Chapter 3

Learning embeddings that reflect similarity

This chapter describes a family of algorithms for learning an embedding

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

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 $n \leq 2N + 1$ 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.

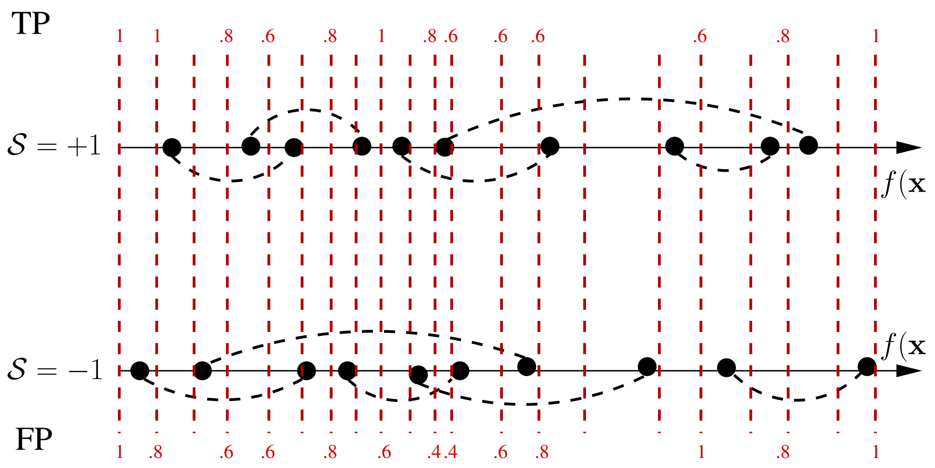


Figure 3-1: Illustration of the operation of Algorithm 4. Similar (top) and dissimilar (bottom) pairs are connected by dashed lines, and are assumed to all have equal weights of 1/10. All 21 distinct thresholds are shown; the TP (top) and FP (bottom) rates are shown only for some. The maximal attainable 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 $\{\langle T_t, \mathrm{TP}_t, \mathrm{FP}_t \rangle\}_{t=1}^n$, where $\mathrm{TP}_t$ and $\mathrm{FP}_t$ 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 $d_{i,1} := \begin{cases} +1 & \text{if } v_{i,1} \le v_{i,2}, \\ -1 & \text{if } v_{i,1} > v_{i,2}. \end{cases}$
 7:     Let $d_{i,2} := \begin{cases} +1 & \text{if } v_{i,1} > v_{i,2}, \\ -1 & \text{if } v_{i,1} \le v_{i,2}. \end{cases}$
 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 $i_j := \max\{i :\ v_i \le T_j\}$
12:     $\mathrm{TP}_j\ :=\ 1 - \sum_{i \le i_j, l_i=+1} w_i d_i$.
13:     $\mathrm{FP}_j\ :=\ 1 - \sum_{i \le i_j, l_i=-1} w_i d_i$.
```

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 $H^{\text{SSC}} : \mathcal{X} \to \{0,1\}^M$ ($M$ 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 $\{\text{TP}_t^d\}, \{\text{FP}_t^d\}$.
6:     for all $t = 1, \ldots, n$ do
7:         if $\text{TP}_t^d - \text{FP}_t^d \geq G$ 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 $H^{\text{SSC}} \triangleq \mathbf{x} \to [h_1(\mathbf{x}), \ldots, h_M(\mathbf{x})]$
```

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,. . . 26	0
Isolet	[16]	617	3899	Classification	1,. . . 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.

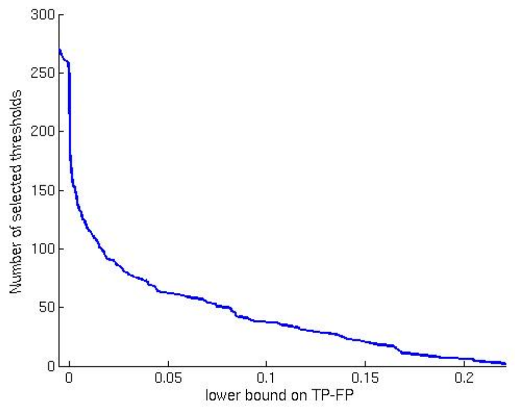
As mentioned in Chapter 1, another measure of the performance of a similarity model is its direct effect on retrieval accuracy. Figures 3-6-3-12 show the plots of the ROC curves for L_1 and SSC on the seven benchmark datasets. In six out of seven datasets, the curves for SSC (blue, dashed) are clearly 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

