection parameter θ, and allows us to perform gradient ascent using standard numerical methods.[15]

Semi-supervised case

The main difference of the semi-supervised case from the supervised one is that we need to take the derivative of the second part of (3.16) containing the expected responses of $\tilde{c}$. Unfortunately, we can no longer use P_j to estimate that expectation since any point on the line $f(\mathbf{x})$ will produce a different response of $\tilde{c}$ when paired with $f(\mathbf{x}_i)$. Thus, we resort to explicitly estimating the expectation, which is given

---

[15]One can also calculate the Hessian to allow for a more efficient search with Newton-Raphson method, but we have not pursued that.

Data set	L_1	SSC	BoostPro
MPG	2.7368 ± 0.4429	2.2376 ± 0.3900	1.9286 ± 0.1941
CPU	4.1969 ± 0.2189	2.1503 ± 0.1500	2.0890 ± 0.1198
Housing	3.4641 ± 0.2568	2.4748 ± 0.5166	2.4985 ± 0.5272
Abalone	1.4582 ± 0.0557	1.4700 ± 0.0606	1.4994 ± 0.0496
Census	24705.0481 ± 988.2865	22480.2135 ± 1588.8343	18379.6952 ± 540.5984

Table 3.3: Test accuracy of constant robust locally-weighted regression. Shown are the mean values $\pm$ std. deviation of mean absolute error (MAE) for 10-fold cross-validation.)

Data set	L_1	SSC	BoostPro
Letter	0.0449 ± 0.0050	0.0426 ± 0.0065	0.0501 ± 0.0061
Isolet	$0.1265 \pm$	0.1713 ± 0.0215	0.0993 ± 0.0237

Table 3.4: Test accuracy of K-NN classification with SSC vs L_1 similarity (mean $\pm$ std. deviation for 10-fold cross-validation.)

by the integral

$$E_{\mathbf{y}}\left[\tilde{c}(\mathbf{x}, \mathbf{y})\right] = \int_{-\infty}^{\infty} \tilde{h}(\mathbf{x} - 1/2)\tilde{h}(\mathbf{y} - 1/2)p(\mathbf{y})d\mathbf{y}. \tag{3.27}$$

We estimate this integral by taking the sum over the available examples. Thus, the expression for $\tilde{r}_m$ becomes

$$\tilde{r}_m = \sum_{i=1}^{N_p} W_m(i)\tilde{c}(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}) - \sum_{j=1}^{N} S_j \frac{4}{N-1} \sum_{b \neq j} \left(h(\mathbf{x}_j) - \frac{1}{2}\right)\left(h(\mathbf{x}_b) - \frac{1}{2}\right). \tag{3.28}$$

Taking the derivative of (3.28) involves assembling N_p terms given in (3.24) (for the positive pairs) and $N(N-1)$ terms for the unlabeled examples. If computation time is of concern and the quadratic dependence on N is infeasible, the latter term may be further approximated by sampling a constant number of $\mathbf{x}_b$'s at, say, fixed percentiles of the distribution of $f(\mathbf{x})$.

3.4.3 Results

Synthetic 2D data

For each of the two similarity tasks introduced in the beginning of Section 3.4, the algorithm constructed an embedding with $M = 200$ dimensions based on $N_p = 1000$ positive examples (and no negative examples), using projections quadratic in x_1 and x_2. Figure 3-13 shows examples of the learned weak classifiers. The plotted regions correspond to h; the value of the classifiers c for any two examples is obtained by

Data Set	L_1	SSC	BoostPro
MPG	13.9436 ± 5.1276	10.0813 ± 3.8950	7.4905 ± 2.5907
CPU	37.9810 ± 5.2729	18.2912 ± 4.2757	9.0846 ± 0.9953
Housing	26.5211 ± 6.8080	14.3476 ± 9.1516	13.8436 ± 8.4188
Abalone	4.7816 ± 0.5180	4.8519 ± 0.4712	4.7602 ± 0.4384
Census	$2.493 \times 10^9 \pm 3.3 \times 10^8$	$2.237 \times 10^9 \pm 3.2 \times 10^8$	$1.566 \times 10^9 \pm 2.4 \times 10^8$

Table 3.5: Test accuracy of constant robust locally-weighted regression on regression benchmark data from UCI/Delve. Shown are the mean ± std. deviation of mean squared error (MSE) over 10-fold cross validation. Results for SVM are from [83]; see text for discussion.

Data set	Error	Method	Source
MPG	7.11	SVM	[83]
CPU	28.14	Regression Trees	[113]
Housing	9.6	SVM	[83]
Abalone	4.31	Neural Network	[83]
Census	1.5×10^9	Regression Trees	[113]
Letter	0.0195	ECOC with AdaBoost	[28]
Isolet	0.0372	SVM	[73]

Table 3.6: The best of the available results of other methods published for a similar experimental setup. The error shown is MSE for the regression sets and mean classification error for the classification sets.

placing them on the figure and comparing the colors at their location. Thus, the pairs of red crosses would be classified as dissimilar (by the weak classifier alone!) while the pairs of circles would be classified as similar. The typical shape of h (origin-centered disks for norm, and "bow-tie" shapes for angle) effectively corresponds to a quantization of the underlying polar coordinate used to define similarity, although the values of those coordinates were withheld during learning. Figure 3-14 shows retrieval results; the lighter regions in the data space correspond to a L_1-ball of radius $R = 20$ in H around the query (shown by cross). The ROC curves for the similarity retrieval/classification are shown in Figure 3-15. We also evaluated the DistBoost algorithm from [60] on these two problems. Note that the comparison is somewhat "unfair" since DistBoost assumes that the similarity corresponds to equivalence classes on $\mathcal{X}$. Nevertheless, DistBoost performed reasonably well, in particular for low values of recall. Overall, on these synthetic data our embedding approach is clearly superior to both DistBoost and the L_1 distance, which performs only slightly better than chance (as expected).

Real data sets

Tables 3.3, 3.5 and 3.4 summarize the results of an experimental comparison of BoostPro with other similarity models as a tool in example-based regression and

(a) Mean absolute error

(b) ROC curve
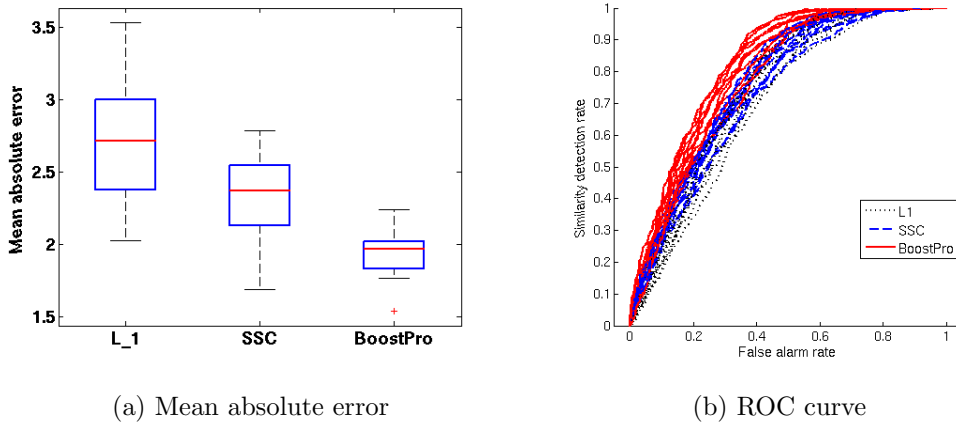
Figure 3-6: Results on Auto-MPG data set. Left: box plot of test mean absolute error of example-based regression using, from left to right, L_1 distance in $\mathcal{X}$, SSC embedding and BoostPro embedding. The plots show distribution of results in ten-fold cross validation. Right: test ROC curves for the ten folds of the cross-validation. Black (dotted): L_1 in $\mathcal{X}$; Blue (dashed): SSC; red (solid): BoostPro.

(a) Mean absolute error

(b) ROC curve
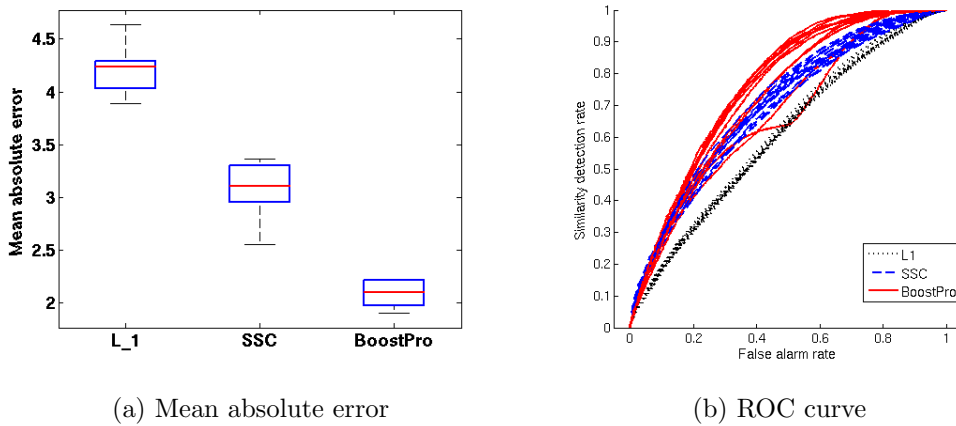
Figure 3-7: Results on Machine CPU. See legend for Figure 3-6.

classification on the seven real data sets. In all data sets the projections used by BoostPro were linear projections with two terms, in other words, each dimension of the embedding is a thresholded linear combination of two coordinates of the input. The performance in terms of mean error is also summarized graphically in Figures 3-6-3-12; these figures show the distribution of mean errors as well as the ROC curves for the three similarity measures on seven data sets.

Selecting the terms in the projection in BoostPro requires some care. With two-dimensional projections it may be possible (if $dim(\mathcal{X})$ is low enough), in principle,

(a) Mean absolute error

(b) ROC curve
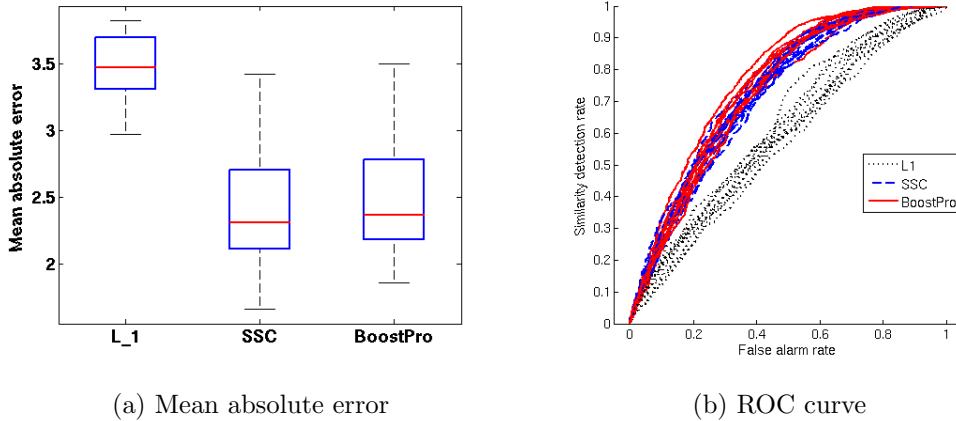
Figure 3-8: Results on Boston Housing. See legend for Figure 3-6.

(a) Mean absolute error

(b) ROC curve
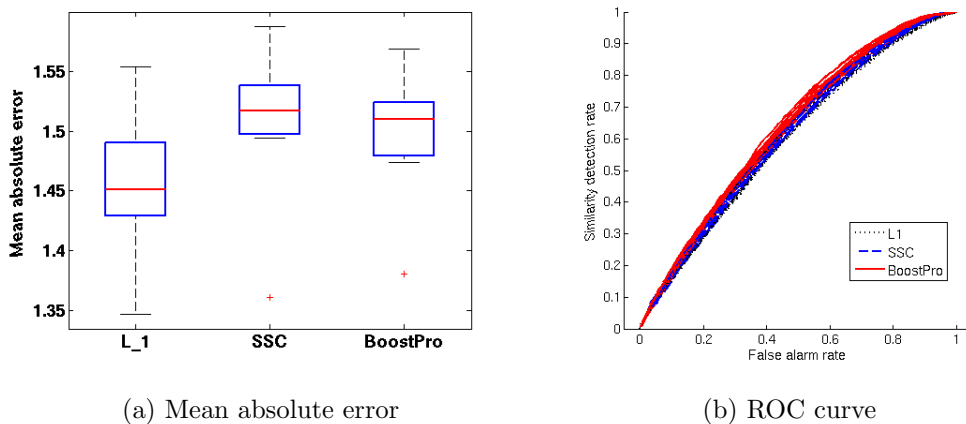
Figure 3-9: Results on Abalone. See legend for Figure 3-6.

to exhaustively consider all $\dim(\mathcal{X})(dim(\mathcal{X}) - 1)/2$ combinations, perform gradient descent on each and select the optimal one. However, it is extremely expensive (at every step of the gradient descent we need to compute the gradient, which requires a pass over all the training data.) In addition, while this may speed up the reduction in training error, there is no requirement to find *the best* weak classifier in a given iteration–just to find a weak classifier better than chance. Therefore, instead of such exhaustive search we consider with a fixed number (typically 100) randomly constructed term combinations, set the projection parameters θ to randomly selected numbers, find the local maximum of r_m by starting the gradient ascent at each of these projections, and select the one that attains the highest r_m. Note that this is an inherently parallelizable procedure, since the gradient ascent proceeds independently from every initialization point. We take advantage of this and use a parallelized

72

(a) Mean absolute error

(b) ROC curve
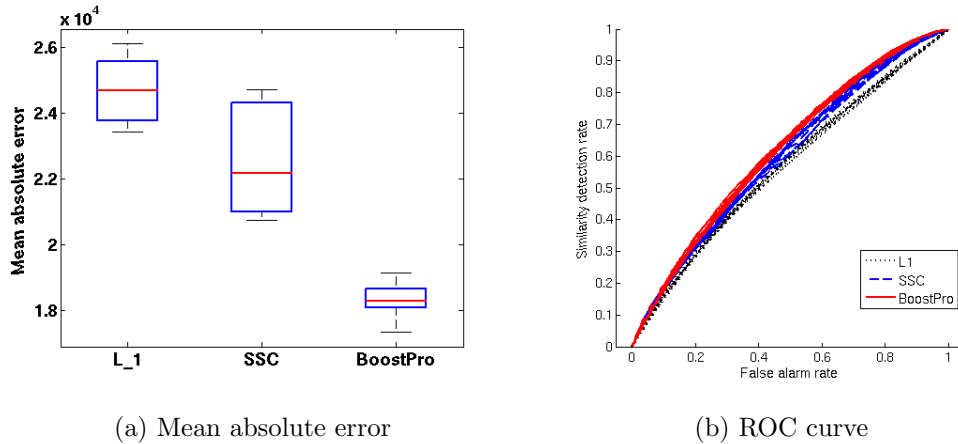
Figure 3-10: Results on US Census. See legend for Figure 3-6.

(a) Mean classification error

(b) ROC curve
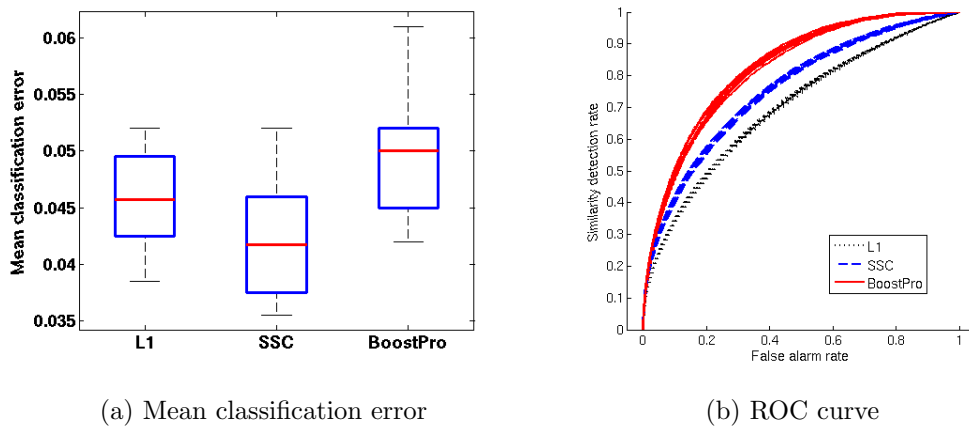
Figure 3-11: Results on Letter. Plots on the left show classification error distributions; otherwise, see legend for Figure 3-6.

implementation. However, we believe that the under-exploration of the space of projections is the main cause for the failure of BOOSTPRO to improve over the other similarity models.

Nevertheless, in most cases, BOOSTPRO outperforms other similarity models robustly, as measured by the means and standard deviations of mean errors in cross validation. The main conclusion from these experiments is that for a practitioner of example-based estimation methods, it is often beneficial to model the similarity rather than apply the default L_1-based neighbor search in $\mathcal{X}$. In some cases there is no improvement, however; we suspect that these are the cases in which the L_1 is an appropriate proxy for similarity. The following "hybrid" approach provides perhaps the safest means of optimizing the performance of a similarity model: using a held-out

(a) Mean classification error

(b) ROC curve
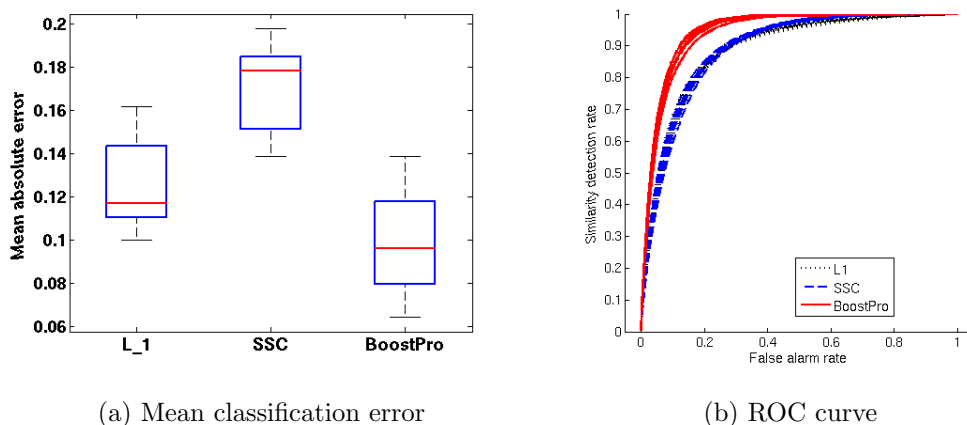
Figure 3-12: Results on Isolet. See legend for Figure 3-11.
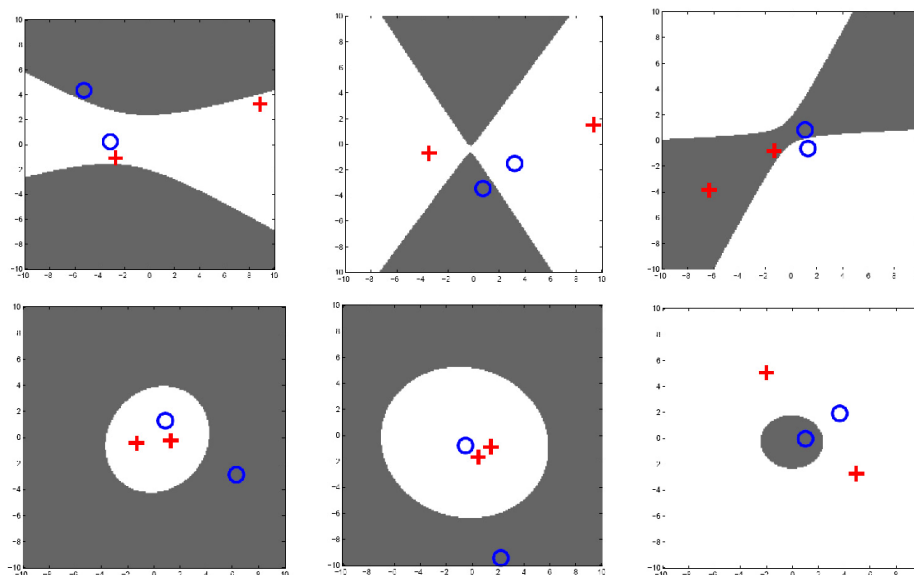
Figure 3-13: Typical weak classifiers (thresholded projections) learned for the angle (top three) and norm (bottom three) similarity on the synthetic 2D data. The darker shade corresponds to the area where $h = +1$. Crosses and circles show pairs that would be classified by the projection as similar and dissimilar, respectively.

test set (or in a cross-validation setting) evaluate the estimation error using each of the three similarity models, and select the one with the best performance.

In order to place these results in the context of state-of-the-art results, we can also compare our results to the best results published in the machine learning literature for the data sets in question, as summarized in Table 3.6.[16] For each data set, we have

---

[16]Due to a large variety of techniques and experimental designs used in such evaluations, such comparisons should be considered carefully. We attempted to locate the most relevant results with
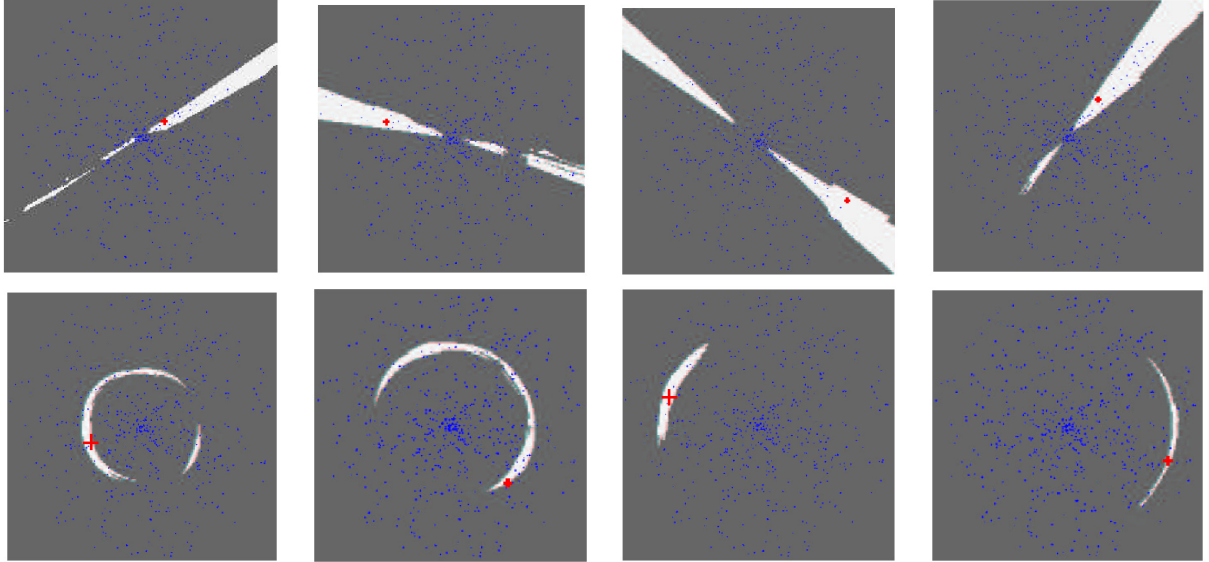
Figure 3-14: Synthetic data: example similarity regions. Light areas correspond to $\mathbf{x}$ such that $\|H(\mathbf{x}) - H(\mathbf{q})\|_H \leq R$, with the query $\mathbf{q}$ shown by the red cross and $R = 20$ set for 0.5 recall. The dots show the training data. Top: angle similarity, Bottom: norm similarity.
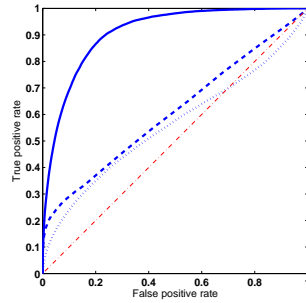
(a) Angle
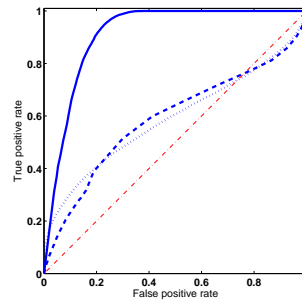
(b) Norm

Figure 3-15: ROC curves for the retrieval experiments with angle and norm similarities (see Figure 3-5. Diagonal: chance. Dotted: L_1. Dashed: DistBoost. Solid: the embedding learned with semi-supervised BoostPro.

---

respect to the specific set of experiments reported here.
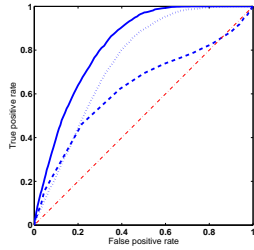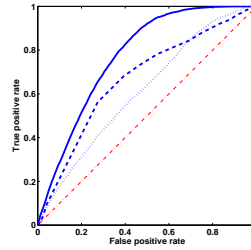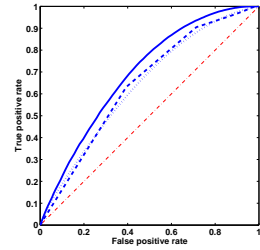
<center>(a) MPG   (b) Housing   (c) Abalone</center>

Figure 3-16: Results on three of the UCI data sets. Comparison of L_1, DistBoost and semi-supervised BoostPro. Diagonal: chance. Dotted: L_1. Dashed: DistBoost. Solid: the embedding learned with semi-supervised BoostPro.

Data set	MPG	CPU	Housing	Abalone	Census	Letter	Isolet
M	180 ± 20	115 ± 48	210 ± 28	43 ± 8	49 ± 10	133 ± 13	121 ± 52

Table 3.7: Lengths of embedding learned with BoostPro on UCI/Delve data; mean $\pm$ std. deviation in 10-fold cross validation.

cited the best result, along with the method with which it was achieved and the source. As can be seen, for some data sets (regression) the simple constant, i.e. zeroth-order, robust locally-weighted regression (introduced in Section 2.2) with a BoostPro-learned embedding performs on a par with the best published results, while for other data sets (primarily classification) its performance appears to be inferior.

Note that our embedding offers a critical advantage over SVM or similar classification machines, when the task of similarity retrieval is relevant. This advantage is in our ability to directly approach this task as a distance-based neighbor retrieval in the embedding space, and to use LSH for a sublinear time search.

The main consequence of this ability is the computational gain. When the similarity notion is inherently related to class labels, SVM could be in principle applied to classify the query example and then retrieve all database examples that have the same class label. Since SVM are known to often retain a significant proportion of the data as support vectors, and since the computational cost of applying an SVM is directly proportional to the number of support vectors, this often will be much more expensive than the fast search with LSH.

When the relevant similarity notion can not be linked to a classification problem, SVM or similar mechanisms are simply not directly applicable to the retrieval task. One possibility to overcome this is to train an SVM classifier of *similarity*, i.e. a classifier that operates on pairs of examples. That, however, would require to apply an SVM, which is often an expensive operation itself, for all pairs formed by connecting the query and each of the database examples. This is clearly prohibitively expensive even with medium-size databases, and completely infeasible for databases of the type we will discuss in the next chapters, with millions of examples. This is in stark contrast to the cost of retrieval with our method, that combines the learned representation in the embedding space with the fast search using LSH, making retrieval in near-real time easily implemented for these very large databases.

BoostPro also shows an improvement over SSCon most data sets, at the same time greatly reducing the embedding size; Table 3.7 shows the average values of M for BoostPro(these values are essentially determined by the stopping criteria of AdaBoost, that stops when it can not find, within a reasonable time, a weak classifier with non-zero r_m.) Compare these numbers to those in Table 3.2.

Semi-supervised scenario

Figure 3-16 shows a result of comparing the semi-supervised version of BoostPro to DistBoost (and L_1) on three of the UCI data sets: Abalone, Housing and Auto-MPG. The ROC curves shown are for a single partition of the data, using 40% for testing. On these three data sets, the advantage of our embedding method is still noticeable, although it is less pronounced, since both DistBoost and L_1 perform better than for synthetic data. In all the five data sets, the expected similarity rate ρ (e.g., the probability that random two examples are similar) is between 0.05 and 0.3. Nevertheless, the positive-only version of the algorithm based on the assumption that this rate is low, performs well.

We have also investigated the effect of the ground truth similarity rate on the
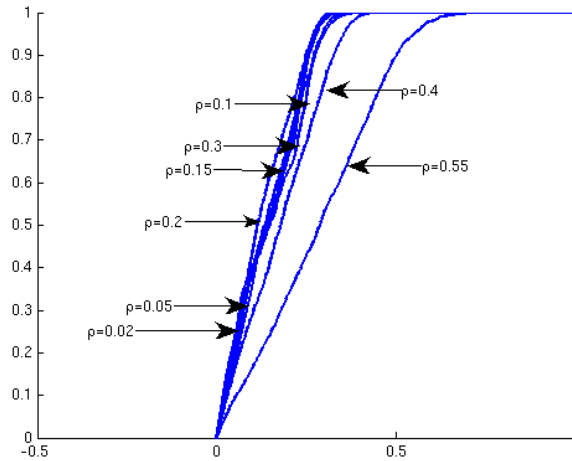
Figure 3-17: Effect of similarity rate on the performance of the semi-supervised BOOSTPRO, on synthetic norm similarity data. The ROC curves are shown for the retrieval task on 1000 test points, using 600 unlabeled points and 2,000 similar pairs, with M=100.

performance of the semi-supervised version of BOOSTPRO. Norm similarity in the 2D "toy" data set is determined by setting a threshold on the difference between Euclidean norms of the two points in question; varying this threshold corresponds to modifying the similarity rate (if the threshold is low, ρ is low). We have evaluated the retrieval performance of the algorithm for a range of values of ρ between 0.02 and 0.55. Figure 3-17 shows the ROC plots for eight values of ρ, obtained by applying the semi-supervised BOOSTPRO. The FP rate was estimated as per equation 3.7, that is, using the probability mass estimate for a threshold and the correction term for the known ρ. From the results it is apparent that the algorithm is very robust, in the sense that the semi-supervised version achieves identical (good) results for ρ up to 0.3; the curve for 0.4 is noticeably inferior, and for 0.55 the curve deteriorates much further. This is consistent with our observation that for values of ρ up to 0.3 in the UCI/Delve data sets, the performance of semi-supervised algorithm does not suffer from replacing actual negative examples with the expectations over all pairs, corrected for the known (or estimated) ρ.

3.5 Discussion

We have developed a family of algorithms for learning an embedding from the original input space $\mathcal{X}$ to an embedding space H. The objective of these algorithms is to to optimize the performance of L_1 distance in the embedding space as a proxy for the unknown similarity $\mathcal{S}$, which is conveyed by a set of examples of positive pairs (similar under $\mathcal{S}$) and, possibly, negative pairs, perhaps along with some unlabeled example in $\mathcal{X}$.

The following summarizes the main properties of each algorithm.

Similarity Sensitive Coding (SSC) The algorithm takes pairs labeled by similarity, and produces a binary embedding space H, typically of very high dimension. The embedding is learned by independently collecting thresholded projections of the data.

Boosted SSC This algorithm addresses the redundancy in SSCby collecting the embedding dimensions greedily, rather than independently. It also introduces weighting on the dimensions of H.

BoostPro This algorithm differs from the Boosted SSCin that the dimensions of the embedding are no longer limited to axis-parallel stumps. We have introduced a continuous approximation for the thresholded projection paradigm in which a gradient ascent optimization becomes possible.

Semi-supervised learning For each of these three algorithms we have presented a semi-supervised version which only requires pairs similar under $\mathcal{S}$, in addition to a set of unlabeled individual examples in $\mathcal{X}$.

As part of the discussion in this chapter we have applied some of the new algorithms to a number of real-world data sets from public data repositories, and observed very good performance, both in terms of the ROC curve of similarity detection and in terms of the prediction accuracy for regression and classification tasks. In the following chapters we will see how the proposed framework can be applied to challenging problems in machine vision.

Chapter 4

Articulated Pose Estimation

In this chapter we describe a new approach to estimation of articulated pose of humans from single monocular images. Our approach is example-based: it reduces the problem of recovering the pose to a database search under L_1 in the embedding space, which is carried out extremely fast using LSH. The embedding is constructed based on edge direction histograms, using the algorithms presented in Chapter 3. Underlying this construction is the definition of a similarity concept under which two images of people are similar if the underlying poses are, and learning an embedding that is sensitive to that similarity.

We start with describing the problem domain and presenting our approach to it in a nutshell in Section 4.1, and cover some related work in Section 4.2. Section 4.3 gives the details of the representation and the learning problems defined for the task. Experimental results in two estimation tasks are described in Sections 4.4 and 4.5. In Chapter 5 we discuss the integration of our approach to single-frame pose estimation into a tracking framework.

4.1 The problem domain

The articulated pose estimation problem is formulated as follows. We are given an image which contains a human body.[1] We also have an *articulation model*–a model of the body that describes the current 3D body configuration in terms of a set of *limbs* and rotational *joints* that connect them into a tree structure.

This model is illustrated in Figure 4-1. The image on the left is not a photograph of a real person but a synthetically generated image of a humanoid model obtained with a computer graphics program POSER [29]. This image corresponds to the articulated model in the left part of the figure. The model is shown by plotting 2D projections of 20 key joints (crosses) and the lines connecting them, that roughly correspond to limbs. This model may be described by 60 numbers, namely the (X, Y, Z) coordinates of the joints (an alternative form of describing the model would be in terms of articulated angles, which we will discuss later.) In fact, there are hundreds of parameters in

---

[1]The presented framework can be applied to any articulated body, but estimating pose of humans is by far the most important task of this sort.
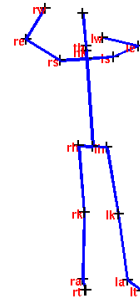
(a) Image of a body   (b) Corresponding articulated model

Figure 4-1: A (synthetic) image of a person and the corresponding articulated model. The goal of pose estimation is to derive the representation from the image on the left. Crosses show key joints, labeled with abbreviations. **l/r**: left/right, **t**: big toe, **a**: ankle, **k**: knee, **h**: hipbone, **s**: shoulder, **e**: elbow, **w**: wrist. Additional parts are the base of the neck **nk**, the base of the skull **th** and the top of the skull (not labeled).

addition to these 60 numbers that affect the resulting image: the articulated pose of additional body parts not accounted for by this coarse model, such as fingers; shape of the actual body parts (the model, so to speak, describes the "bones", but not the flesh); facial expression; hair style; clothing; illumination etc. Added to that could be the parameters that describe the scene, the objects in the background etc. The goal of a computer graphics program like POSER is to start with these parameters and produce a realistic image, that is, to go from the right half of Figure 4-1 to the left half. The goal of computer vision is the opposite. In the context of articulated pose estimation this goal is to start from the left half (the image), and recover the relevant parameters (the right half) of the representation that "generated" the image, while ignoring the *nuisance parameters*–all those additional aspects of the visual scene listed above. When the image is actually synthetically generated the success of this task is easy to measure, since we have access to the ground truth. For real images such evaluation is more difficult. When measurements of the underlying pose are available, for example obtained using a motion capture device at the same time as the images are taken, this may be done in a precise fashion.[2] In other cases this may be subjective, or it may depend on the success of a "downstream" application that relies on the estimated pose (we discuss some applications in the next section and in Chapter 5.)

---

[2]However, special caution is required to make sure the motion capture setup, e.g. special clothing or visible sensors, is not used by the estimation algorithm to "cheat".

4.2 Background on pose estimation

There exists a large body of literature on the estimation of the pose of articulated bodies. We only focus here on work most related to our approach. It should also be noted that much more attention has been given to the task of *articulated tracking* of humans: recovering the sequences of articulated poses from a video showing a moving person. This task is usually approached in a qualitatively different way from single-frame pose estimation. In particular, tracking algorithms (with almost no exceptions) rely on the assumption of manual initialization. While the tracking setup is in some ways more challenging than the single-frame one, it also allows access to provides valuable cues from motion that are not available in a static task. This may allow, in particular, to disambiguate certain situations which are very difficult or even impossible to disambiguate with a single frame. We will not discuss tracking here, but in Chapter 5 we will describe tracking algorithms that integrate our approach to single-frame pose estimation with a tracking setup, allowing us to relax or even abandon the initialization assumption.

Providing automatic initialization (and re-initialization throughout the sequence) for tracking is among the most important applications of single frame pose estimation. In fact, having a perfect pose estimator would eliminate the need for specialized tracking algorithms, since the accurate pose recovery would simply be done in every frame. Of course, this is not possible since single-frame estimation is ill-posed: in many "interesting" activities there is a great deal of occlusion of some body parts by others, there is often ambiguity related to symmetry, mirror reflections etc. Nevertheless the ability to recover pose from a single image is crucial for successful tracking. We discuss this in more detail in Chapter 5.

Much of the work has relied on deterministic methods guided on the known geometry of the articulated body. In [111] 3D pose is recovered from the 2D projections of a number of known feature points on an articulated body. Other efficient algorithms for matching articulated patterns are given in [45, 94, 88]. All of these approaches assume that detectors are available for specific feature locations, and that a global model of the articulation is available. Another family of approaches can somewhat relax these assumptions, at the cost of relying on the availability of multiple views [58].

Other techniques are based on statistical learning approaches. In [87] pose estimation is reduced to contour shape matching using *shape context* features. In [95], the mapping of a silhouette to 3D pose is learned using multi-view training data. These techniques were successful, but they were restricted to contour features and generally unable to use appearance within a silhouette. Some methods explicitly work with silhouettes only [40, 2] but those, due to a rather impoverished representation that greatly increases ambiguity, are usually restricted to a specific type of activity (walking is particularly popular.)

In [6] a hand image is matched to a large database of rendered forms, using a sophisticated similarity measure on image features. This work is most similar to ours and in part inspired our approach to pose estimation. However, the complexity of nearest neighbor search makes this approach difficult to apply to the very large numbers of examples needed for general articulated pose estimation with image-based

distance metrics.

Finally, we should emphasize that the task of pose estimation we are considering is decoupled from the tasks of *detection* and *localization*, i.e., determining whether an image contains a person and finding the specific portion of the image occupied by the person. There are a number of methods for carrying out those tasks, and we will assume that localization is solved by an external algorithm. Specific arrangements for obtaining this information in our experiments is described in Sections 4.4 and Section 4.5.

4.3 Example-based pose estimation

We approach pose estimation as a regression task, and develop an example-based approach to solving it. As described in Section 2.2.1, we can define a similarity concept $\mathcal{S}_p$ corresponding to pose similarity. We assume that we have access to a large and representative[3] database of images labeled with the corresponding poses. Then, the pose in a query image $\mathbf{x}_0$ can be estimated in by the following two steps:

- Find in the database some examples of poses similar to the unknown pose in $\mathbf{x}_0$.

- Using the retrieved examples, infer the pose in $\mathbf{x}_0$.

This fairly vague recipe is detailed in the sections below.

4.3.1 Pose-sensitive similarity

Suppose that a pose is represented by a parameter vector θ (we discuss some parameterizations below). Let $\mathbf{x}_1$ and $\mathbf{x}_2$ be two images depicting people whose articulated poses are, respectively, θ_1 and θ_2. Then, we define

$$\mathcal{S}_{p,R}(\mathbf{x}_1, \mathbf{x}_2) = +1 \;\Leftrightarrow\; \mathcal{D}_\theta(\theta_1, \theta_2) \leq R. \tag{4.1}$$

This is a generic similarity "template", and the precise definition depends on two parameters: the distance $\mathcal{D}_\theta$ used to compare poses, and the appropriate threshold R on that distance. The threshold could be set in two ways. The first is by finding R which meets some perceptual criteria: if $\mathcal{D}_\theta(\theta_1, \theta_2) \leq R$, then human observers will generally agree that the two poses "look similar", or are similar for the purpose of a particular application. Our approach to learning similarity from examples, developed in Chapter 3, is perfectly suited for such a definition since all it requires is a set of examples of similar pairs–which in this case may be supplied by human observers. A second method of setting R is by means of validation tuning with a specific estimation

---

[3]In the sense that for a random pose drawn from the distribution of poses, there is, with high probability, an example with a similar pose, under the relevant definition of similarity discussed in this section.

algorithm. That is, if the goal is to recover pose as precisely as possible,[4] and the estimation algorithm relies on similarity defined in (4.1), then we may look for R that minimizes the final error.

As for $\mathcal{D}_\theta$, there are two avenues for defining it, and the choice depends on the representation of the articulated model. A common representation, common in computer graphics and animation, is by *joint angles*[93]. Consider a directed graph representation of an articulated tree, where each node corresponds to a joint (we use the term joint loosely to refer to any rigid point in the model, so that, for instance, the top of the skull is also considered a "joint".) Edges leaving the node correspond to the limbs connected to that joint, and they connect it to the joints on the other side of the limb. Then the entire configuration of the model in 3D is given by a set of 3D rotation parameters in each joint plus the global position and orientation of the root, which is usually at the hip joint. This representation is convenient to describe articulation, and especially to parametrize articulated motion. Also, it describes the body articulation independently of the sizes of actual limbs. However it makes defining distances quite cumbersome. For instance, a 20 degree change in an angle may affect the global position of body parts very little if it is in a finger, or very much if it is in the hip.

For this representation, we use the *mean cosine deviation* distance $\mathcal{D}_{cos}$:

$$\mathcal{D}_{cos}(\theta_1, \theta_2) = \sum_{i=1}^{m} \left(1 - \cos(\theta_1^i - \theta_2^i)\right) \tag{4.2}$$

The second representation is in terms of 3D joint locations [57]. If there are L joints in the model, then the pose θ_i is fully described by $\theta_i = [\theta_i^1, \ldots, \theta_i^L]$, where the location of the j-th joint is given by $\theta^j = [\theta_{x,i}^j, \theta_{y,i}^j, \theta_{z,i}^j]^T \in \mathbb{R}^3$. This representation is somewhat redundant, since there are strong constraints on the relative locations of neighboring limbs, however it is very explicit and thus convenient for manipulating and comparing poses.

For this representation, we define the maximum deviation distance $\mathcal{D}_D$ by the maximum L_1 distance between any two corresponding joints in 3D:

$$\mathcal{D}_D(\theta_1, \theta_2) = \max_{1 \leq j \leq L} \sum_{d \in x,y,z} |\theta_{d,1}^j - \theta_{d,2}^j|. \tag{4.3}$$

In accordance with the approach we have outlined above, we will learn an embedding of the images space into a new space H, such that for two images $\mathbf{x}_1, \mathbf{x}_2$ and the corresponding poses θ_1, θ_2, $\|H(\mathbf{x}_1) - H(\mathbf{x}_2)\|$ is, with high probability, low if $\mathcal{D}_D(\theta_1, \theta_2) \leq R$.

---

[4]Note that this is rarely the real goal of an application; for instance, in an activity recognition scenario, or for understanding gestures, an error of a few degrees or a few centimeters relative to the "ground truth" is rarely a problem.
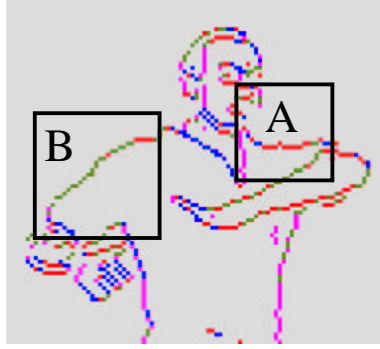
Figure 4-2: Illustration of the edge direction histogram (EDH) representation. Colors correspond to detected edge orientation red=0, green=$\pi/4$, purple=$\pi/2$ and blue=$3\pi/4$.

4.3.2   Image representation

Before we approach the learning task, we need to design the representation of the input space $\mathcal{X}$. The simplest decision would be to simply use the pixels of the image. However it is clearly not very helpful, due to a large effect of the nuisance parameters (color and illumination in particular) on the pixel intensities, and we would benefit from a representation that is more invariant to nuisance parameters while capturing information useful for inferring pose. In this chapter we will use the representation by multi-scale *edge direction histograms* (EDH) [68], often used in image analysis and retrieval, but until now it has not, to our knowledge, been used for pose analysis.

In order to compute EDH, we apply an edge detector of choice (we have used the Sobel detector [54]) to obtain an edge map, i.e. a binary image in which the value of a pixel is 1 if a detected edge passes through it. Next, each detected edge pixel is classified into one or more of four direction bins: $\pi/8, 3\pi/8, 5\pi/8, 7\pi/8$. This is done by applying a local gradient operator at each of the four orientations, and thresholding the response. Then, the histograms of direction bins are computed within sliding square windows of varying sizes (scales) placed at multiple locations in the image; the scales and the location grid are parameters to be set. This yields four integer values (the counts for the four direction bins) for each scale and location. The resulting multi-scale EDH is obtained by concatenating these values in a fixed order. Figure 4-2 illustrates the EDH representation; each of subwindows A and B contributes four numbers, calculated by counting edge pixels of four colors within the subwindow.[5]

Assuming, as we do, that the person localization task is solved for us and the image is centered on the bounding box of the body, a reasonable measure of similarity to apply to this representation is the L_1 distance, since a particular bin in the histogram corresponds to a roughly fixed location on the body. It is interesting to note the connection of this distance to the Hausdorff and Chamfer distances often used to

---

[5]Some pixels, in particular the ones at edge intersections, may have multiple colors, i.e. multiple orientations, assigned to them.

compare silhouettes or edge images [11]. A related distance is the Earth-Mover's distance[55].

Another interesting connection is to shape contexts [11], that have been used for pose estimation among other tasks [86, 87]

4.3.3 Obtaining labeled data

Our approach relies on the availability of a large database of images labeled with poses. Such a database may be constructed either by means of computer graphics package, such as POSER. or by recording data from human subjects. The synthetic generation is an appealing option since it is extremely cheap, can provide an arbitrarily large number of examples, and makes it easy to include as much variability in the data as desired (subject to model limitations of the software.) Importantly, it also provides accurate ground truth of the pose for every image. The resulting images can be quite realistic in terms of pose appearance (see Figures 4-3 and 4-6 for some examples).