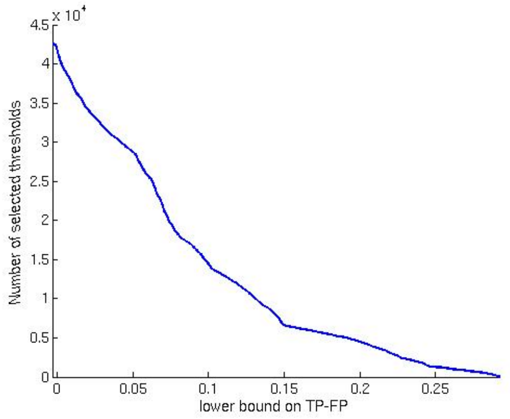
An immediate effect of the value of G is on the value M, the number of selected bits. Setting G too high will result in failure to construct an embedding; setting it too high will result in an embedding with a huge number of bits, not only not efficient but also

[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.

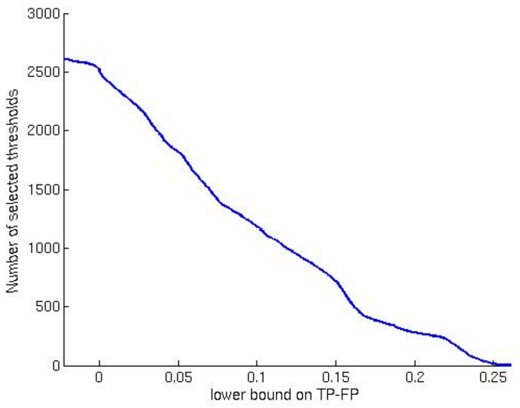
impractical due to the required storage space. Figure 3-2 shows, for four datasets, how the number of accepted thresholds (pooled over all dimensions) declines as the lower bound on the gap increases.



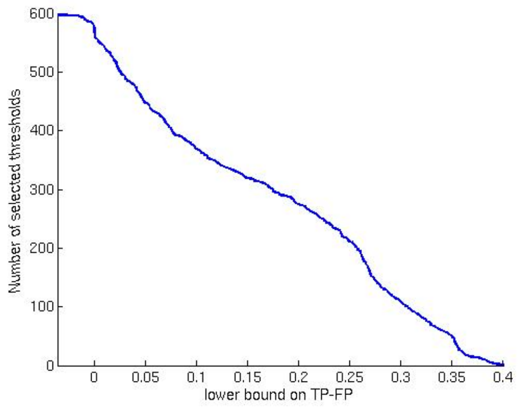
(a) Letter, dimension 4



(b) CPU, dimension 11



(c) Boston housing, dimension 10

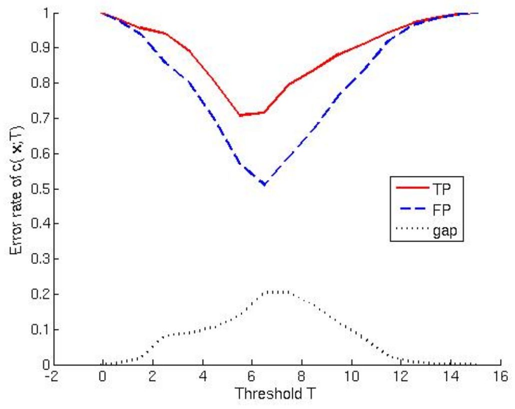


(d) Auto-MPG, dimension 3

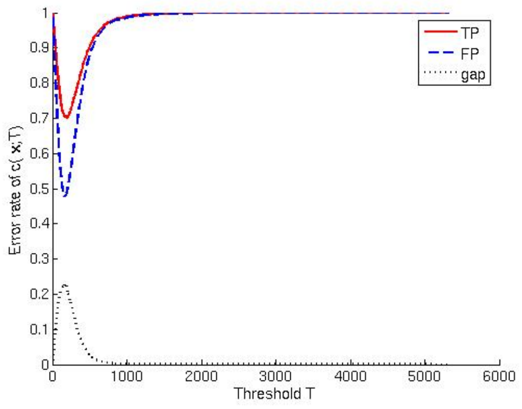
Figure 3-2: The distribution of TP-FP gap values for four data sets (pooled over all dimensions.)

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.