Alternatively, such a database could also be created by recording images of real people in a variety of poses, along with the poses themselves measured by one of the available methods for that (usually based on instrumenting the actor with some sort of sensors.) However, this may be extremely expensive, labor-intensive and time-consuming. This may be possible for a constrained set of poses, for instance associated with a particular task or activity. If the goal is to have a very large database highly representative of the general pose space, this approach is probably infeasible, and even more so if we also want to include a significant variation in nuisance parameters in the data. One potential advantage of such a database, of course, is that the real training images may, in some sense, look more "like" the real test images the system would encounter. However in our opinion the state-of-the-art in computer graphics, as exemplified by POSER, removes this concern since the synthetic images are close in quality to the real ones, at least for the single-frame pose estimation purposes.[6] A more important advantage of a human-based database is in the realistic nature of the poses it contains, both in terms of the distribution and in terms of attainable configurations.

Fortunately, there is a way to have the best of both worlds. A set of poses can be recorded with a motion capture setup, and then used to create a large set of synthetic images by changing the viewpoint, slightly perturbing the poses, and randomly assigning the nuisance parameters. This is the approach taken to obtain the training data used in experiments described in Section 4.5 and in Chapter 5.

4.4 Estimating upper body pose

The experiments described in this section[7] deal with estimating only a partial pose, namely that of the upper body. The joints model specifies the location of shoulders, elbows and wrists. It is assumed that the person in the image is visible from about

---

[6]This may not yet be the case for synthetic rendering of motion!

[7]This section is based on the work published in [105]

Figure 4-3: Example training images for upper body pose estimation

the knee level up and is standing in an upright posture. The orientation (yaw) of the body is not constrained, and may vary between the two profile views, $\pm 90^o$.

4.4.1 Training data

The database of poses contains 500,000 images obtained by sampling uniformly at random the space of articulation angles, applying a feasibility correction algorithm of POSER (to prevent configurations which are either anatomically impossible or physically impossible, e.g. surface intersections), and rendering a 180×200 pixel image with randomly assigned nuisance parameters: illumination (obtained by modeling 4 random light sources), hair style, clothing, and hand configuration. As stated above, we assume that the body has been segmented from background, scaled, and centered in the image. Thus no background detail was generated, so the figures are on a uniform background. Figure 4-3 shows some examples.

4.4.2 The learning setup

The EDH representation was constructed with windows of sizes 8, 16 and 32, with each window sliding through locations spaced by half its size, yielding 11,728 histogram bins per image. With two bytes to represent each histogram bin, this requires above 11 Gigabytes to record the EDH for the full database.

Pose similarity was defined by setting a threshold of 0.5 on the $\mathcal{D}_{cos}$ between poses. This value was chosen by inspection, as it corresponded to a good cutoff between perceptually similar and dissimilar pairs of poses. Not surprisingly, similarity in this domain is a rare event; the similarity rate ρ defined in Section 3.2.4, measured on a million random pairs constructed over the training data, was only 0.0005.

Using the EDH representation as the input space $\mathcal{X}$, we constructed a training set for SSC: 100,000 positive examples and 1,000,000 negative examples. The larger number of the negative examples was motivated by the unbalanced nature of the

Model	$k = 7$	$k = 12$	$k = 50$
k-NN	0.882 (0.39)	0.844 (0.36)	0.814 (0.31)
Linear	0.957 (0.47)	0.968 (0.49)	1.284 (0.69)
const LWR	0.882 (0.39)	0.843 (0.36)	0.810 (0.31)
linear LWR	0.885 (0.40)	0.843 (0.36)	0.808 (0.31)
robust const LWR	0.930 (0.49)	0.825 (0.41)	0.755 (0.32)
robust linear LWR	1.029 (0.56)	0.883 (0.46)	0.738 (0.33)

Table 4.1: Mean estimation error for 1000 synthetic test images, in terms of $\mathcal{D}_{cos}$. Standard deviation shown in parentheses. Not shown are the baseline error of 1-NN, 1.614 (0.88), and of the exact 1-NN based on L_1 in $\mathcal{X}$, 1.659. LWR stands for locally-weighted regression, see Section 2.2.

problem, discussed in Chapter 3.

We evaluated a number of TP-FP gap values on a small validation set, and set the lower bound on the gap g to 0.25. With that gap bound, SSC selected 213 dimensions. Thus, the size of the database could be reduced, with the most economical data storage, from 11 Gigabytes to less than 14 Megabytes (recall that the dimensions produced by SSC are bit valued.) This data structure was then indexed by LSH, with l=80 tables and $k = 19$ bits per hash key. Note that the application of algorithm 3 (p. 2.4.2) is particularly simple on the bit-valued embedding H since each dimension only has one possible threshold. Thus the application of SSC with subsequent indexing by LSH may be seen as simply learning of an appropriate family of LSH functions.

We also tested the semi-supervised version of SSC described in Chapter 3. As expected for the low similarity rate in this case, the results were very similar to the results with the fully supervised version: we obtained 221 dimensions, with 97% overlap with the dimensions learned with the supervised algorithm. Thus we get essentially identical results with more than 10 times reduction in learning time (since the semi-supervised algorithm uses only 1/11 of the training examples used in the fully-supervised one.)

4.4.3 Results

To quantitatively evaluate the algorithm's performance, we tested it on 1000 synthetic images, generated from the same model, so that the ground truth is available. Table 4.1 summarized the results with different methods of fitting a local model; 'linear' refers to a non-weighted linear model fit to the neighborhood. The average size of the candidate set C found by LSH (i.e. the union of the buckets in the hash tables) was 5300 examples, about 1% of the data. We found that in almost all cases, the true nearest neighbors under $\mathcal{D}_H$ were among the candidates, which means that we do not pay significant cost for the speedup obtained with LSH.

The locally-weighted regression (LWR) [7] model was tested with zeroth-order, or constant, model (i.e., weighted average of the neighbors) and first-order, or linear,

Figure 4-4: Examples of upper body pose estimation (Section 4.4). Top row: input images. Middle row: top matches with LSH on the SSC embedding. Bottom row: robust constant LWR estimate based on 12 NN. Note that the images in the bottom row are not in the training database - these are rendered only to illustrate the pose estimate obtained by LWR.

model (i.e., weighted linear fit.) The robust LWR [22] re-weighted the neighbors in 5 iterations. The purpose of robust LWR, as explained in Section 2.2, is to reduce the influence of the outliers (examples with high residual under the current model fit) by iteratively decreasing their weights.

The results confirm some intuitive expectations. As the number of approximate neighbors used to construct the local model increases, the non-weighted model suffers from outliers, while the LWR model improves; the gain is especially high for the robust LWR. Since higher-order models require more examples for a good fit, the order-1 LWR only becomes better for large neighborhood sizes. Overall, these results show consistent advantage to LWR. Note that the robust linear LWR with 50 NN is on average more than twice better than the baseline 1-NN estimator.

We also tested the algorithm on 800 images of a real person; images were processed by a simple segmentation and alignment program, using a statistical color model of the static background and thresholding by intensity change. Figure 4-4 shows a few examples of pose estimation on real images. Note that the results in the bottom row are not images from the database, but a visualization of the pose estimated with robust linear LWR on 12-NN found by LSH; we used a Gaussian kernel with the bandwidth set to the d_X distance to the 12-th neighbor. In some cases (e.g. leftmost column in Figure 4-5), there is a dramatic improvement versus the estimate based on the single NN. The number of candidates examined by LSH was significantly lower than for the synthetic images - about 2000, or less than .5% of the database. This is expected since the real images differ from the synthetic ones in many subtle ways.

Figure 4-5: More examples, including typical "errors"; see legend of Figure 4-4. Note the gross error in the leftmost column, corrected by LWR. Examples in the right two columns are among the ones with most severe error in the test set.

It takes an unoptimized Matlab program less than 2 seconds to produce the pose estimate. This is a dramatic improvement over searching the entire database for the exact NN under L_1 in the embedding space, which takes more than 5 minutes per query, and in most cases produces the same top matches as the LSH. Note that exact search under L_1 distance in $\mathcal{X}$ (EDH) would take a number of days, in particular due to the enormous size of the database mentioned above.

Lacking ground truth for these images, we relied on visual inspection of the pose for evaluation. For about 2/3 of the examples the pose estimate was judged accurate; Figures 4-4 and 4-5 show a number of examples of typical estimates. On the remaining examples it was deemed inaccurate, on some examples the error was quite significant. Figures 4-4 and 4-5 show a number of examples, including two definite failures. Note that in some cases the approximate nearest neighbor is a poor pose estimate, while robust LWR yields a much better fit.

Nevertheless this system clearly can be improved. We can identify three sources of failure. One, not directly related to the learning and estimation procedures, is imperfect segmentation and alignment. The other potential reason is the suboptimal set of dimensions found by SSC (perhaps due to a poor choice of the gap bound); we suspect that 213 dimensions in the embedding is not rich enough a representation. The third problem is related to the limitations of the synthetic training set, in terms of coverage and representativeness of the problem domain. The experiment reported in the next section addressed some of these issues.

Figure 4-6: Examples of images in the motion capture-based repository of full body pose used in the experiments in Section 4.5.

4.5 Estimating full body pose

In this experiment we estimate full body pose, with the articulated model containing 60 parameters (this is the model illustrated in Figure 4-1(b).)

4.5.1 Training data

To improve the quality of the database we used the motion capture sequence freely available from [41]. The database contains over 600 sequences recorded from a variety of activities from everyday life (walking, greeting, brushing teeth), athletics (soccer, martial arts), etc. We collected 550,000 unique poses (with $\mathcal{D}_D$ between any two poses, as defined in (4.3), at least 1cm) and rendered a 240×320 pixel image from each pose at three random yaws, yielding a repository of 1,650,000 images labeled with the ground truth pose. The figure in each image is rendered at a random 2D location within the virtual scene, with up to 1m translation, in order to represent variability and with the intent to make the resulting estimator invariant to moderate translations (the 2D location is considered a nuisance parameter.) Figure 4-6 shows some examples of the images in this repository.

From each image we extracted the bounding box of the silhouette (using the fact that these synthetically generated images have known segmentation and thus the silhouette mask is available), and computed the EDH representation as described above, yielding 13,076 bins in a histogram.

4.5.2 Learning setup and results

We selected 60,000 images from the repository, constrained to upright postures. From these, we formed 20,000 positive pairs, subject to the similarity defined as in (4.1) with $\mathcal{D}_D$ as the pose distance and $r = 3cm$.

We then applied a semi-supervised version of BOOSTPRO, using linear projections over two dimensions. That is, each dimension of the embedding is obtained by taking

Figure 4-7: Testing on synthetic input. Column 1: test images. Columns 2-4: top 3 matches in H.

two random dimensions of the EDH, and optimized as described in Section 3.4.2, and the projections are combined by the semi-supervised boosting algorithm introduced in Section 3.4.1. In this way we constructed a 1,000-dimensional embedding H.

To get a better understanding of the relationship between independently selecting the dimensions of H with SSC and applying a greedy ensemble learning algorithm in BOOSTPRO, we also measured the TP-FP gap of the selected dimensions. As may be expected, some of the selected features, when considered alone, have very low gap values (as low as .02), nevertheless, they are selected by the boosting since their weighted gap, or equivalently the value of the objective r_m is high.

Figures 4-7 and 4-8 show examples of retrieval by exact NN search in the embedding space H. A more thorough evaluation of the error is reported in the next chapter, where we discuss integration of our pose estimation approach into a tracking framework.

4.6   Discussion

We have presented an example-based approach to articulated pose estimation from a single image. Its main difference from the previously proposed methods is that it does not attempt to build a global model of pose-image relationship, which is notoriously difficult. Instead, we use a large synthetic database to directly learn to detect when the poses underlying two images are similar, and, at the same time, construct an embedding into a space where that similarity is modeled by low L_1 distance between embedded images. The embedding framework and the resulting ability to retrieve similar poses by a simple L_1 search combined with the power of LSH give this approach a critical advantage: the solution to the complex problem of pose estimation becomes very simple and very fast. To our knowledge, no other single-frame pose estimation method that achieves similarly accurate estimates has a comparable speed. These properties make this pose estimation approach well suited as a component in articulated tracking algorithms. In the next chapter we describe two systems in which this is taken advantage of.

Figure 4-8: Results on real input. Column 1: test images. Columns 2-4: top 3 matches in H

Chapter 5

Articulated Tracking

This chapter describes two state-of-the-art probabilistic articulated tracking systems that rely on the pose estimation framework presented in Chapter 4. What distinguishes these systems from most other approaches to tracking is the use they make of task-specific similarity models, and specifically of the embeddings learned to facilitate detection of and search for similarity under these models. The similarity embeddings are an essential component of the systems from which they derive the ability to establish and evaluate hypotheses more efficiently.

In Section 5.1 we give a brief introduction to articulated tracking and discuss the role played by single-frame pose estimation. The first system, described in Section 5.2, is aimed at motion-driven swing dancing animation. That is a joint work with L. Ren, J. Hodgins, P. Viola and H. Pfister, published in [93]. Our contribution in that work has focused on the design of an example-based approach, based on similarity learning, to evaluating the hypotheses arising in the context of tracking constrained motion with a discrete dynamics model. The second system, that offers a new approach to the task of general articulated tracking, is a joint work with D. Demirdjian, L. Taycher, K. Grauman and T. Darrell. It was published in [35], and is described here in Section 5.3. In contrast to the first system, here our similarity learning framework is responsible for generating hypotheses based on estimated similarity between stored examples and the current observation.

5.1 Articulated tracking

The task of articulated tracking is to recover a *sequence* of articulated poses from a video (sequence of images) showing a moving articulated object, in most cases a human. The definition of pose, and the desired form of the output for each frame remain the same as in the static pose estimation task introduced in the previous chapter. What makes the tracking task qualitatively different is the presumed dependencies between consecutive poses. Tracking algorithms attempt to take advantage of such dependencies by framing the problem as that of probabilistic, usually Bayesian, inference. Consequently, the tracking task is often reformulated as estimating, at every frame, the *posterior distribution* of the pose parameters given the observations

so far.

5.1.1 Probabilistic tracking framework

Let $\theta^{(t)}$ be the articulated pose at time t, and $\mathbf{x}^{(t)}$ the observation recorded at that time–that is, an image, or a set of images received by a multi-camera system. We will denote $\mathbf{x}^{(1,\ldots,t)}$ the entire sequence of observations recorded up to, and including, time t. A standard step in probabilistic training is to model the *dynamics* of the motion by a distribution $p(\theta^{(t+1)}|\theta^{(1)},\ldots,\theta^{(t)})$. Usually folded into this model is the assumption that the current pose is independent of the pose in most of the previous frames, given the last few ones, so that the *prior* distribution of the pose given the "history" is

$$p(\theta^{(t+1)}|\theta^{(1)},\ldots,\theta^{(t)}) \;=\; p(\theta^{(t+1)}|\theta^{(t-\tau)},\ldots,\theta^{(t)}). \tag{5.1}$$

The relationship between the observation and the image is modeled by the *likelihood* function $p(\mathbf{x}_t|\theta^{(t)})$. The posterior distribution of the pose in the current frame, given all the observations so far, can be written as

$$\begin{aligned} p(\theta^{(t+1)}|\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(t+1)}) \;&\propto\; p(\mathbf{x}^{(t+1)}|\theta^{(t+1)}) \\ &\times \int_{\theta^{(t-\tau)},\ldots,\theta^{(t)}} p(\theta^{(t+1)}|\theta^{(t-\tau)},\ldots,\theta^{(t)}) p(\theta^{(t-\tau)},\ldots,\theta^{(t)}|\mathbf{x}^{(1)},\ldots,\mathbf{x}^{(t)}), \end{aligned} \tag{5.2}$$

using the Bayes rule to invert the likelihood and normalized to integrate to unity. The first factor in the integrand in (5.2) is computed according to the dynamics model (5.1); the second factor is expanded, recursively, using the same equation. An important consequence of this approach is that it is necessary to have an estimate for some initial frames in the sequence (or a reliable narrow estimate of the posterior in those frames) known. Providing such initialization is one of the main roles played by single-frame pose estimation, which is not dependent on the past estimates and observations.

In practical applications of tracking it is usually necessary to commit to a specific point estimate, that is, to produce a set of deterministic values of the model parameters. The commonly used principal way to form such an estimate in a probabilistic framework is to compute the *maximum a-posteriori* (MAP) value of the pose, that is, one that maximizes (5.2).

5.1.2 Models of dynamics

The form of the prior depends on the assumptions about the dynamics in the system, and on the "depth of the horizon" τ.

Continuous models

One popular model is the Gaussian *diffusion model* under which $\tau = 1$, and the prior is $p(\theta^{(t+1)}|\theta^{(t)}) = \mathcal{N}(\theta^{(t)};\mathbf{0},\mathbf{\Sigma})$. This essentially means that the model constrains the magnitude of the motion. This is the model used in the system described in

Section 5.3. More complex models that use higher values of τ may be able to model higher order properties of the motion such as velocity or acceleration. Such models are either analytical, implementing linear filtering mechanism like the Kalman filter and its variants [110], or non-parametric and learned from the data [72]. In these models the information about the past is entirely contained in the prior distribution. The prior distribution may be represented in a parametric form, or in a semi-parametric form, that is, by a set of samples [66]. The latter approach leads to the *particle filtering* framework, among the most popular ones in articulated tracking.

Discrete models

In some applications, the motion is highly structured, in the sense that only a finite, and relatively small, number of transitions are considered possible from any given pose. Furthermore, the number of attainable qualitatively different poses is also finite.[1] This is usually the case with activities that follow certain well-defined rules, like many sports or dancing. Under these constraints, it is possible to write down the dynamics model by explicitly enumerating the possible poses as nodes of a graph, and possible pose transitions as directed edges in this graph, weighted by the probability of the transition. This leads to the *motion graph* [72], the model used in the system described in Section 5.2.

5.1.3 Likelihood and similarity

The form of the likelihood term in (5.2) is typically determined by a generative model of image given pose. In most algorithms, maximizing the likelihood is achieved by a gradient-based optimization, in which the model is iteratively used to predict an observation, and the mismatch between the prediction and the actually observed data is used to improve the model. Two major problems with this approach are its computational complexity (likelihood computations are often the computational bottleneck of a tracking algorithm) and the dependence on the starting location [108].

In the context of motion graph tracking, computing the likelihood is reduced to evaluating the match between the input frame and a finite set of hypotheses, namely all the poses θt_i in the notation of (5.3). In other words, this is an instance of the similarity detection problem formulated in Section 1.1, and we will approach it using the tools developed in Chapter 3.

The likelihood model in the systems described in this chapter is example-based: likelihood is evaluated implicitly, by comparing the input frame to stored examples for which the pose is known:

$$p(\theta_i|\mathbf{x}^{(t)}) \ \sim \ \mathcal{S}_{pose}(\mathbf{x}^{(t)}, \mathbf{x}_i).$$

More precisely, the likelihood is assumed to be high for poses whose associated observations are similar to the current observation. This is made possible by applying the similarity learning framework developed in Chapter 3. The assumption underlying

---

[1]That is, the number of poses qualitative different up to some tolerance.

this approach is that examples similar to the input under $\mathcal{S}_{pose}$ correspond, with high probability, to peaks of the likelihood.

5.2 Case I: Motion-driven animation

This section describes a system for motion-driven animation in the domain of swing dancing. The application here is to allow a user to perform a swing dance in front of a multi-camera system, parse the motion into an admissible sequence of swing steps, and render it on the screen along with a matching action by a "virtual partner". The details of the entire system are available in [93]; here we will be focusing on the motion parsing task, which is essentially an articulated tracking task constrained to a specific *dictionary* of poses defined by the rules of swing dance.[2]

5.2.1 The setup and training data

The visual input to the system consists of three silhouettes extracted[3] from synchronized, calibrated consumer-grade digital cameras, mounted so that the fields of view overlap over a working area of approximately 8'×24'. These are concatenated to form a single three-view observation $\mathbf{x}$. The 62 parameters of the articulated model θ can be divided into the pose parameters ξ, describing the configuration of the limbs in the articulated tree (as joint angles in person-centered coordinate system), and the parameters specifying the orientation and location of the entire articulate tree in the world coordinate system. The orientation β can be encoded by a single number, the *yaw* angle, as we can assume that a swing dancer's body is generally in an upright posture. Specifying location involves additional two degrees of freedom λ for the coordinates on the ground plane where the center of mass is projected. In the remainder of this section we will refer to ξ as pose, to make this distinction clear.

The dynamics of the swing dance are modeled by a motion graph, constructed from 5,120 frames of motion capture data recorded with a professional dancer. Transitions are modeled using the distance between poses and ignoring the global orientation and location. In addition to these transitions, a small number of transitions are added manually to ensure compliance of the graph with the choreographic rules of swing.

The graph constructed in this way encodes a rather rigid constraint on the movement speed; if the user moves significantly slower or faster than the dancer in the motion capture recording session, no transitions will match his or her motion. To alleviate that, the motion graph is *augmented* by adding a few nodes on each edge to allow for slower motion and extra edges (from each node to its grandchildren and grand-grandchildren in the graph) to allow for faster motion.

---

[2]The contribution to [93] by the thesis author was primarily in this component.

[3]The silhouette is extracted by applying a simple color-based foreground/background segmentation model, finding the bounding box of the foreground pixels and resizing it to 60×60 pixels.

Tracking with motion graph

The prior distribution (5.1) corresponding to the motion graph is a discrete probability mass function, assigning values according to the weights on the out-going edges from $\theta^{(t)}$ and putting zero weights on absent edges. Consequently, the posterior is also discrete and can be described by listing all the paths $\theta^{(1)} \ldots, \theta^{(t)}$ in the graph. For computational reasons, it is possible to maintain only paths of certain maximal *buffer length b* (to avoid combinatorial explosion of the number of paths.) Furthermore, depending on the application we may afford to maintain a small *look-ahead* buffer, that is, we can defer inferring the pose in frame t until we have seen the frame $t + a$.[4]

Suppose that at the time $t + a$ we maintain n_t paths

$$\{\theta_i^{(t-b)}, \ldots, \theta_i^{(t+a)}\}_{i=1}^{n_t}.$$

The expression for the posterior is

$$p(\theta^{(t)}|\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t+a)}) \ \propto \ p(\mathbf{x}^{(t)}|\theta^{(t)})$$
$$\times \sum_{i=1}^{n_a} p(\theta^{(t)}|\theta^{(t-b)}, \ldots, \theta^{(t-1)}, \theta^{(t+1)}, \ldots, \theta^{(t+a)})p(\theta^{(t-\tau)}, \ldots, \theta^{(t)}|\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(t)}),$$

$$(5.3)$$

5.2.2   Two-stage architecture

In principle, one could attempt to build a similarity classifier which would be invariant to the external parameters of the model (location and yaw). However, this makes for a very complex problem, since the appearance of the silhouettes depends greatly both on yaw and on the pose and, albeit to a much lesser extent, on the location. Instead, we will follow a divide-and-conquer approach which breaks the estimation into a two-stage process.

At the first stage, the yaw of an observation is estimated and quantized into one of fixed yaw "bins". At the second stage, the pose of the observation is estimated using a similarity model that "specializes" on a particular yaw bin.[5] Each of these stages deals with an input subspace in which severe variations of appearance are largely accounted for by the relevant parameters to be recovered. Below we describe the role played by similarity detectors in the design of both stages.

The training data

As mentioned in the previous chapter, an example-based approach requires a representative database of pose-labeled observations. The human data itself in this case is

---

[4]The resulting dynamic model could in principle be modeled as a hidden Markov model, however estimation in this framework is not practical due to the complexity of the state space together with the requirements from the application: it has to be real-time and, importantly, have no "jitter" in the resulting animation.

[5]This architecture resembles the mixture of experts architecture[67], with the yaw classifier acting as a gating function.

representative of the relevant body configurations, but is not inclusive of all possible angles and locations in the working space (the latter affects the input significantly due to perspective distortion in the cameras). Thus, a database in this case is built following the same paradigm as in Section 4.5: by using human motion capture data described above to generate a larger set of synthetic examples.

For each recorded frame of the human data, an artificial set of silhouettes is rendered, using a computer graphics package, for every combination of 36 yaw bins and five fixed locations. Each yaw bin c is associated with the angle $c\pi/36$, and covers all yaw values $c\pi/36 - \pi/72 \leq \beta \leq c\pi/36 + \pi/72$; the five locations are in the center and four corners of the work area. This procedure yields a database of 921,600 example observations (triple silhouettes).

These observations are represented in the space similar to the EDH (see Chapter 4) but more appropriate for binary silhouette data: the concatenated set of responses of *box filters*. These filters can be visualized, as in Figure 5-1, by boxes placed over an image region and divided into black and white segments. The value of a filter is computed by subtracting the sum of the pixel values within the black portions of the box from the sum of the pixels within the white portions. These filters were introduced in [115] in a similar context, where the set of responses of such filters was used as a feature space for ensemble face detectors. It was also shown in [115] that each response can be computed very efficiently using the integral image transform (in which the value in each position is the sum of the pixels above and to the left from it.) We used three types of such filters, all seen on Figure 5-1: a two-part (vertical or horizontal), a three-part (vertical or horizontal), and a four-part "checkerboard". Filters of each type are slid, at multiple scales, through the image similarly to the multi-scale EDH. The resulting representation for a 180×60 observation contains more than 200,000 values.

Pose-invariant yaw classification

We treat the problem of estimating the yaw as a regression problem where the range of the target function is the discrete set of yaw bin centers $c\pi/36$. The precision (bin width) of ten degrees was chosen by examining its potential effect on the second stage of the process–the pose classification. We have observed that, on the one hand, finer divisions of the yaw range do not seem to significantly reduce appearance variation between images rendered for the same body pose for angles within the same bin, and on the other hand, coarser divisions seem to significantly increase such variation.

Once formulated as a regression with a set precision threshold, this problem naturally fits the framework outlined in Section 2.2.1. We therefore define a yaw-sensitive similarity between two images $\mathcal{S}_{yaw}$ to be +1 if the underlying yaws fall in the same bin–ignoring the pose and the location! Examples of a similar and of a dissimilar pair under $\mathcal{S}_{yaw}$ are shown in Figures 5-2 and 5-1.

Using this definition we form a training set of similar and dissimilar pairs, and apply the Boosted SSC algorithm. To make sure the training examples are representative of the poses while keeping them of manageable size, we apply k-means clustering [38] to find 50 centers of pose clusters. We then construct each of the 4,000
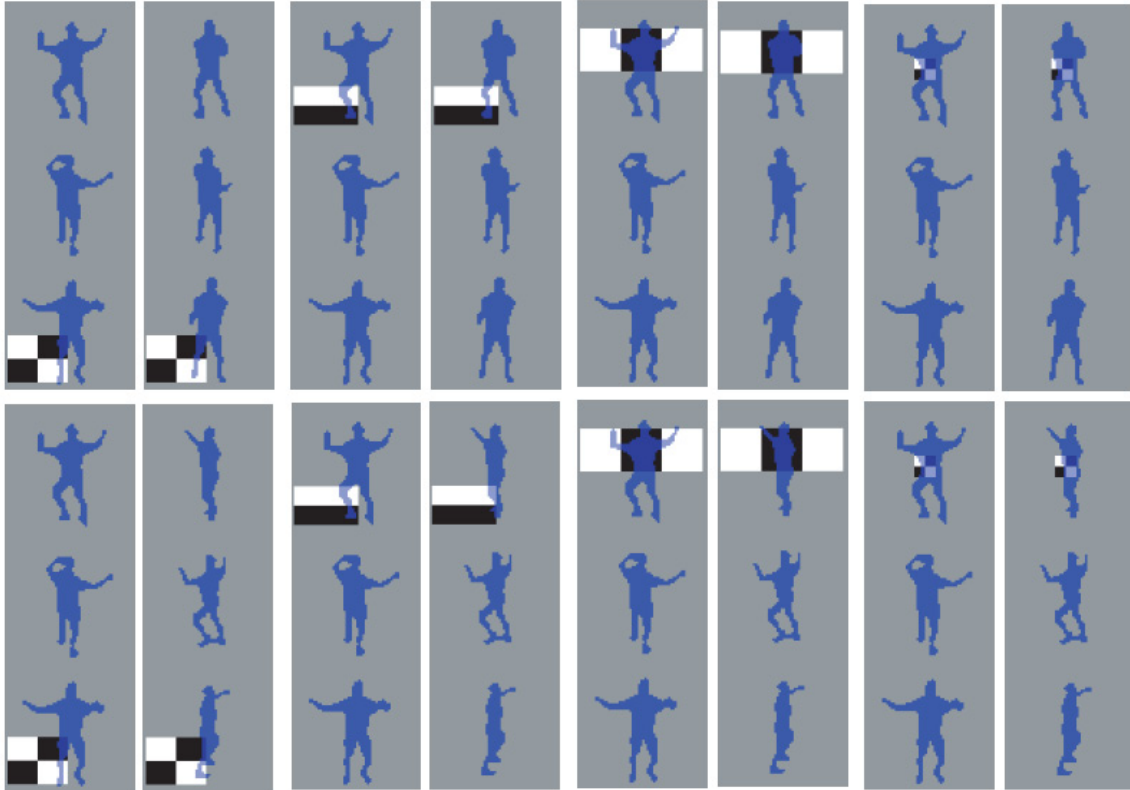
Figure 5-1: Some of the projections (filters) learned by Boosted SSC for yaw similarity. Top row: positive examples. Bottom row: negative examples. A projection corresponding to a box filter is computed by subtracting the sum of pixels under the rark regions of the box from the sum of pixels under the white regions. From [93].
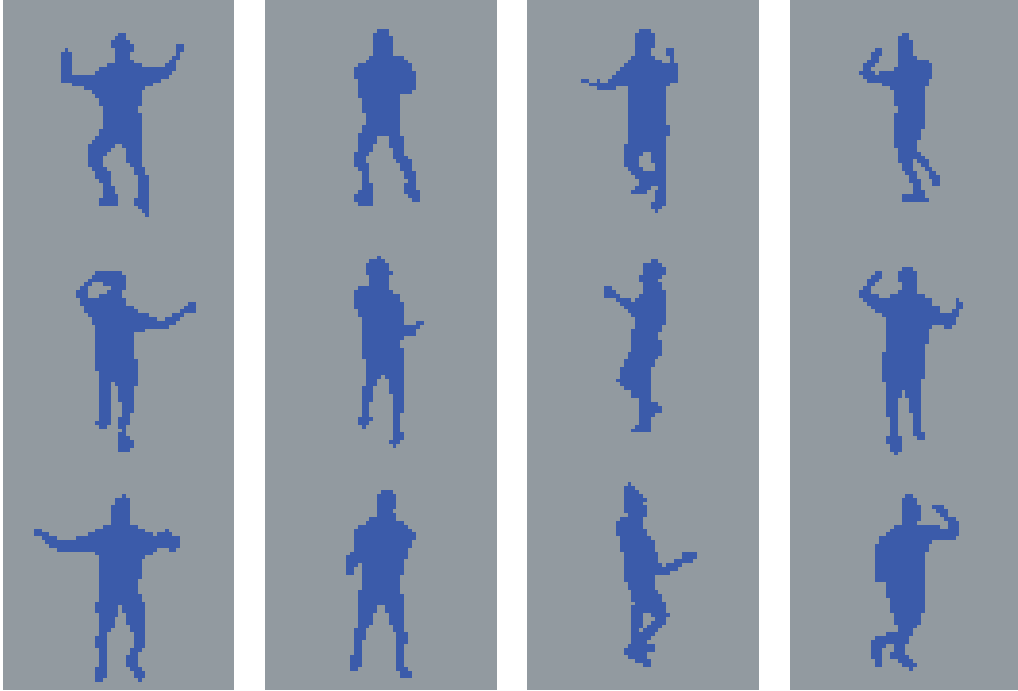
Figure 5-2: Positive (left pair) and negative (right pair) training examples for yaw similarity; pose and location are ignored in determining $\mathcal{S}_{yaw}$. From [93].

positive examples by choosing a random bin, selecting two of the cluster centers, and selecting a random location (out of five possible) for each. 6,000 negative examples are formed by a similar process, pairing cluster centers across bins. Increasing the number of training examples beyond 10,000 is impractical (recall that with the finite number of projections, as in this case, each iteration of Boosted SSC involves examination of the value of each projection on each pair.)

We have also set up an additional set of labeled pairs (300,000 positive and 37,000,000 negative), not used in training, to serve as validation set. Testing on this set reveals that 10,000 examples may not be sufficient to cover the space adequately. Therefore, we follow the *resampling* approach [106]. We maintain a much larger set that serve as a "pool" of training examples (800,000 positive and 2,000,000 negative). Every 40 iterations of boosting, we resample a 10,000-example training set of pairs by the following procedure:

1. The response $y_i = \sum_m \alpha_m c_m(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)})$ of the current ensemble classifier is computed for each example pair in the pool.

2. Each example is assigned a weight $\exp(-l_i y_i)$.

3. The weights are normalized to form a distribution.

4. A new 4,000+6,000-strong training set is sampled according to that distribution; the AdaBoost distribution is set to uniform.

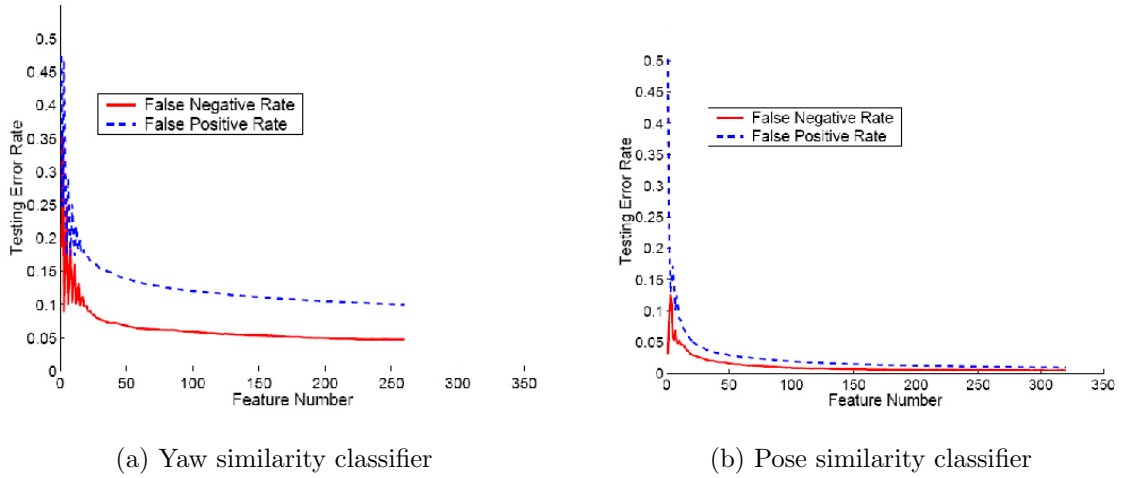(a) Yaw similarity classifier   (b) Pose similarity classifier

Figure 5-3: Test errors for yaw (top) and pose (bottom) similarity classifiers, as a function of the number of features collected by AdaBoost. From [93].

Note that the computational cost of this algorithm is quite high, even with this relatively small fraction of examples explicitly examined. Fortunately, most of the steps in each iteration, including the resampling steps, can be trivially parallelized, by dividing the data and the features among processes. Most of the computation is done in parallel, and only modest amount of inter-process communication is required at each iteration to combine the error estimates, select the winning weak classifier and distribute the updated parameters such as α to each process. We have implemented such a parallelized version of the algorithm, and ran it on a Beowulf cluster, using between 80 and 120 processes.

Figure 5-3(a) shows the behavior of the error on the validation set as boosting proceeds. As can be seen, the error is steadily decreasing until it levels off, and the algorithm is stopped after 260 iterations. We thus have obtained a weighted 260-bit encoding that corresponds to $\mathcal{S}_{yaw}$. the estimation of the yaw for a new observation follows the paradigm presented in Section 2.2.1, using a 20-NN classifier with L_1 distance in the embedding space. Due to the real time speed requirement of the application, we use LSH (Section 2.4.2) to perform the search for neighbors. To deal with potential failures due to mirror symmetry of the silhouettes, the actual estimate during tracking is made more robust by the following procedure:

1. Instead of taking a simple majority vote, we build a histogram of the labels (yaw values) of the 20 NN and smooth is with a low-pass filter.

2. Two highest peaks in the histogram are located.

3. We form a set of four candidates from those peaks and their mirror reflection (by adding π to each).
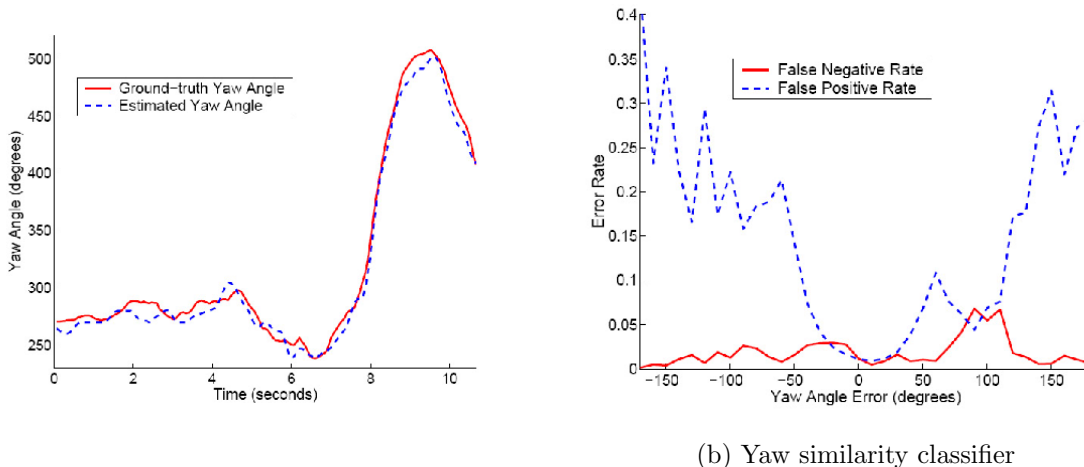
103

(b) Yaw similarity classifier

Figure 5-4: 5-4(a): performance of the yaw estimator on a real video sequence of swing dancer. Solid: ground truth from motion capture; dashed: estimate. 5-4(b): effect of error in yaw on the resulting pose similarity classifier. The correct yaw in this case is zero. From [93].

4. Out of these four candidates, we choose the one closest to the estimated yaw in the previous frame.

Figure 5-4(a) shows an example performance of the final yaw estimator on a real video recorded simultaneously with the motion capture. For 73% of the frames in this sequence, the error in yaw estimate is below 10 degrees - i.e., a correct yaw bin assignment under our definition of sensitivity. In 92.5%, the error is below 20 degrees, and in 98% below 30 degrees. To understand how tolerable are errors of different magnitude, we tested the effect of incorrect yaw estimation on the accuracy of the resulting pose similarity classifiers (we describe learning these classifiers below). The results for the case of zero degrees yaw, shown in Figure 5-4(b), imply that errors of up to 30 degrees do not cause significant increase in the error of pose similarity classifier. We believe that this is explained by the relatively smooth transitions between the appearances of the same pose in neighboring yaw bins. It is interesting to consider the difference between false negative and false positive rates: the latter is significantly higher, in particular for yaw estimates with large error, since the responses of the similarity classifier in that regime become more random, and as we have noted in Chapter 3 it is "easier" to misclassify a dissimilar pair by a random set of projections/thresholds.

Yaw-specific pose estimation

For each yaw bin β, we learn a classifier of the pose similarity $\mathcal{S}_{pose}^{\beta}$ defined according to (4.1), with $\mathcal{D}_{\theta}$ being the L_2 in the pose parameter space. The learning proce-

104

Figure 5-5: Positive (left pair) and negative (right pair) training examples for pose similarity for a fixed yaw. From [93].

dure closely follows the one described above for the yaw similarity learning, with the following differences:

- For positive examples, all the 5,120 poses, rather than 50 cluster centers, are used, since the similarity rate for the pose is significantly lower than for yaw, and the total number of potential positive pairs is much lower. For negative examples, we use clustering in the way described above.

- There are 3,000 positive and 5,000 negative examples in the training set, and 200,000/10,000,000 examples in the resampling pool.

- Resampling occurs every 80 iterations.

- The number of boosting iterations is 320.

As can be seen in Figure 5-3(b), the behavior of the error of a typical pose similarity classifier follows the same trend as for the yaw classifier. However, the absolute level of the error is much lower. This reveals that the difficulty of the yaw estimation task significantly exceeds that of the pose classifier. We relate that to two factors. One is the larger visual diversity among the examples seen by the yaw similarity classifier, versus any of the 36 pose similarity classifiers. The other factor is the much smaller, in absolute numbers, set of potential positive examples in the case of pose

estimation; as a result, it is possible to represent those better in the training set provided to the learning algorithm, whereas in the case of yaw classifier we must resort to clustering as a pre-processing step.

Once the pose similarity classifiers are learned, we could follow the same approach as with yaw estimation, i.e. convert the classifier to an embedding, and build a NN-based regression mechanism using the L_1 in the embedding space. However, recall that under the motion graph model we need, for frame t, to consider a finite number of examples - the nodes $\theta_1^{(t)}, \ldots, \theta_{n_t}^{(t)}$ in terms of Equation 5.3. The number of currently maintained paths n_t is typically small due to relatively low branching factor of the motion graph. Therefore, we can afford to use the similarity classifier explicitly! In other words, we can form n_t pairs of observations $(\mathbf{x}^{(t)}, \mathbf{x}_i^{(t)})$ and compute the response[6] $\sum_{m=1}^{M} \alpha_m c_m(\mathbf{x}^{(t)}, \mathbf{x}_i^{(t)})$.

5.2.3 Performance

The system described in this section has been successfully tested on a number of real video sequences. To obtain a quantitative measure of error, an evaluation was done on one such sequence for which motion capture data, recorded simultaneously with the video, was available. On that sequence, performance of our system was compared to the performance of a system using a commonly used set of features–Hu moments [63]. Hu moments are based on seven moment invariants of a binary shape 9the silhouette in this case). The purpose of this evaluation was to try to ascertain the effect of learning similarity and of using the box filter representation.

Figure 5-6 shows the histogram of error *differences*; the superior performance of the learned similarity representation over Hu moments is clear. In 86% percent of the frames, the error with Hu moments exceeded the error of our system. The effect of the learned embedding (or equivalently the learned distance measure) is further illustrated in Figure 5-7. The figure shows some typical examples of poses retrieved by the nearest neighbor search with respect to the ground truth, the embedding distance, and the distance in Hu moment space (the results shown were obtained with no temporal information to emphasize the effect of the space/distance on the retrieval.) The average per frame $\mathcal{D}_\theta$ error of the embedding NN was 44cm, compared to 25cm with the "hindsight" NN (finding the best match knowing the correct ground truth value) and to 76cm with Hu moments.

Finally, in addition to measuring the accuracy of pose estimation on a per-frame basis, extensive evaluation was done to compare the quality of the resulting animation to that obtained with state-of-the-art methods in the field; see [93] for details.

---

[6]We found that ignoring the weights α_m speeds up the computation, which is then reduced to calculating Hamming distance, with no significant effect on the results.
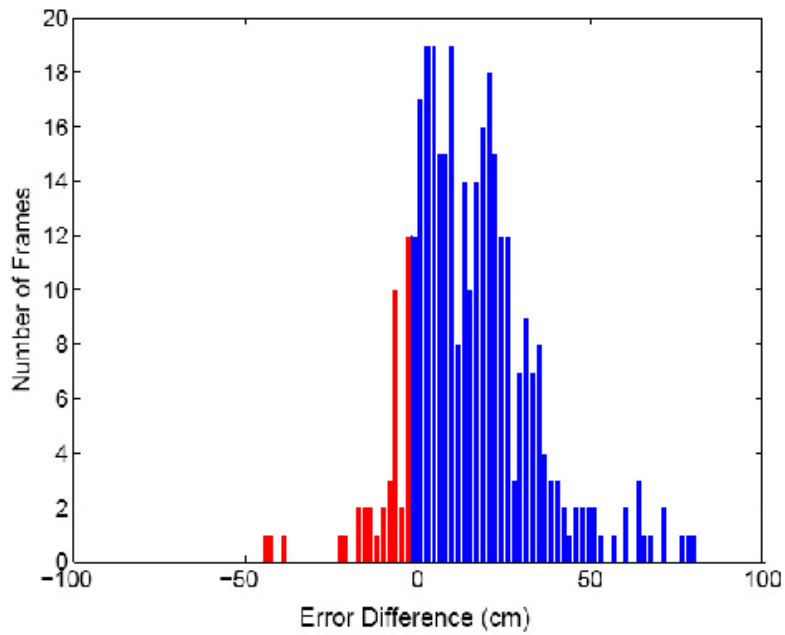
Figure 5-6: Histogram of differences between the $\mathcal{D}_\theta$ error of 1-NN in Hu moments and the 1-NN error in the learned similarity embedding; positive values mean smaller error for our system. Obtained on one real dance sequences with measured ground truth. From [93].
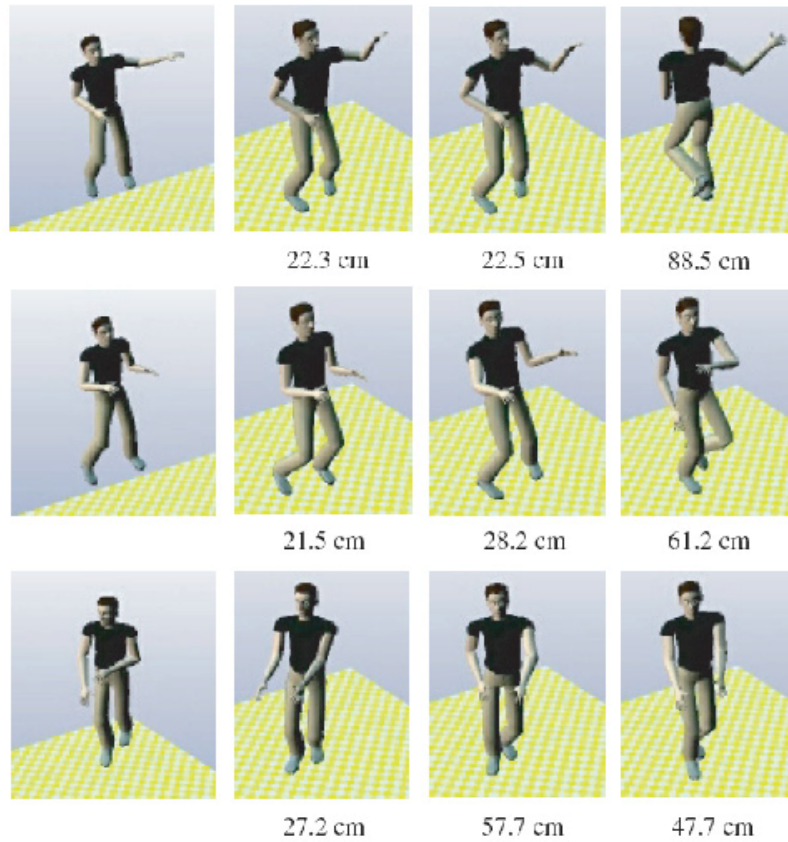
Figure 5-7: Representative examples of pose retrieval accuracy with various methods. Each row corresponds to one frame. Columns, from left to right: Ground truth; best match in the database by brute search; best match in the embedding space; best match with Hu moments. Numbers: L_2 error in the pose space. From [93].