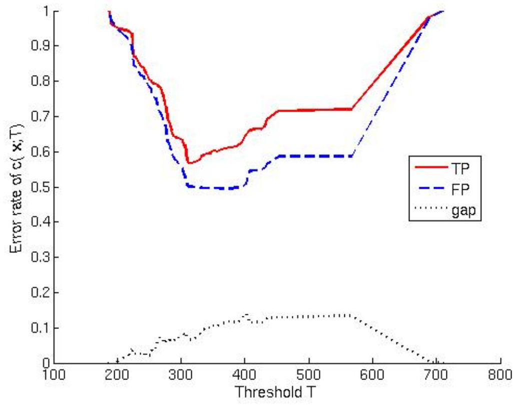
One other observation from Figure 3-3 is that the false positive rates appear to be bounded from below at around 1/2. We will discuss this phenomenon and its implications in Section 3.2.4.



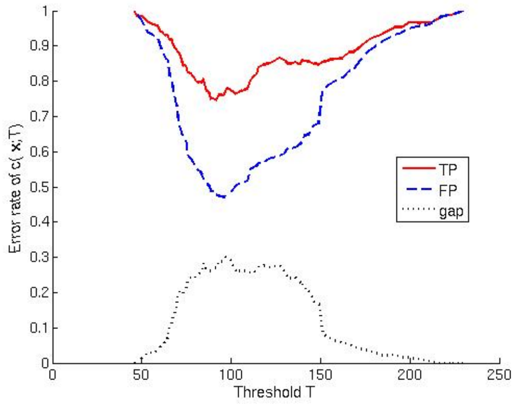
(a) Letter, dimension 4



(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.

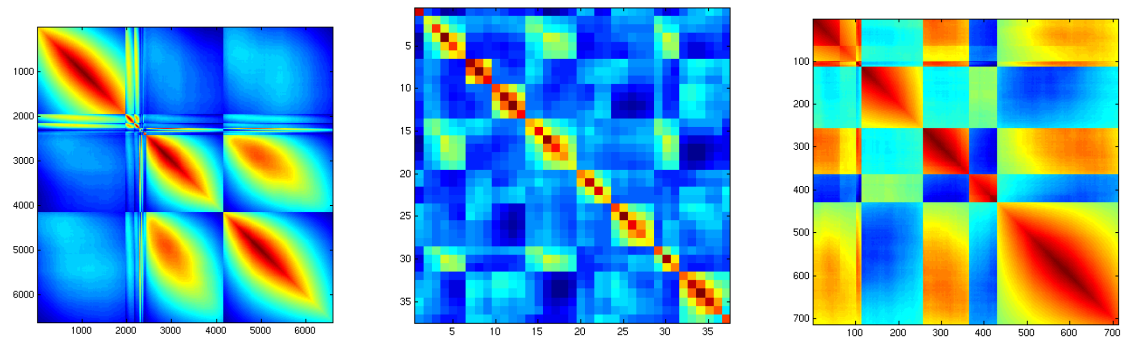
tures has been discussed in [97], in the context of binary classification problems.) It is interesting to compare M to the length of the original representation. In terms of the "nominal" number of dimensions, M is typically higher(as evident in Table 3.2) than $\dim(\mathcal{X})$. However, the effective 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

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

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:

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

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 $P^+ = \{\left(\mathbf{x}_i^{(1)}, \mathbf{x}_i^{(2)}\right)\}_{i=1}^{N_p}\} \subset \mathcal{X}^2$.
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 $\mathrm{TP}_i$ and $\mathrm{FP}_i$ 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 Algo-
       rithm 4, but using only similar pairs in $P^+$
5:     for all $j = 1, \ldots, n$ do
6:         Let $i_j := \max\{i : \, v_i \leq T_j\}$
7:         $\mathrm{TP}_j \; := \; 1 - \sum_{i \leq i_j} w_i d_i$.
8:         Let $\pi_j = \sum_{i:\, f(\mathbf{x}_i) \leq T_j} s_i$.
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

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

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

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

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

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

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 $\mathrm{TP}_d^j$ and $\mathrm{FP}_d^j$
        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 $\alpha_m$ to minimize $Z_m(\alpha)$ (see text).
10:   If $\alpha_m \leq 0$, 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

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 $\pi_m = \Pr(f(\mathbf{x}) \leq T)$. 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