5.3 Case II: General pose tracking with likelihood modes

If the goal of the tracking application is not restricted to a specific domain with constrained and structured motion, continuous models of dynamics are more appropriate. The system described in this section implements such a model. It is however unusual in that it avoids the problem shared by many state-of-the-art tracking algorithms: being led astray by an inaccurate prior. This is achieved by using a weak prior, and using similarity search in an embedding space to obtain the peaks of the likelihood function directly from the observation. Since such search can be implemented extremely fast using LSH, the resulting tracking system is also very fast.

5.3.1 Pose similarity as likelihood sampling

As mentioned in Section 5.1.3, we adopt the assumption that the examples closest to the current observation in the similarity embedding space are, with high probability, close to peaks of the likelihood. More precisely, we assume that at least one example among those returned by a K-NN search in the embedding space will correspond to each high peak in the likelihood.

Following this assumption, we can think of searching a large database of labeled images under a learned pose similarity as a very fast evaluation and sampling of approximate likelihood. The examples not returned by the search (i.e., the vast majority of the database) are considered to have been pruned away by this procedure, which is taken to mean that their likelihood is low. The returned examples are treated as candidates for high likelihood. However, since the search is approximate, we do not directly use these examples in estimation, but rather use them as starting points for local optimization.

The likelihood optimization in our system uses the Iterative Closest Point (ICP) algorithm [15] to find a local optimum of the match between a set of points on the surface constructed for the articulated model and a corresponding set of points on the estimated surface of the observed body (the latter is obtained by applying a standard stereo matching algorithm.)

5.3.2 Tracking with likelihood modes

The main motivation behind the algorithm described here is to avoid a typical failure mode of tracking algorithms that rely on the prior: being led astray and losing track due to inaccurate, and overly confident, prior. One reason for using a strong prior is that it provides a starting point for an optimization step in which the model is improved with the objective to increase the likelihood (and thus the posterior), typically by means of an iterative gradient ascent algorithm. This strategy often is reasonable, however it may fail in cases of occlusion, unusually fast or slow motion, and simply when the tracker made significant mistakes in the previous few frames. Applying
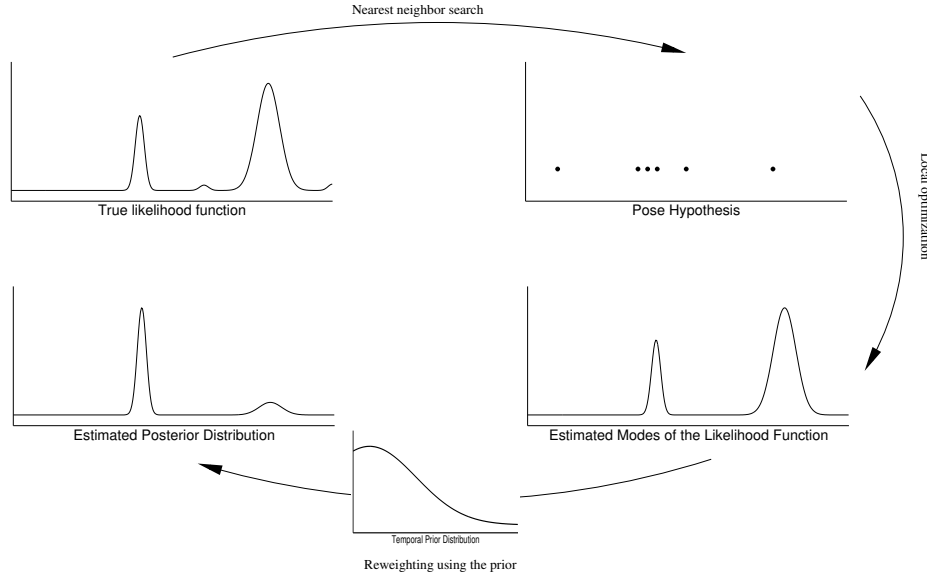
Figure 5-8: Schematic illustration of the estimation flow in ELMO. The temporal prior is obtained by multiplying the posterior from the previous frame by a high-bandwidth Gaussian. From [35].

strong prior model in such cases results in a bad starting point for the gradient descent, which may have critical effect on the quality of the eventual result [108].

The operation of our tracking algorithm (called ELMO: Tracking with Likelihood MOdes) is schematically shown in Figure 5-8. At every frame, we maintain a parametric estimate of the pose posterior in the previous frame in the form of a mixture of Q Gaussians. A temporal prior is obtained from this estimate by multiplying the posterior by a high-bandwidth Gaussian window, in accordance with the diffusion dynamics model.

Given the new observation, we compute its embedding into a space H and retrieve the poses corresponding to K top matches in H. As mentioned above, our assumption is that the poses associated with the retrieved examples have, with high probability, high likelihood given the current observation. Each of these K candidate poses is then used as a starting point for an ICP search to find the adjacent local maximum of the explicit likelihood. Once these modes of the likelihood are found, we multiply them by the prior obtained, as mentioned above, by smoothing the posterior from the previous frame. The result is the estimate of the posterior in the current frame.

5.3.3   Implementation and performance

In our system we use the concatenated multi-scale edge direction histogram (EDH) as the feature space, as described in the previous chapter. Using a combination of color background model with stereo volume information we extract a bounding box of the silhouette and normalize it to 200×200 pixels. With 3 scales (8, 16 and 32 pixels) and with location step size of half the scale, the EDH yields $N = 13,076$

bins. Applying SSC , and specifying the minimum TP-FP gap of .1, we obtained $M = 3547$ embedding dimensions.

Having encoded the data (described in Section 4.5) according to the learned embedding, we constructed LSH with $l = 50$ hash tables with $k = 18$ bit keys. At the run time, we retrieve for every frame $K = 50$ training examples and use their poses to initialize the ICP.

Using six synthetic sequences generated based on human motion capture data, obtained from [41], we compared the performance of ELMO to that of two alternative algorithms: the state-of-the-art Condensation particle filtering approach [36], with 1,000 particles, and the ICP differential tracker [34]. We also measured the per-frame error resulting from retrieving the NN under the SSC embedding directly. The sequences are rendered with significant perspective distortion, relatively wide range of motions, much self-occlusion and are therefore challenging for a tracking algorithm.

The summary of the results of this comparison appears in Figure 5-9. As can be seen, the average error with ELMO is the lowest. Figure 5-10 shows the per-frame account for segments from two of the synthetic sequences. The error with ELMO is the lowest in most of the frames. The performance of Condensation was the worst among the tracking algorithms (and only better than the per-frame static estimation with SSC that ignores dynamics). It is possible that with additional particles Condensation would improve its performance. However, with 1,000 particles it is already two orders of magnitude slower than the other methods.

It is also interesting to note that although the average performance of the single-frame pose estimation with SSC is inferior to that of proper tracking algorithms, it fares relatively well, with the error being less than a standard deviation above the average errors of Condensation and ICP trackers. Looking at the sequence data we can see that in fact there are segments in which it does better than those algorithms. This provides some insight into the success of ELMO: relying on the robust single-frame estimator allows it to recover from some severe errors, since at most after a few frames the matches found with LSH in the embedding space become again close to the likelihood peaks. This is also related to an additional advantage not explicit in these figures: ELMO does not need manual initialization, instead using the NN search from the very first frame.

In addition to the tests with synthetic data, where exact error evaluation is possible, we evaluated the performance of ELMO on a number of real video sequences. Figures 5-11 and 5-12 show a few frames from two of such sequences. Notably, the frame rate in these sequences was only four frames per second, creating significant inter-frame pose differences. With tracking algorithms strongly depending on the prior, this would normally pose much difficulty. However, ELMOdeals with this gracefully.

5.4   Discussion

In both of the tracking systems described in this chapter, an example-based pose estimation algorithm is a crucial component. In the swing dance animation system it
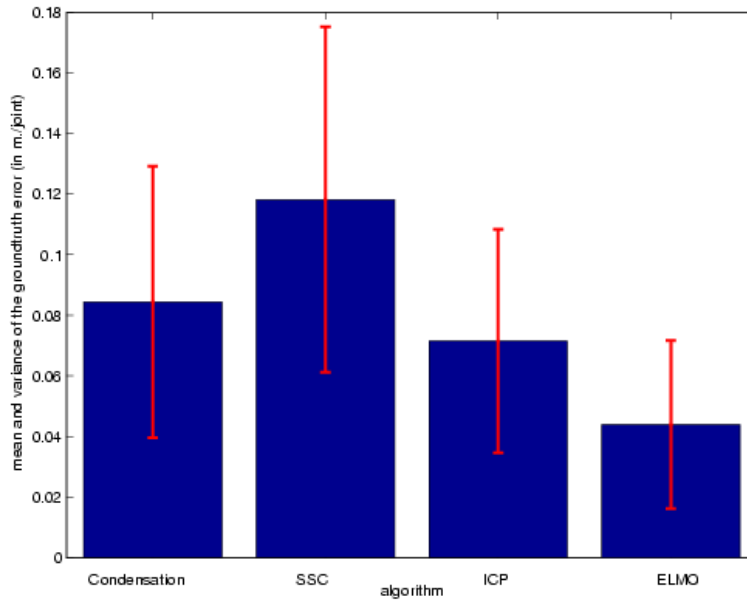
Figure 5-9: Average and standard deviation of the ground truth error obtained using Condensation, SSC (PSH, in terms of [35]), ICP, and ELMO on six sequences of 1000 images each. ELMO outperforms the Condensation, ICP and SSC algorithms. The average error per joint for ELMO is less than 5 cm. From [35].

allows for efficient evaluation of multiple hypotheses. In ELMO, it provides a means of automatic initialization and re-initialization, and in combination with the weak temporal model for the prior makes the tracker qualitatively less vulnerable to losing track.

Two key properties of our similarity learning approach that make it possible are the classifier/embedding duality, that enables us to reduce the problem of evaluating a large hypothesis space to the problem of search in a database for neighbors under the L_1 distance, and the setup for learning the similarity specific to the task at hand. The latter, as exemplified in the swing animation system described in this chapter, allows us to break down complex estimation problems into much simpler ones, leading to a winning divide-and-conquer architecture.
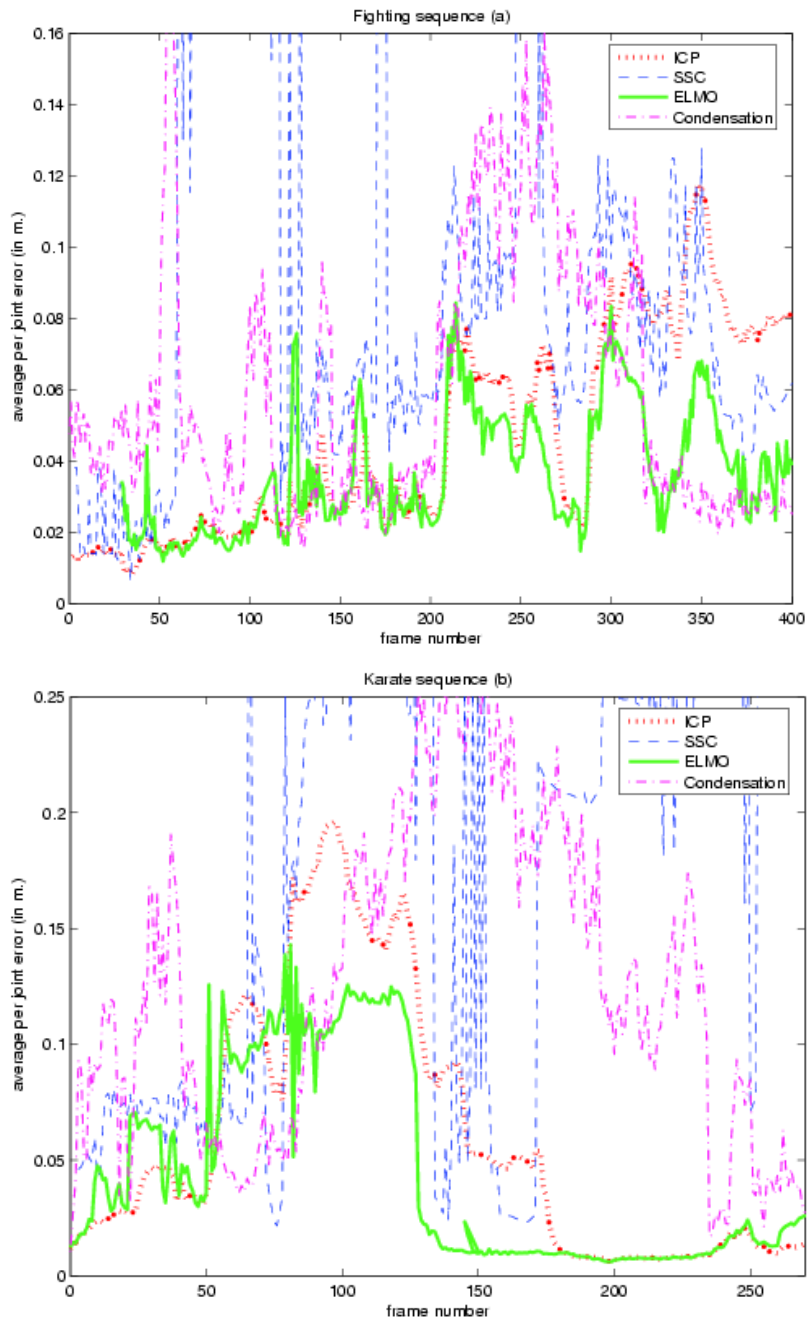
Figure 5-10: Tracking results on two of the six test sequences (for better clarity, only segments of the sequences are shown). The graphs report the ground truth error (vs. frame number) corresponding to Condensation, SSC, ICP and ELMO. (a) Fighting sequence, (b) Karate sequence. The error corresponding to the ELMO algorithm is almost always the smallest. From [35].

113
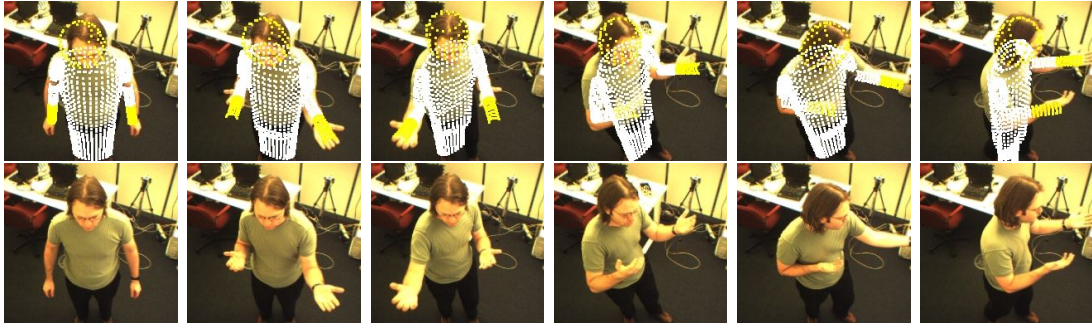
Figure 5-11: Tracking results extracted from the *dance* sequence. From [35].
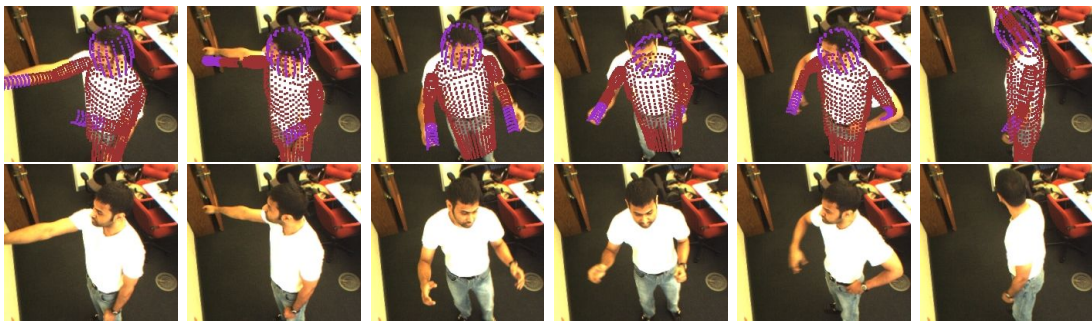
Figure 5-12: Tracking results extracted from the *whiteboard* sequence. From [35].

Chapter 6

Learning Image Patch Similarity

The ability to compare image regions (*patches*) has been the basis of many approaches to core computer vision problems, including object, texture and scene categorization. Developing representations for image patches has also been in the focus of much work. However, the question of appropriate similarity measure between patches has largely remained unattended.

In this chapter we focus on a specific case study: learning similarity of natural image patches under arbitrary rotation and minor shift. Figure 6-1 shows a few examples of groups of patches that are similar to each other. Such a definition may not be broad enough, that is, some patches that do not fit this description still may be considered similar in the context of a given task. For instance, if the goal is to categorize an object, rather than recognize a specific instance, patches that look visually dissimilar at this low lever may still correspond to semantically matching parts.[1] However, we believe that for many reasonable tasks that involve matching patches, the underlying definition of similarity must include this limited case.

Rather than developing a new representation that is invariant to the transformation in question, we will consider two popular representations of patches: sparse overcomplete code and scale-invariant feature transform (SIFT). Specifically, we will investigate how the tools developed in this thesis can be used to improve matching with these representations.

Section 6.1 provides some background on algorithms that use patch matching, and reviews approaches to measuring patch similarity. We define the target similarity concept in Section 6.2, and the patch descriptors used in the experiments are introduced in Section 6.3. We then describe, in Section 6.4.2 an experimental evaluation that demonstrates an improvement in matching with both of the descriptor types by using a similarity-driven embedding learned with BOOSTPRO.

---

[1] Note that the framework presented in this chapter can be extended to learn such a similarity as well, as long as examples are provided.

6.1 Background

The main context in which comparing image patches has emerged in computer vision is that of high-level vision tasks, that can be described as scene understanding. This includes:

Object recognition This means finding a specific object: a face of a certain person, a shoe of a particular make, a magazine etc.

Object categorization and object class detection Rather than looking for a specific instance of an object, the interest here is in all objects that belong to a certain class, for an appropriate definition of the latter: any face, any car, etc.

Entire image classification Sometimes the goal is not to localize, or determine the presence of, an object but rather to assign the entire image to a certain class. For instance, location recognition and texture classification belong to this category of tasks.

Broadly speaking, approaches to these tasks can be divided into three groups with regards to how they represent image information. The first group consists of methods that rely on *global features*. This may include methods that collect a histogram of measurements over an entire image [82, 101], or shape matching techniques that directly model an entire shape in a parametric form [104, 74].

The second group consists of methods that operate on local image features, but do not directly operate on image patches. This includes methods that compute histograms of measurements over a limited region, be it measurements of shape [11, 55] or filter responses [115, 101].

Finally, the third group, of most relevance here, directly operates on image patches. Most of the methods in this group involve, in addition to matching the patches, a geometrical reasoning component. This component may involve a full model of joint location of parts as in constellation models [46, 71], or a more loose set of constraints like in random field models [91], or fragment-based approaches [3, 9]. Some methods that have been proposed avoid modeling geometry altogether [107].

It should be emphasized that in addition to recognition and classification, other tasks may benefit from patch matching paradigm. Notable examples are fragments-based segmentation [17] and wide-baseline stereo reconstruction [81].

6.1.1 Patch similarity measures

The question of measuring similarity between patches has not received very much attention in the computer vision literature. Usually, a standard distance measure is adopted for whatever representation is used.

Pixel-based distance

The simplest similarity measure consists of directly comparing the pixel values of the two regions, e.g. by means of the L_1 distance

$$D(\mathbf{x}_1, \mathbf{x}_2) = \sum_d |\mathbf{x}_{1(d)} - \mathbf{x}_{2(d)}|.$$

This is rarely a useful measure, since it is extremely sensitive to minor transformations, both in geometry (shifts and rotations) and in imaging conditions (lighting or noise).

Correlation

Normalized correlation between patches $\mathbf{x}_1$ and $\mathbf{x}_2$ is defined as

$$NC(\mathbf{x}_1, \mathbf{x}_2) = \frac{\sum_d \left(\mathbf{x}_{1(d)} - \bar{\mathbf{x}}_1\right) \left(\mathbf{x}_{2(d)} - \bar{\mathbf{x}}_2\right)}{\sigma_1 \sigma_2},$$

where $\bar{\mathbf{x}}_i$ and σ_i are the mean and standard deviation of pixels in $\mathbf{x}_i$. Because of the factoring in of the means it is much more robust than the pixel-wise distance. Normalized correlation has been used extensively in fragment-based recognition [3, 9], where it is assumed that viewing conditions are fixed, or alternatively that there exist examples from all viewing conditions–in other words, not matching a patch to a version of itself rotated by 90 degrees is acceptable. We would like to avoid such an assumption.

Descriptor distance

Another popular method is to compute a descriptor of each patch, and then simply apply a distance measure on the two descriptors. Most commonly the descriptors are vectors in a metric space of fixed dimensions and the distance of choice is L_1. Matching with SIFT descriptors, discussed in detail in Section 6.3.2 is perhaps the most popular example of such an approach [76]. Another popular descriptor is the *shape context* [11], often used when shape is believed to be the crucial component of the recognition system. Shape contexts are based on the local histogram of contour points in the vicinity of the selected location. The distance of choice for shape contexts is typically χ^2.

Probabilistic matching

A different approach is taken by some of the methods that instead of measuring distance between representations patches, evaluate directly the probability that the two patches belong to the same class. This is usually limited to models in which a fixed number of patch classes, called *parts*, are combined in some framework. A well known example of this kind is the family of constellation models [19, 46].
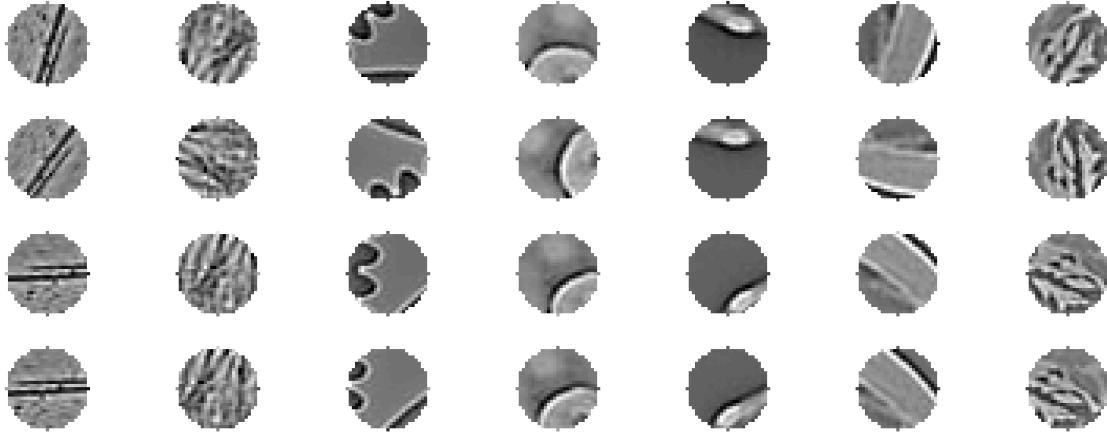
Figure 6-1: Examples of similar patches from whitened images. Each column contains rotated and shifted versions of the same original patch, so that by definition patches in each column are similar.

6.1.2 Interest operators

An important aspect of any patch-based technique is the method for selecting the patches that are subsequently evaluated for similarity–both in the test image and in labeled training images. This issue is often coupled with that of descriptors and similarity, since the invariance in the descriptor is induced by providing it with an estimate of the transformation of the patch with respect to some canonical reference frame, for instance, the angle necessary to align the main axis of a region with the vertical axis, or the size of a region.

There is a large number of interest operators, and extensive evaluation has been undertaken in a number of cases [102, 77, 71, 84]. On the other hand, the role of these operators (and the role of the keypoint concept, in general) in object recognition is still somewhat controversial: recent results in [9, 14, 80] show that excellent performance can be achieved without using keypoints or interest operators at all. In general, we would like to remain agnostic on this issue, and focus on the analysis of descriptors and their role in similarity matching. When necessary (namely, with SIFT descriptors) we will assume that an appropriate detector has been applied, and therefore will make available the basic information that would normally be provided by such a detector (location and scale of the patch).

6.2 Defining and labeling visual similarity

For the purposes of the study in this chapter, we consider two patches to be similar if they could be obtained by taking an image and then rotating (and/or shifting by a small amount) the imaging device and taking an image again–and extracting the patches from the same image location. Equivalently, the two patches are similar if there exists a transformation, consisting of a shift by between zero to two pixels, followed by an arbitrary in-plane rotation, that makes the two patches be identical,

up to pixel-level aliasing and possible boundary effects due to shift.[2]

In terms of image representation, this definition of similarity resembles the low-level invariance believed to exist in the lateral geniculate nucleus (LGN) and higher areas in the primate visual cortex [39, 64]. It also matches an intuitive notion of visual similarity that is related to semantics of the visual world: two patches are similar if they correspond to the same element of a physical scene. This leads to the idea underlying the *slow-feature analysis* technique for unsupervised learning of spatio-temporal coding, and in particular invariance to transformations, in the visual cortex [116]. In SFA, a stimulus to the learning algorithm is constructed by simulating a "natural movie"–a sequence of inputs obtained by moving a receptive field slowly (i.e., by applying only mild transformations to obtain the next frame) in the image. The objective of the algorithm is, in effect, to learn features that are efficient in encoding such sequences.

We take this idea and apply it to the task of labeling similar image patches. For a given image patch in a natural image, any number of patches similar to it may be transforming the receptive field according to the desired similarity. So, if we want to ignore the rotation, images obtained by rotating the receptive field at a fixed location can serve as examples of similar pairs. If shifts by up to a certain number of pixels are to be ignored, any two shifts within those bounds will produce image similar to each other, etc.

Figure 6-1 shows a few examples of sets of similar patches generated by the procedure outlined above. Patches in each column are versions of one patch (top row), rotated by a random angle or shifted in random direction by 2 pixels.

This notion of similarity adheres to the definition of equivalence given in Chapter 1. Also, as mentioned in Chapter 2, this notion of equivalence does not translate to a transitive equivalence relation in the patch space: taking a shifted patch and shifting it again will, again, create a similar pair, but the third patch may no longer be similar to the first one.

6.3   Patch descriptors

We will be focusing on two descriptors that have very different properties and were designed under very different objectives. The first one, the coefficients of a sparse overcomplete code, corresponds to a generative model of the patch, and is by design not invariant to transformations. The second, SIFT, is a constructed with a discriminative task in mind, specifically to be invariant under shift and rotation (when used in conjunction with an interest operator).

---

[2]Note that we work with patches shape like a disk (up to the pixelation aliasing artifacts) rather than a more common rectangle. This is to diminish the artifacts introduced in the corners by rotating a rectangular patch.
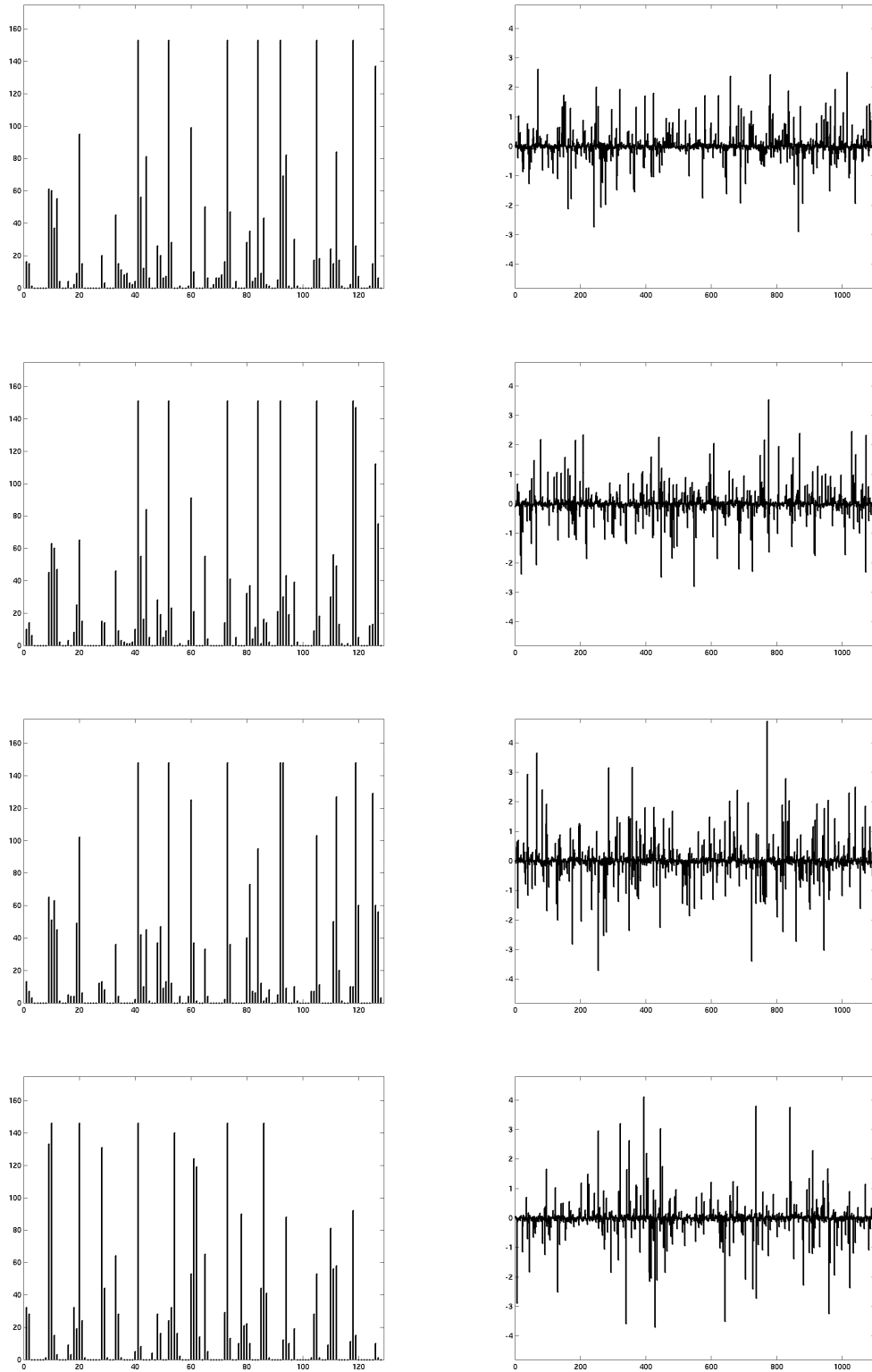
Figure 6-2: Descriptors for the patches in the third column from left in Figure 6-1. Left: SIFT descriptors. Right: coefficients of a sparse overcomplete code.

6.3.1 Sparse overcomplete code

A large body of work has been focused on construction of codes for natural images that would posses the key properties of the V1 cell populations, namely orientation selective and localized receptive fields. It has been shown in a number of studies [89, 64] that these properties emerge as a result of a learning procedure whose objective is to maximize the fidelity of reconstruction along with maximum sparseness of response. The latter is believed to be an important principle of sensory coding in the brain, related to the statistical properties of natural visual scenes[47].

Specifically, a sparse overcomplete code defines a generative model of the patch as a linear combination of C *basis functions*, that are themselves patches of the same dimension:

$$\mathbf{x} = \sum_{c=1}^{C} a_c \phi_c = \mathbf{\Phi a}. \tag{6.1}$$

The general objective of constructing this code is, naturally, to reduce the reconstruction error, which is measured by the energy in the residual. Under that objective alone, if we fix the basis functions, the optimal decomposition of a patch $\mathbf{x}$, in terms of the coefficients $\mathbf{a} = [a_1, \ldots, a_C]^T$ could be found as

$$\mathbf{a}^* = \operatorname*{argmin}_{\mathbf{a}} \|\mathbf{x} - \mathbf{\Phi a}\|. \tag{6.2}$$

However, two key properties of the codes we are discussing here are *overcompleteness* and *sparseness*. The former means that the number C of basis function is higher than the dimension of $\mathbf{x}$. The latter means that a majority of coefficient have very low absolute value for any given patch. As a consequence of overcompleteness, (6.2) will generally not have a unique solution. This is where the sparseness property becomes important: the optimization criterion is augmented by a penalty term, that drives the coefficients to zero. This can be written [89] as

$$\mathbf{a}^* = \operatorname*{argmin}_{\mathbf{a}} \|\mathbf{x} - \mathbf{\Phi a}\| + \lambda \sum_{c=1}^{C} S(a_c), \tag{6.3}$$

where $S(a_c)$ is an appropriate cost function that penalizes values away from zero. The specific form of this function is a matter of design; it should be noted that a choice of S corresponds to imposing a statistical model on the distribution of the coefficients (prior). For instance, the cost function

$$S(a) = \log(1 + a^2)$$

used in our experiments (following [89]) can be shown to correspond to the Cauchy prior.

It is important to distinguish this coding scheme from coding with non-overcomplete representations. The main difference is that there is no closed-form solution to the optimization, and it has to approached with an iterative optimization algorithm per-

forming gradient descent on the cost function. The algorithm, suggested in [89], starts with a random basis; a number of random patches are extracted from the training images, and the optimal coefficients with respect to the current basis are computed by means of gradient descent on (6.3). Given these coefficients, the basis functions are updated, with the objective to decrease the residual. The update rule is

$$\Delta\phi_c \;=\; \eta a_c \left(\mathbf{x} - \mathbf{\Phi a} \right),$$

where η is the learning rate. The process is repeated iteratively, until no further significant changes in $\mathbf{\Phi}$ are recorded.

Such codes have been successful in low-level vision tasks, such as image denoising [98]. Some properties of this representations make it potentially appealing for recognition purposes: Sparseness makes the code likely to provide good discriminative power [8, 92, 3], and high reconstruction fidelity means that they encoding will likely retain relevant information from the original image patch. A key question though is how to compare two patches encoded in this fashion.

The right column in Figure 6-2 shows the coefficient vectors for four similar patches (that appear in the third column from the left in Figure 6-1.) One immediate observation from the figure is that there is a significant variation in the codes; this suggests that the naïve use of L_1 between the coefficient vectors $\mathbf{a}$ may not be a good choice of similarity measure between patches.

The instability of the sparse overcomplete code is in fact a direct consequence of its key properties. Consider a particular patch, for which optimizing the coefficients in (6.2) makes a coefficient a_c to have a high absolute value. If the same patch is shifted by one pixel, the basis function ϕ_c will probably still account for some of the patch appearance reasonably well. However, there likely (due to the overcompleteness of the code) will be another basis function ϕ_j which will account for the shifted patch even better. Furthermore, the sparseness constraint will encourage the absolute values of both a_c and a_j to decrease. As a result, it will be more optimal from the perspective of reducing the cost function value to "keep" ϕ_j and suppress ϕ_c for the shifted patch. This explains the typical pattern in which even a small variation in the patch may cause significant changes in the code, and the resulting effect on similarity between transformed patches.

6.3.2 SIFT

The SIFT descriptor [77] is based on a histogram of oriented gradients within the region it describes. It is computed for a known location, scale and orientation in the image (provided by an interest operator). The descriptor is computed in the following way (the parameters given here are the ones used in our experiments, and can be varied when implementing the algorithm):

- The calculations are based on the appropriate level in the Difference-of-Gaussian pyramid, rather than on the original image. This induces scale-invariance.

- The orientation and magnitude of the intensity gradient are computed on a
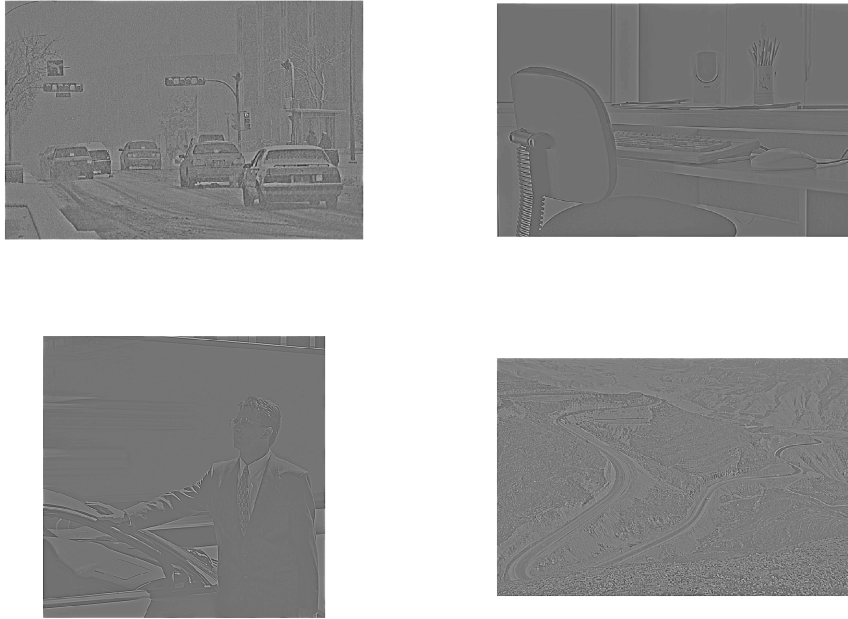
Figure 6-3: Examples of whitened images (four out of 100) used to train and test patch similarity measures.

fixed 16×16 grid in the vicinity of the target location. The values are weighed by a high-bandwidth Gaussian window centered at the location.

- A smoothed histogram of gradient orientations is computed for each subregion of 4×4 grid locations. With 8 bins in the histogram, this yields a 128-dimensional descriptor.

Note that the invariance to scale, rotation and translation with this descriptor are subject to an accurate estimation by an interest operator. The main source of the robustness of a SIFT descriptor is in their reliance on histograms and on gradients. As a result, it is not sensitive to absolute changes in intensity and to minor shifts. A detailed analysis of its stability is given in [77]; the experiments presented below can be seen as an additional verification of these properties of SIFT.

6.4 Experiments

In our experiments we used a collection of 100 natural images taken from the Hemera database [59]. These images contain natural and man-made scenes, indoors and outdoors.

6.4.1 Collecting descriptors

Sparse overcomplete code coefficients

We used a code learned on a set of 250 natural images, not used in the descriptor evaluation. The images were whitened as described in [89], to flatten the power spectrum; this whitening appears to be critical to ensure proper convergence of the code learning algorithm. Figure 6-3 shows a few examples of the training images. The learned code consists of 1,100 basis function, for a disk-like patch of diameter 27 pixels. The total number of pixels in a patch is 529, so that the code is more than twice overcomplete. Figure 6-4 shows a representative sample of the basis function in the code. Most of the emerging basis functions correspond to localized oriented filters,[3] a typical result consistent with numerous reports in the literature.

The set of patches used in the experiments was generated by the following procedure. For each image, we selected 100 locations, subject to a minimal variance criterion: the intensity variance within a patch centered at a selected location should be at least equal to the variance within the image. For each location (r, c) we:

- Draw four random angles between 0 and 360°, and for each angle extract a patch centered at (r, c) and rotate it by each of the angles; this produces four patches.

- Extract (with no rotation) patches centered at $(c - 2, r - 2)$, $(c - 2, r + 2)$, $(c + 2, r - 2)$ and $(c + 2, r + 2)$. This produces another four patches.

This results in a total of 80,000 patches: 100 images×100 locations×8 similar patches associated with each location. For each patch we calculate the coefficients of the sparse code by applying the optimization in (6.3). The right column in Figure 6-2 shows four examples of the resulting 1,100-dimensional descriptors for a set of similar patches in Figure 6-1.

SIFT descriptors

To collect SIFT descriptors for the same patches represented by the sparse codes, we use the original (unwhitened) images. For each of the 80,000 collected patches, we define a keypoint at the location of that patch and with the scale corresponding to the diameter of the patch. We do not, however, "disclose" the rotation, and the descriptor is computed assuming the rotation is zero degrees.

The 128-dimensional descriptors were computed using the code from the Visual Geometry Group at Oxford.[4] The left column in Figure 6-2 shows the SIFT descriptors for the same patches as the codes on the right.

---

[3] Note that in general, since there is no closed-form solution for **a**, it is not possible to infer the filter corresponding to a basis function ϕ directly from ϕ, but rather one needs to estimate it on a sample of natural stimuli.

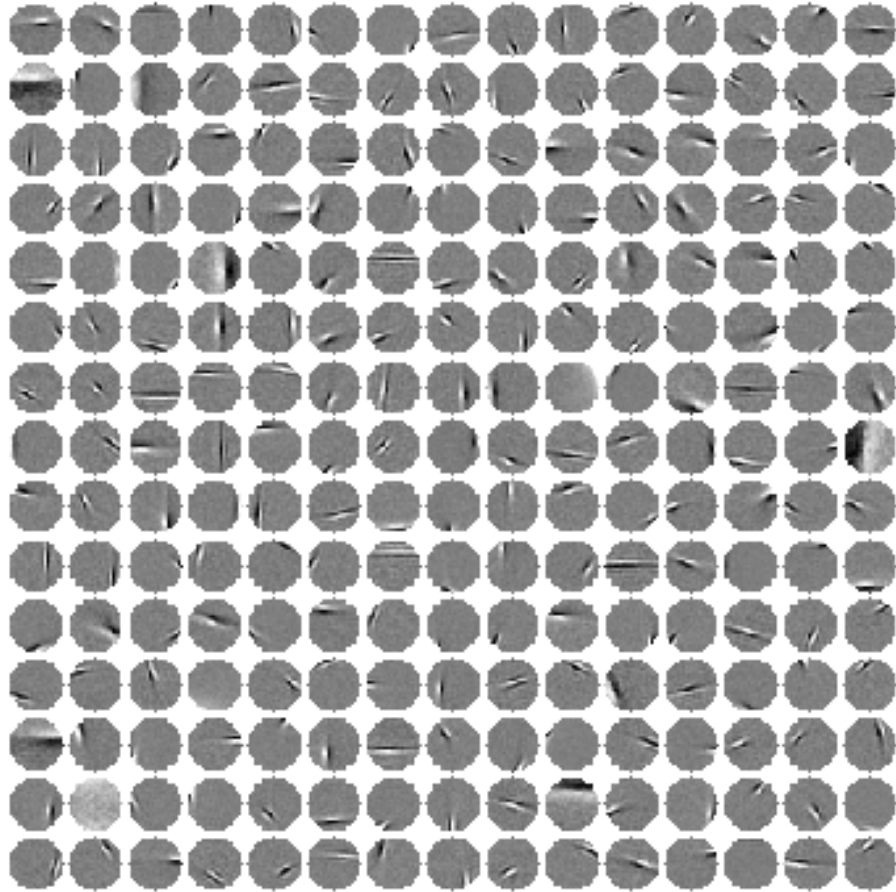[4] `http://www.robots.ox.ac.uk/~vgg/software/`

Figure 6-4: Example basis function for the sparse overcomplete code used in the experiments. 225 out of 1100 basis functions are shown. Patch size is 529 pixels (an approximate disk with diameter 27).

6.4.2 Embedding the descriptors for similarity

Under the similarity notion defined above, we can construct 280,000 similar and more than 3×10^9 dissimilar pairs out of the 80,000 patches. We randomly selected 6,000 similar and 10,000 dissimilar pairs for training, and five times as many distinct pairs for testing. All the results shown in this section were computed on the testing pairs.[5]

The embeddings for both SIFT and sparse codes were learned by running BOOST-PRO on the training pairs, using linear ten-term projections (linear combinations of 10 dimensions.)[6] Since the space of the combinations is very large, we initialized the

---

[5] The baseline similarity measures which do not involve any learning have, of course, identical performance on the training and testing data.

[6] We experimented with smaller numbers of terms, however the results, measured on an independent validation set, were significantly worse. We believe this is due to the fact that with a random pair of dimensions, if they are "useless", not much can be done by the gradient ascent, whereas with ten dimensions, there is higher likelihood that at least some are useful, and the projection coefficients are updated accordingly.

Similarity	Pixels	L_1 on $\mathbf{a}$	L_1 on $H(\mathbf{a})$	SIFT	$H(\text{SIFT})$
AUC	0.6049	0.5651	0.6847	0.8794	0.9633

Table 6.1: Area under ROC for similarity measures compared in our evaluation.
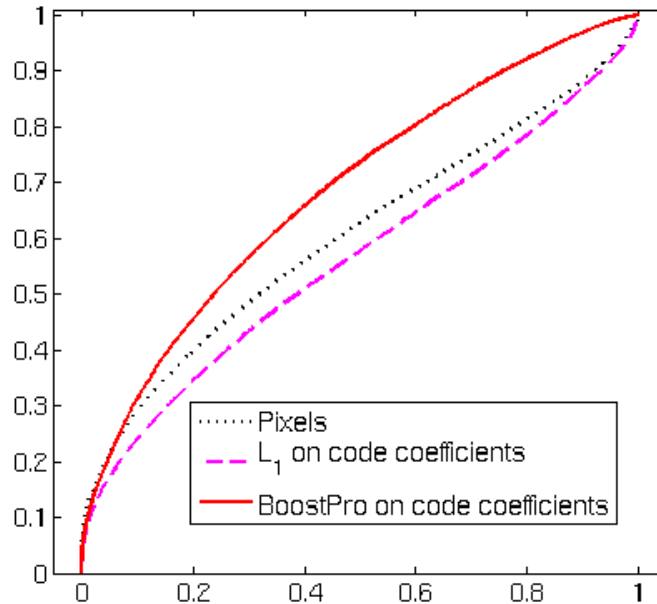

Figure 6-5: ROC curves with sparse overcomplete codes. Dotted: L_1 distance on pixels. Dashed: L_1 distance on code coefficients. Solid: L_1 in BOOSTPRO embedding.

gradient descent from 100 random projections in every iteration of boosting, and selected the best among the finishing points of the 100 gradient ascent chains. In both cases, the boosting was run for 100 iterations, thus producing one hundred embedding dimensions–a dimensionality reduction for both descriptors, particularly significant for the sparse codes! However, one should keep in mind that this is not a new representation of reduced dimension that retains all useful properties of the original one; the purpose of the embedding is explicitly to facilitate similarity detection, and, for instance, the generative power of the sparse code is lost in the 100-dimensional embedding.

We are interested in the accuracy of matching similar patches. Since the match is decided by thresholding a numerical value , namely, the distance either in the original descriptor space or in the embedding space, we can use the ROC curves to compare the models of similarity. These are shown in Figures 6-5 and 6-6, and the areas under the curves are given in Table 6.1.

A comparison between the baseline ROC curves (dashed lines in Figures 6-5 and 6-6, and the area under the curves given in Table 6.1, confirm the intuition that the
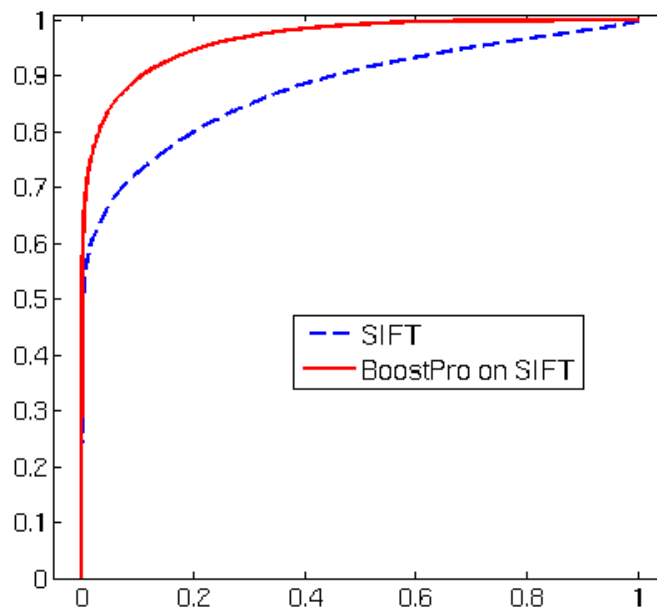
Figure 6-6: ROC curves with SIFT descriptors. Dashed: L_1 on SIFT descriptors. Solid: L_1 in BoostPro embedding.

sparse code coefficients in their "raw" form are not effective for matching patches. The performance of both pixel-based match and the L_1 distance on the sparse code coefficients is essentially not much better than random.

The ROC of the distances in the similarity embedding, on the other hand, are clearly superior to those of the distances in the original descriptor space, both for SIFT and for the sparse codes. For SIFT, the embedding yields an improvement in the area under curve to more than 0.96, with the equal error rate of 0.8930 (i.e., the probability of detecting a correct match of 89.3% corresponds to probability 10.7% of false match). By comparing the patch paitrs misclassified by the two measures we can see that most of the gain is achieved in learning the rotation correspondences between the gradient histogram bins, and as a result the error rate on patches similar up to rotation is decreased with the embedding similarity. As for the shifts, the descriptor is already quite robust to matching shifted patches. These findings are in agreement with the analysis of SIFT performance in [77].

We also have analyzed the embedding of the sparse code coefficients. Figure 6-7 shows, in each row, the basis patches which form the projection, with the coefficients written under the corresponding patches. The three rows correspond to the first three projections. Note that due to the non-convex nature of the optimization surface in BoostPro, these are not necessarily, or even likely, the most efficient projections, but rather simply the projections that happened to be picked up first. This is also reflected in the magnitude of the α_m values (the height of spikes in Figure 6-8). Often in application of AdaBoost these values decrease steadily, however this is not the case

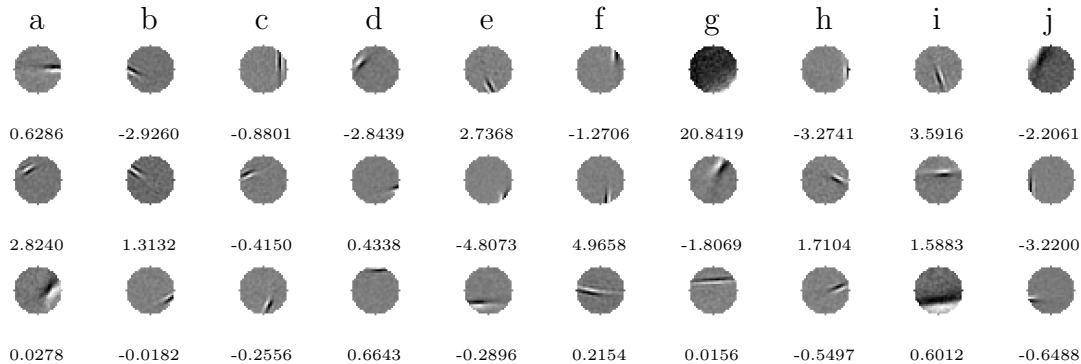| a	b	c	d	e	f	g	h	i	j
0.6286	-2.9260	-0.8801	-2.8439	2.7368	-1.2706	20.8419	-3.2741	3.5916	-2.2061
2.8240	1.3132	-0.4150	0.4338	-4.8073	4.9658	-1.8069	1.7104	1.5883	-3.2200
0.0278	-0.0182	-0.2556	0.6643	-0.2896	0.2154	0.0156	-0.5497	0.6012	-0.6488

Figure 6-7: The first three projections in the embedding of sparse code coefficients learned with BoostPro. The number under each basis patch is the coefficient in the projection. Thresholds, selected for each projection by applying Algorithm 4, are 6.85, .16 and 0.003, respectively. The letters are used to identify the columns.

here.

We will discuss in some detail the top row of the figure, showing the first projection; letters on top help identify the basis functions. Recall from Chapter 3 the general interpretation of the projections: two patches must fall on the same side of the threshold in order for the projection to contribute to the matching score (or, equivalently, for the projection *not* to contribute to the distance value in H.) Perhaps the best way to describe the effect of the individual components of the projection is to say that if for the two patches the coefficients for the basis function d have high value, and the coefficients for the function e have low value (i.e., large absolute value and negative sign), it increases their chances to be considered similar by the weak classifier associated with this projection. Note that the low frequency basis function g has a coefficient much higher than others, so that this particular projection largely looks at the spatial frequency of the features within the patch, so, for instance, two relatively "flat" patches will be likely considered similar by this projection.

Generally, the magnitude of the coefficients in the projection determines the importance of the corresponding basis function (within that projection). The signs are only meaningful relative to the signs of the other projection components. For instance, the sign of the function d is not important by itself: its contribution to similarity judgment on two patches that both have $a_d = 3$ will be the same as for two patches with $a_d = -3$ (in both cases it will "move" both patches in the same direction relative to the threshold.) However, in combination with other basis functions and their signs in the projection in the same projection the sign does play a role: if two basis functions are likely to correspond to similar patches their sign will likely match. For instance, a patch with $a_f = 1$ and a patch with $a_h = 1$ will have a positive contribution from the corresponding projection coordinates (i.e. they will be "moved" towards the same side of the threshold); this probably is explained by the possibility of a shift that move a vertical edge on the right side of the patch in the right-left direction.

Finally, Figure 6-8 shows the embeddings themselves, for the same example patches as the descriptors in Figure 6-2. This figure provides a visual illustration of the improved correspondences between the representations of similar patches (much more noticeable with SIFT.)

6.5 Discussion

The main conclusion from the experiments presented here is that it is beneficial to learn a similarity model, rather than to rely on properties of a descriptor and use the metric as a proxy for such similarity. This is the case both when the descriptor in question is not designed to be invariant to transformations, as is the case with the sparse overcomplete codes, and when it is specifically designed with such invariance in mind, which is the case with SIFT. For both descriptors, it pays off to learn a similarity model, as far as the matching quality measured by ROC is concerned. Our experiments also suggest that for matching similar image patches under rotations and minor shifts, the best method among the ones investigated is to use the SIFT descriptor of the region, embedded into a similarity-reflecting space by applying BoostPro.

Another important conclusion is that the sparse overcomplete codes for image patches can be potentially used in a matching-based framework, if a proper similarity measure (or equivalently a proper embedding of the code) is used, rather than the naïvely applied L_1 distance. While the improvement is still not enough to place this descriptor in the same "league" with SIFT, we believe that certain characteristics make it appealing and warrant further looking into its use in recognition. First, it is based on a generative model of natural images, and thus can *exlain* the patch in terms of the fundamental visual elements comprising it. Second, the performance achieved here did not require any information from an interest operator, of the kind necessary to apply SIFT. This may be an appealing property for approaches that do not use the keypoint paradigm.

An aspect of our approach to modeling visual similarity of patches that distinguishes it from other approaches, is that we do not impose an invariant representation directly. Instead we learn a representation that makes similarity under the transformations explicit, thus causing the *matching* to become invariant. Another key aspect is that similarity is *learned*, not hand-crafted into the model. This is promising from the perspective of class-specific or object-specific matching, when similarity is defined in different ways for different classes, or when the type of patches to be compared is restricted. Of course, this promise has to be evaluated in further experiments, which are in the focus of our ongoing work.

The implications of this study on "downstream" applications to recognition and classification also remain to be seen. Our goal here has been to separate the patch similarity measure from the context in which is is used by an overall recognition strategy, under the assumption that any strategy would benefit from a better matching component. Further experiments are needed to quantify the effect of the improvement in matching on the accuracy in the final task of a system. Experiments in

this direction should extend the notion of similarity (or, equivalently, invariance) to affine transformations not included in our experiments above, namely, out-of-plane rotations. However, we expect that the effect of learning the embedding will be maintained in this extended framework. Another issue that may require special care is the selection of informative patches. An almost "flat" patch may be, under most reasonable definitions of visual similarity, similar to any other flat patch however establishing such a match may not be informative from a classification or recognition perspective. More generally, some patches are more useful than others for a particular object class. This leads to the idea of class-specific visual similarity between patches. We have not yet developed an approach that would achieve this but some ideas are discussed briefly in the next chapter.
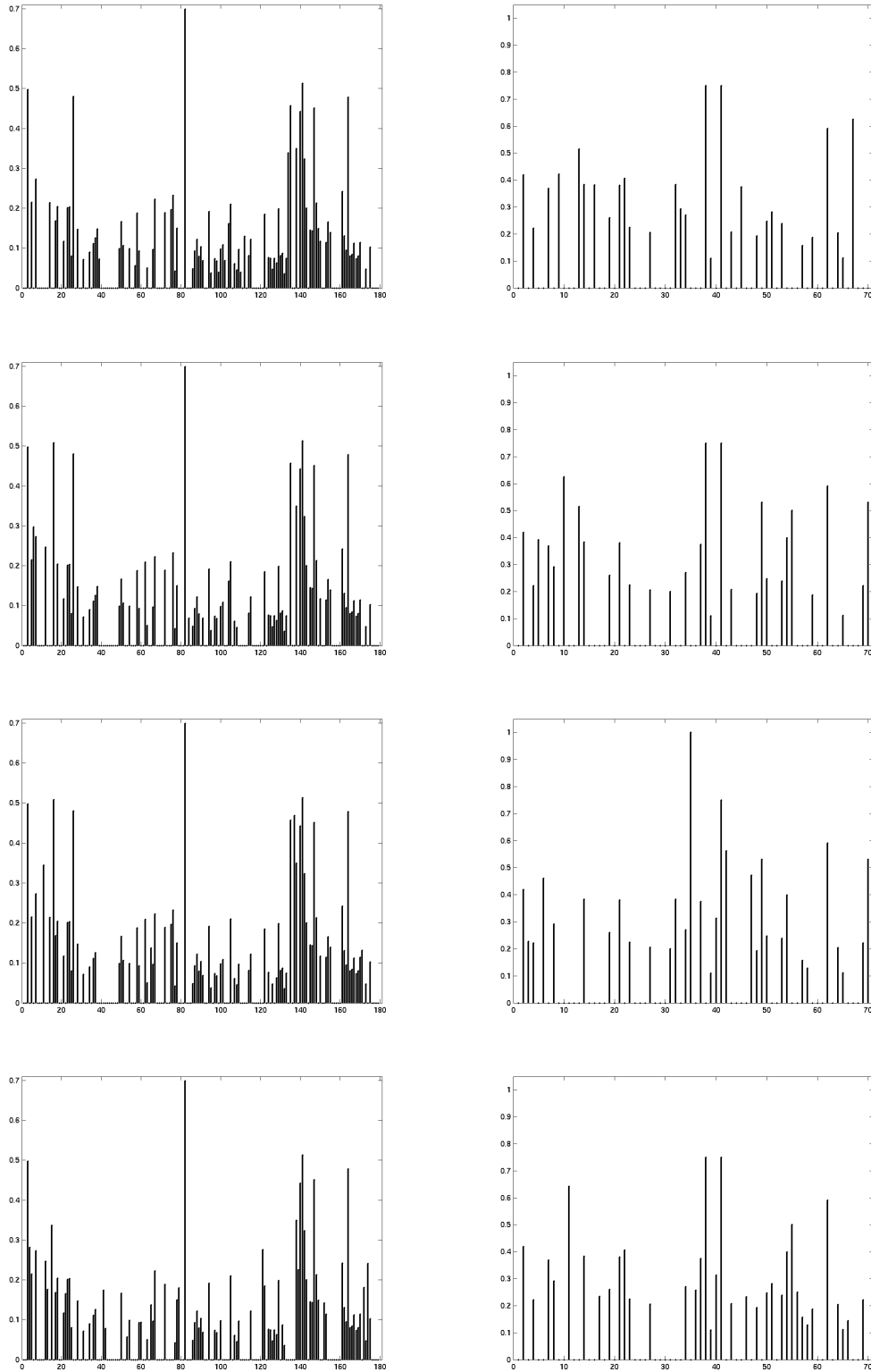
Figure 6-8: Embeddings of the codes shown in Figure 6-2. Left: BoostPro on SIFT. Right: BoostPro on sparse code coefficients.

Chapter 7

Conclusions

In this concluding chapter we summarize the contributions of this thesis and the possible impact as we see it, and discuss the important directions of future work.

7.1 Summary of thesis contributions

The central problem addressed in this thesis is the problem of modeling a boolean similarity concept, which is conveyed only be means of examples of what constitutes similar and dissimilar pairs under that concept. Before we summarize the specific technical contributions in the remainder of this section, below are the main conclusions we see emerging from our work.

- It is usually beneficial to learn a model for the similarity relevant to the task, be it regression, classification or retrieval. It rarely hurts, and usually improves the performance of the end goal application. Of course, the precise gain of learning similarity for any given application can be assessed by standard validation techniques.

- Such learning can be successfully done directly from examples of similarity judgment specific for the task, with minimal assumptions regarding the properties of the underlying similarity concept. In many cases, for instance when the task involves regression, the learning procedure including labeling similarity examples can be completed fully automatically.

- In some problems, such as pose estimation, example-based methods have been generally overlooked since it is commonly assumed they are computationally infeasible. It does not have to be the case; with suitable embedding technique it may be possible to provide a way of extremely efficient example-based estimation in complex, high-dimensional problems. Our approach, to our knowledge, is the first to combine the power of learning task-specific similarity with the general embedding framework that allows this.

7.1.1 Learning algorithms

The basis of our approach is to construct an embedding

$$H(\mathbf{x}) \;=\; [\alpha_1 h_1(\mathbf{x}), \ldots, \alpha_M h_M(\mathbf{x})],$$

such that low distance between $H(\mathbf{x})$ and $H(\mathbf{y})$ corresponds, with high probability, to positive label assigned by the similarity $\mathcal{S}(\mathbf{x}, \mathbf{y})$. The main advantage of this approach, and what distinguishes it from the alternatives known to us, is that it achieves two important goals:

- It provides us with a set of similarity classifiers on pairs of examples. This set is parametrized by the value of the threshold on distance in the embedding space H.

- It reduces the problem of similarity search to the problem of search for neighbors with respect to the L_1 distance. As a result, we are able to leverage state-of-the-art search algorithms like LSH, that have sublinear running time.

In Chapter 3 we have presented a family of learning algorithms that construct an embedding of the form described above:

Similarity Sensitive Coding (SSC) The algorithm[1] takes pairs labeled by similarity, and produces a binary embedding space H, typically of very high dimension. The embedding is learned by independently collecting thresholded projections of the data. This algorithm improves the performance of example-based methods on some data sets, and has been used however its utility is largely limited to cases when the underlying similarity is close to L_1 distance, with some modifications. This algorithm has been successful in articulated pose estimation domain, as described in Chapters 4 and 5.

Boosted SSC This algorithm[2] addresses the redundancy in SSCby collecting the embedding dimensions greedily, rather than independently. It also introduces weighting on the dimensions of H. We have applied this algorithm to the tasks of pose and orientation estimation for an articulated tracking application, described in Chapter 5.

BoostPro This algorithm is a generalization of Boosted SSCin that the dimensions of the embedding are no longer limited to axis-parallel stumps. We have introduced a continuous approximation for the thresholded projection paradigm in which a gradient ascent optimization becomes possible. This algorithm further improves the performance of example-based methods on standard benchmark data. We also show its performance on articulated pose estimation, in chapter 4. Finally, we have used this algorithm to learn visual similarity of image patches, and have shown significant improvement over standard similarity measures used with two patch descriptors.

---

[1]Published in [105]; joint work with P. Viola.
[2]Published in [93]; joint work with L. Ren, J. Hodgins, H. Pfister and P. Viola.

Semi-supervised learning For each of these three algorithms we have presented a semi-supervised version which only requires pairs similar under $\mathcal{S}$, in addition to a set of unlabeled individual examples in $\mathcal{X}$.

7.1.2 Example-based pose estimation

In chapters 4 and 5 we have introduced a new approach to pose estimation from single image. Contrary to previously proposed approaches, it does not use a parametric model that is to be fitted to the image. Instead, it uses the learned similarity embedding to search a large database of images with known underlying poses. As a result, the notoriously difficult problem of fitting the articulated pose model is reduced to two much simpler, and much faster, steps: search in a database for (approximately) nearest neighbors, and fitting a local low-order model to the retrieved neighbors. To our knowledge this approach achieves state-of-the-art performance while requiring significantly less time per image than alternative approaches.

7.1.3 Articulated tracking

The main impact of our approach on articulate tracking is in providing a way of automatic initialization of the tracker and, effectively, subsequent re-initialization in every frame. In Chapter 5 we have described two tracking systems that take advantage of this ability. Both systems have been demonstrated to be superior, in terms of combined speed, accuracy and robustness, to state-of-the-art alternatives.

7.1.4 Patch similarity

In Chapter 6 we have described another application of the similarity learning framework: learning visual similarity of natural image patches under rotation and small translation. For two patch descriptors (the sparse overcomplete code coefficients and the very popular SIFT descriptor) we have shown that by learning an embedding of the descriptor with BOOSTPRO and using the distance in the embedding space, we can significantly improve the matching accuracy. The main contributions of this study are:

- This is the first attempt, to our knowledge, to improve the matching accuracy of standard (and widely used) descriptors by learning a similarity model specific to the invariant properties the matching is intended to capture.

- The fact that the learned similarity is measured by the L_1 distance in the embedding space is very significant from the computational point of view, since in a large-scale recognition system we may need to probe databases with millions of patches for similarity to the input set of patches. Our framework allows us to apply algorithms like LSH, and perform this search in sublinear time.

7.2 Direction for future work

Theoretical investigation An open theoretical question that arises from the work presented here pertains to the class of similarity concepts that can be attained by the embedding algorithms presented in Chapter 3. By departing from the framework of LSH to similarity-sensitive framework introduced in Section 2.4.2, we extend the class of similarities from L_1 to a more general family. It would be interesting to characterize the properties of this family, and the connections between the geometry of a similarity concept in $\mathcal{X}^2$ and the extent to which an embedding learned by our algorithms can represent that concept.

Evaluation We believe that a number of interesting additional experiments would be useful to better understand the differences between algorithms and the conditions under which each algorithm is best applicable. Such experiments include an evaluation of boosted SSC on more tasks, in addition to the pose estimation task in Chapter 5, to better understand its capacity and limitations and an investigation into better ways of setting the bound G on the TP-FP gap in SSC. In addition, we are investigating improved strategies of selecting the projection terms (i.e. the dimensions used in a projection) in BoostPro, especially for high-dimensional representation where even approximating the exhaustive search of the space of fixed-size term combinations is impractical.

Another aspect of empirical evaluation that should be improved is in the area of comparing pose estimation algorithms. Although lack of a standard articulated pose benchmark with known ground truth (neither real images nor realistic synthetic ones) makes this difficult, it is important to compare alternative approaches; one approach we are aware of which may provide a suitable alternative has been recently proposed in [2].

Extending the similarity framework In the Introduction we mention definitions of similarity that are more refined than the boolean notion addressed in this thesis. The algorithms presented here are developed to deal with the boolean case, however we believe they can be extended to learning ranking. The main change in the formulation is the transition from the classification of pairs to the classification of triples. Recent work [5] suggests that an embedding can be learned that represents ranking under known distance functions. We believe that it may be possible to extend such an approach to the case when the ground truth ranking is conveyed only by examples, in a spirit similar to our extension of the LSH. One important application of such extension would be in information retrieval, where feedback often is available in the form of ranking rather than just binary labels on the results.

Learning features for visual classification The results presented in Chapter 6 suggest a promising direction of future research in the use of learned similarity. It would be interesting to investigate the effect of embedding the descriptors (and the improved matching accuracy) on classification performance. Below we present an

idea for integration of the similarity learning approach developed in this thesis in a multi-category classification architecture.

Evidence from neuroscience [39] suggests that the majority of cells in the visual pathway may be placed within a computational hierarchy. As the level in the hierarchy increases, which roughly corresponds to retino-cortical direction (away from the retina),

- The invariance increases: features become less sensitive to various transformations.

- Selectivity increases: it takes a more distinctive image elements to activate a feature. Consequently, higher layers should be more overcomplete and sparse.

- Receptive fields become larger.

- Receptive fields become more complex (more non-linear, in particular).

From a computational point of view, the order in the hierarchy corresponds to order of processing: the lowest level corresponds to measures computed directly from the image pixels, and the values in subsequent layers may be computed from the values obtained in the lower levels. However, it is not clear how the flow of sensory information and decisions across the hierarchy is organized in the brain; in particular, there exists a huge number of feedback projections along the visual pathway, the function of which is not fully understood.

It would be interesting to explore a hierarchical representation organized in accordance with the computational principles mentioned above. Finding the learning algorithm for constructing such a hierarchy is the main challenge in designing such an architecture. An interesting approach could be to learn the lower, less selective layers in an unsupervised way, while the higher, more selective layers could be better learned on a per-category basis, perhaps in conjunction with learning object- or part-specific similarity operators, along the lines developed in this thesis.

Figure 7-1 shows a "cartoon" of this approach. An appealing property of it is that lower-level features are necessarily shared between all categories, while higher-level features are more likely to be unique for a given class (although the learning algorithm should probably allow for sharing in later layers as well).

It's important to emphasize the difference between this approach and, say, the standard multi-layer neural network where a designated output layer is the only one affecting the decision. In the proposed hierarchy there is no output layer per se, but rather the entire set of features is considered in similarity calculations. This is achieved by allowing any feature to be used in similarity-reflecting embeddings for the highest (categorical) level.
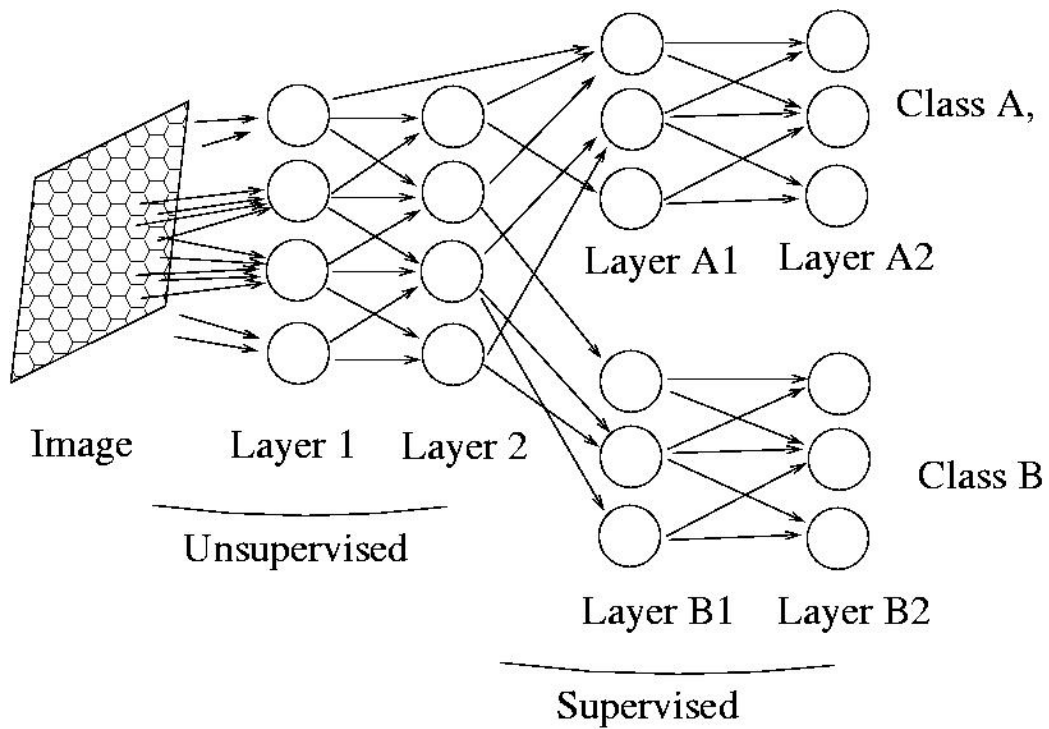
Figure 7-1: A cartoon of the proposed hierarchical representation, showing the sharing of the features and the two-stage learning architecture. A representation for a given image patch may include any of the features from the lower (generic) layers and any of the features from the higher, class-specific layers.

Bibliography

[1] Delve Datasets. http://www.cs.toronto.edu/ delve/data/datasets.html.

[2] Ankur Agarwal and Bill Triggs. 3d human pose from silhouettes by relevance vector regression. In *International Conference on Computer Vision & Pattern Recognition*, pages II 882–888, Washington, DC, June 2004.

[3] S. Agarwal and D. Roth. Learning a Sparse Representation for Object Detection. In *European Conference on Computer Vision*, 2004.

[4] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[5] V. Athitsos, J. Alon, S. Sclaroff, and G. Kollios. Boostmap: A method for efficient approximate similarity rankings. In *IEEE Conf. on Computer Vision and Pattern Recognition*, Madison, WI, June 2004.

[6] V. Athitsos and S. Sclaroff. Estimating 3D Hand Pose from a Cluttered Image. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 432–439, Madison, WI, June 2003.

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

[8] D. H. Ballard and L. E. Wixson. Object Recognition Using Steerable Filters at Multiple Scales. In *Proceedings of IEEE Workshop on Qualitative Vision*, 1993.

[9] Evgeniy Bart and Shimon Ullman. Class-based matching of object parts. In *Proceedings of CVPR Workshop on Image and Video Registration*, 2004.

[10] J. S. Beis and D. G. Lowe. Shape Indexing Using Approximate Nearest-Neighbor Search in High-Dimensional Spaces. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 1000–1006, 1997.

[11] S Belongie, J. Malik, and J. Puzicha. Shape Matching and Object Recognition Using Shape Contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, April 2002.

[12] Y. Bengio, J.-F. Paiement, P. Vincent, O. Delalleau, N. Le Roux, and M. Ouimet. Out-of-sample extensions for lle, isomap, mds, eigenmaps, and spectral clustering. In *Neural Information Processing Systems*, 2004.

[13] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[14] A. C. Berg, T. L. Berg, and J. Malik. Shape Matching and Object Recognition using Low Distortion Correspondence. In *IEEE Conf. on Computer Vision and Pattern Recognition*, June 2005.

[15] P.J. Besl and N. MacKay. A Method for Registration of 3D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14:239–256, February 1992.

[16] C. L. Blake and C. J. Merz. UCI repository of machine learning databases. [http://www.ics.uci.edu/∼mlearn/MLRepository.html], 1998.

[17] E. Borenstein and S. Ullman. Class-Specific, Top-Down Segmentation. In *European Conference on Computer Vision*, pages 109–124, Copenhagen, Denmark, May 2002.

[18] A. Borodin, R. Ostrovsky, and Y. Rabani. Lower bounds for high dimensional nearest neighbor search and related problems. In *Proceedings of the Thirty-First Annual ACM Symposium on Theory of Computing (STOC'99)*, pages 312–321, New York, May 1999. Association for Computing Machinery.

[19] Michael C. Burl, Markus Weber, and Pietro Perona. A probabilistic approach to object recognition using local photometry and global geometry. In *European Conference on Computer Vision*, Freiburg, Germany, 1998.

[20] Sumit Chopra, Raia Hadsell, and Yann LeCun. Learning a similarity metric discriminatively, with application to face verificatio. In *IEEE Conf. on Computer Vision and Pattern Recognition*. IEEE Press, 2005.

[21] W. S. Cleveland. Robust locally weighted regression and smoothing scatter plots. *Journal of American Statistical Association*, 74(368):829–836, 1979.

[22] W. S. Cleveland and S. J. Delvin. Locally weighted regression: an approach to regression analysis by local fitting. *Journal of American Statistical Association*, 83(403):596–610, 1988.

[23] M. Collins, R. Schapire, and Y. Singer. Logistic regression, adaboost and bregman distances. In *COLT: Proceedings of the Workshop on Computational Learning Theory, Morgan Kaufmann Publishers*, 2000.

[24] T. M. Cover. Estimation by the nearest neighbor rule. *IEEE Transactions on Information Theory*, 14:21–27, January 1968.

[25] T. M. Cover. Rates of Convergence for Nearest Neighbor Procedures. In *Proc. 1st Ann. Hawaii Conf. Systems Theory*, pages 413–415, January 1968.

[26] T. M. Cover and P. E. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13:21–27, January 1967.

[27] T. F. Cox and M. A. A. Cox. *Multidimensional scaling.* Chapman & Hall, London, 1994.

[28] K. Crammer and Y. Singer. On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. *Journal of Machine Learning Research*, 2:265–292, December 2001.

[29] Curious Labs, Inc., Santa Cruz, CA. *Poser 5 - Reference Manual*, 2002.

[30] D. Michie, D.J. Spiegelhalter, and C.C. Taylor, editors. *Machine Learning, Neural and Statistical Classification.* Ellis Horwood, 1994.

[31] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *annual acm symposium on computational geometry*, 2004.

[32] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, Heidelberg, New York, second edition, 2000.

[33] V. de Silva and J. B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In *Neural Information Processing Systems*, pages 705–712, 2002.

[34] D. Demirdjian and T. Darrell. Using Multiple-Hypothesis Disparity Maps and Image Velocity for 3-D Motion Estimation. *International Journal of Computer Vision*, 47(1-3), 2002.

[35] D. Demirdjian, L. Taycher, G. Shakhnarovich, K. Grauman, and T. Darrell. Avoiding the streetlight effect: Tracking by exploring likelihood modes. In *Proceedings of the International Conference on Computer Vision*, Beijing, PRC, October 2005 (to appear).

[36] J. Deutscher, A. Blake, , and I. Reid. Articulated body motion capture by annealed particle filtering. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 126–133, Hilton Head, USA, June 2000.

[37] L. Devroye. On the inequality of Cover and Hart in nearest neighbor discrimination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3:75–78, 1981.

[38] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern classification.* John Wiley & sons, New York, second edition, 2001.

[39] E. R. Kandel, J. H. Schwartz, and T. M. Jessell. *Principles of Neural Science.* Elsevier, New York, 3rd edition, 1991.

[40] A. Elgammal and C. S. Lee. Inferring 3D Body Pose from Silhouettes using Activity Manifold Learning. In *IEEE Conf. on Computer Vision and Pattern Recognition*, Washington, DC, July 2004.

[41] {Eyes, JAPAN}. Motion capture sequences database. www.mocapdata.com, 2005.

[42] C. Faloutsos and K.-I. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In Michael J. Carey and Donovan A. Schneider, editors, *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data*, pages 163–174, San Jose, California, 22–25 1995.

[43] J. Fan and I. Gijbels. *Local Polynomial Modelling and Its Applications.* Chapman and Hall, 1996.

[44] M. Farach-Colton and P. Indyk. Approximate nearest neighbor algorithms for Hausdorff metrics via embeddings. In *40th annual symposium on foundations of computer science*, pages 171–179, New York, NY, 17 October 1999. IEEE Computer Society Press.

[45] P. Felzenszwalb and D. Huttenlocher. Efficient matching of pictorial structures. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 66–75, Los Alamitos, June 13–15 2000. IEEE.

[46] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, volume 2, pages 264–271, June 2003.

[47] D. Field. What is the goal of sensory coding? *Neural Computation*, 6:559–601, 1994.

[48] J. Fritz. Distribution-Free Exponential Error Bound for Nearest Neighbor Pattern Classification. *IEEE Transactions on Information Theory*, 21(5):552–557, September 1975.

[49] A. Frome, D. Huber, R. Kolluri, T. Bulow, and J. Malik. Recognizing objects in range data using regional point descriptors. In *European Conference on Computer Vision*, Prague, Czech Republic, May 2004.

[50] K. Fukunaga and D. M. Hummels. Bias of nearest neighbor error estimates. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(1):103–112, January 1987.

[51] B. Georgescu, I. Shimshoni, and P. Meer. Mean shift based clustering in high dimnensions: A texture classification example. In *International Conference on Computer Vision*, 2003. (to appear).

[52] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99)*, pages 518–529, San Francisco, September 1999. Morgan Kaufmann.

[53] J. Goldberger, S. T. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood component analysis. In *Neural Information Processing Systems*, pages 513–520, 2004.

[54] R. Gonzalez and R. Woods. *Digital image processing.* Prentice hall, Upper Saddle River, New Jersey, 2nd edition, 2001.

[55] K. Grauman and T. Darrell. Fast contour matching using approximate earth mover's distance. In *IEEE Conf. on Computer Vision and Pattern Recognition*, Washington, DC, June 2004.

[56] K. Grauman and T. Darrell. The pyramid match kernel: Discriminative classification with sets of image features. In *International Conference on Computer Vision*, Beijing, PRC, October 2005.

[57] K. Grauman, G. Shakhnarovich, and T. Darrell. A bayesian approach to image-based visuall hull reconstruction. In *Proceedings IEEE Conf. on Computer Vision and Pattern Recognition*, Madison, WI, 2003.

[58] K. Grauman, G. Shakhnarovich, and T. Darrell. Inferring 3d structure with a statistical image-based shape model. In *Proceedings of the International Conference on Computer Vision*, Nice, France, October 2003.

[59] Hemera Images. www.hemera.com.

[60] T. Hertz, A. Bar-Hillel, and D. Weinshall. Boosting margin-based distance functions for clustering. In *International Conference on Machine Learning*, Banff, Canada, July 2004.

[61] T. Hertz, A. Bar-Hillel, and D. Weinshall. Learning distance functions for image retrieval. In *IEEE Conf. on Computer Vision and Pattern Recognition*, Washington, DC, June 2004.

[62] G. R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(5):530–549, May 2003.

[63] M. K. Hu. Visual Pattern Recognition from Motion Invariants. 8:179–187, 1962.

[64] A. Hyvrinen and P. O. Hoyer. A two-layer sparse coding model learns simple and complex cell receptive fields and topography from natural images. *Vision Research*, 41(18):2413–2423, 2001.

[65] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing (STOC-98)*, pages 604–613, New York, May 23–26 1998. ACM Press.

[66] M. Isard and A. Blake. Condensation – conditional density propagation for visual tracking. *International Journal of Computer Vision*, 29(1):5–28, 1998.

[67] R. A. Jacobs, M. I. Jordan, S. J. Nowlan, and G. E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

[68] A. K. Jain and A. Vailaya. Image retrieval using color and shape. *Pattern Recognition*, 8(29):1233–1244, 1996.

[69] M. Jones and P. Viola. Face Recognition Using Boosted Local Features. Technical Report TR2003-025, MERL, Cambridge, MA, May 2003.

[70] Y. Ke, D. Hoiem, and R. Sukthankar. Computer vision for music identification. In *IEEE Conf. on Computer Vision and Pattern Recognition*, San Diego, CA, June 2005 (to appear).

[71] Svetlana Lazebnik, Cordelia Schmid, and Jean Ponce. Semi-local affine parts for object recognition. In *British Machine Vision Conference*, volume volume 2, pages 779–788, 2004.

[72] J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive Control of Avatars Animated with Human Motion Data. *ACM Transactions on Graphics: Special Issue Proceedings of SIGGRAPH*, 21(3):491–500, 2002.

[73] H. Lei and V. Govindaraju. Speeding Up Multi-class SVM by PCA and Feature Selection. In *Feature Selection in Data Mining, Workshop*, pages 72–79, 2005.

[74] B. Leibe and B. Schiele. Analyzing contour and appearance based methods for object categorization. In *IEEE Conf. on Computer Vision and Pattern Recognition*, Madison, WI, 2003.

[75] T. Liu, A. W. Moore, and A. Gray. New Algorithms for Efficient High Dimensional Non-parametric Classification. In *Neural Information Processing Systems*, 2003.

[76] D. G. Lowe. Object recognition from local scale-invariant features. pages 1150–1157, Corfu, Greece, December 2000.

[77] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[78] David G. Lowe. Similarity metric learning for a variable-kernel classifier. Technical report, February 15 1994.

[79] S. Mahamud and M. Hebert. The optimal distance measure for object detection. In *IEEE Conf. on Computer Vision and Pattern Recognition*, Madison, WI, June 2003.

[80] Raphaël Marée, Pierre Geurts, Justus Piater, and Louis Wehenkel. Random subwindows for robust image classification. In *IEEE Conf. on Computer Vision and Pattern Recognition*, volume 1, pages 34–40, June 2005.

[81] J. Matas, O. Chum, U. Martin, and T. Pajdla. Robust wide baseline stereo from maximally stable extremal regions. In *British Machine Vision Conference*, pages 384–393, London, UK, September 2002.

[82] Bartlett W. Mel. SEEMORE: Combining color, shape, and texture histogramming in a neurally inspired approach to visual object recognition. *Neural Computation*, 9(4):777–804, 1997.

[83] David Meyer, Friedrich Leisch, and Kurt Hornik. The support vector machine under test. *Neurocomputing*, 55:169–186, September 2003.

[84] Krystian Mikolajczyk, Tinne Tuytelaars, Cordelia Schmid, Andrew Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. Van Gool. A comparison of affine region detectors. *Submitted to International Journal of Computer Vision*, 2004. Submitted in August 2004.

[85] Baback Moghaddam, Tony Jebara, and Alex Pentland. Bayesian face recognition. *Pattern Recognition*, 33:1771–1782, 2000.

[86] G. Mori, S. Belongie, and J. Malik. Shape contexts enable efficient retrieval of similar shapes. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 723–730, Lihue, HI, 2001.

[87] G. Mori and J. Malik. Estimating Human Body Configurations using Shape Context Matching. In *European Conference on Computer Vision*, pages 666–680, 2002.

[88] G. Mori, X. Ren, A.A. Efros, and J. Malik. Recovering human body configurations: Combining segmentation and recognition. In *IEEE Conf. on Computer Vision and Pattern Recognition*, volume 2, pages 326–333, 2004.

[89] B. A. Olshausen and D. J. Field. Sparse coding with an overcomplete basis set: A strategy employed in V1. *Vision Research*, 37:3311–3325, 1997.

[90] D. Psaltis, R. R. Snapp, and S. S. Venkatesh. On the Finite Sample Performance of the Nearest Neighbor Classifier. *IEEE Transactions on Information Theory*, 40(3):820–837, May 1994.

[91] A. Quattoni, M. Collins, and T. Darrell. Conditional Random Fields for Object Recognition. In *Neural Information Processing Systems*, 2004.

[92] R. P. Rao and D. Ballard. Object indexing using an iconic sparse distributed memory. In *Intl. Conf. on Computer Vision*, pages 24–31, 1995.

[93] L. Ren, G. Sharknarovich, J. Hodgins, H. Pfister, and P. Viola. Learning silhouette features for control of human motion. *ACM Transactions on Graphics*, 2005 (to appear).

[94] R. Ronfard, C. Schmid, and B. Triggs. Learning to parse pictures of people. In *European Conference on Computer Vision*, Copenhagen, Denmark, 2002.

[95] R. Rosales and S. Sclaroff. Specialized mappings and the estimation of body pose from a single image. In *IEEE Human Motion Workshop*, pages 19–24, Austin, TX, 2000.

[96] S. T. Roweis and L. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, December 2000.

[97] S. Mahamud. *Discriminative Distance Measures for Object Detection*. PhD thesis, Carnegie-Mellon University, 2002.

[98] P. Sallee and B. Olshausen. Learning sparse multiscale image representations. In *Neural Information Processing Systems*, 2003.

[99] R. E. Schapire. A brief introduction to boosting. In *IJCAI*, pages 1401–1406, 1999.

[100] R. E. Schapire and Y. Singer. Improved boosting algorithms using confidence-rated predictions. *Machine Learning*, 37(3):297–336, 1999.

[101] B. Schiele and J. L. Crowley. Recognition without Correspondence using Multidimensional Receptive Field Histograms. *ijcv*, 36(1):31–50, January 2000.

[102] C. Schmid, R. Mohr, and C. Bauckhage. Evaluation of interest point detectors. *International Journal of Computer Vision*, 37(2):151–172, 2000.

[103] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, 2002.

[104] Thomas B. Sebastian, Philip N. Klein, and Benjamin B. Kimia. Recognition of Shapes by Editing Shock Graphs. In *International Conference on Computer Vision*, pages 755–762, Vancouver, BC, July 2001. IEEE Computer Society.

[105] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter sensitive hashing. In *Proceedings of the International Conference on Computer Vision*, Nice, France, October 2003.

[106] C.A Shipp and L. I. Kuncheva. An investigation into how AdaBoost affects classifier diversity. In *Proc. IPMU*, pages 203–208, 2002.

[107] J. Sivic, B. C. Russell, A. A. Efros, A. Zisserman, and W. T. Freeman. Discovering objects and their location in images. In *International Conference on Computer Vision*, 2005.

[108] C. Sminchisescu and B. Triggs. Estimating Articulated Human Motion with Covariance Scaled Sampling. *International Journal of Robotics Research*, 22(6):371–393, 2003.

[109] R. R. Snapp and S. S. Venkatesh. Asymptotic derivation of the finite-sample risk of the k nearest neighbor classifier. Technical Report UVM-CS-1998-0101, University of Vermont, Burlingotn, Burlington, VT, October 1997.

[110] B. Stenger, P. R. S. Mendonca, and R. Cipolla. Model-Based 3D Tracking of an Articulated Hand. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 310–317, Lihue, HI, December 2001.

[111] C. J. Taylor. Reconstruction of articulated objects from point correspondences in a single uncalibrated image. *Computer Vision and Image Understanding*, 80(3):349–363, December 2000.

[112] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, December 2000.

[113] Luís Torgo. Error estimators for pruning regression trees. pages 125–130, Chemnitz, Germany, April 1998.

[114] P. Viola and M. Jones. Fast and robust classification using asymmetric adaboost and a detector cascade. In T. G. Dietterich, S. Becker, and Z. Ghahramani, editors, *Advances in Neural Information Processing Systems 14*, Cambridge, MA, 2002. MIT Press.

[115] Paul Viola and Michael Jones. Robust real-time face detection. In *International Conference on Computer Vision*, pages 747–747, Los Alamitos, CA, July 2001. IEEE Computer Society.

[116] Laurenz Wiskott and Terrence Sejnowski. Slow feature analysis: Unsupervised learning of invariances. *Neural Computation*, 14(4):715–770, 2002.

[117] Jianxin Wu, James M. Rehg, and Matthew D. Mullin. Learning a rare event detection cascade by direct feature selection. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems 16*, Cambridge, MA, 2004. MIT Press.

[118] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell. Distance metric learning with application to clustering with side-information. In S. Thrun S. Becker and K. Obermayer, editors, *Advances in Neural Information Processing Systems 15*, pages 505–512. MIT Press, Cambridge, MA, 2003.