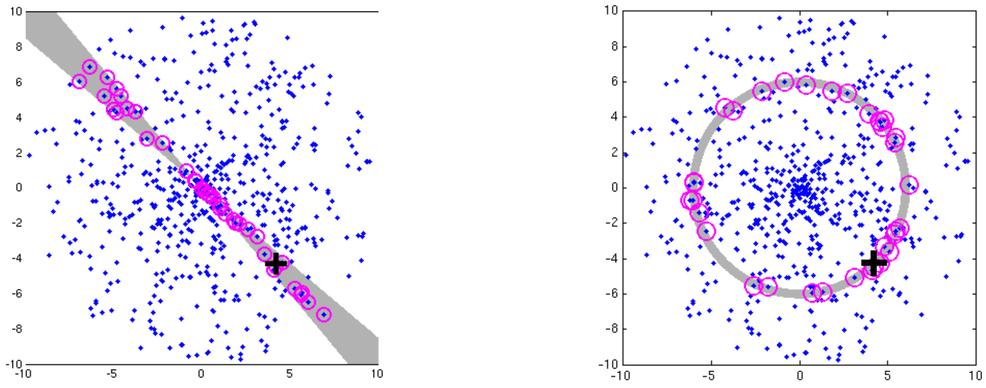

(a) Angle similarity $\mathcal{S}_A$ (b) Norm similarity $\mathcal{S}_N$

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 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, $q = 1, \dots, D$ 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 ± 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 ±	0.1713 ± 0.0215	0.0993 ± 0.0237

Table 3.4: Test accuracy of K-NN classification with SSC vs L_1 similarity (mean ± 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.

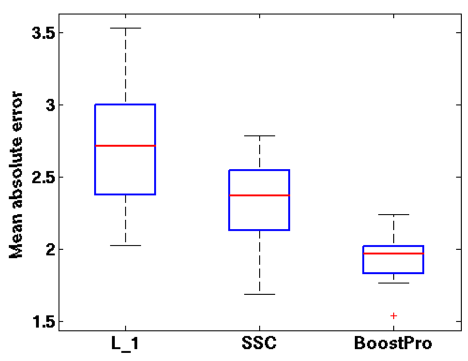
Data set	Error	Method	Source
MPG	7.11	SVM	[83]
CPU	28.14	Regression Trees	[113]
Housing	9.6	SVM	[83]
Abalone	4.31	Neural Network	[83]
Census	$1.5{\times}10^9$	Regression Trees	[113]
Letter	0.0195	ECOC with AdaBoost	[28]
Isolet	0.0372	SVM	[73]

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.

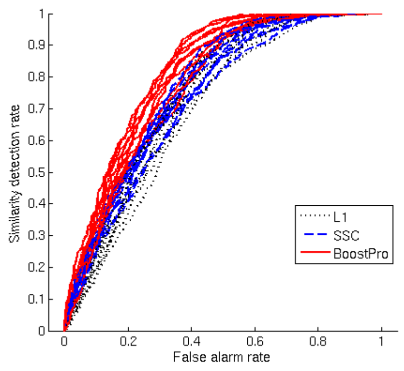
placing them on the figure and comparing the colors at their location. Thus, the pairs of red crosses would be classified as dissimilar (by the weak classifier alone!) while the pairs of circles would be classified as similar. The typical shape of h (origin-centered disks for norm, and "bow-tie" shapes 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

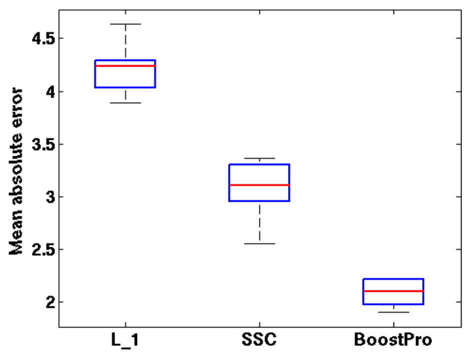


(a) Mean absolute error

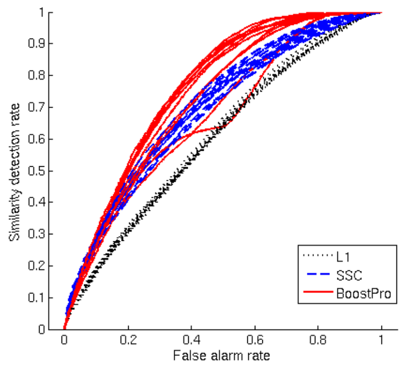


(b) ROC curve

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

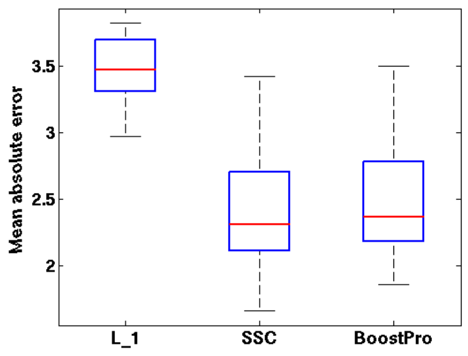


(b) ROC curve

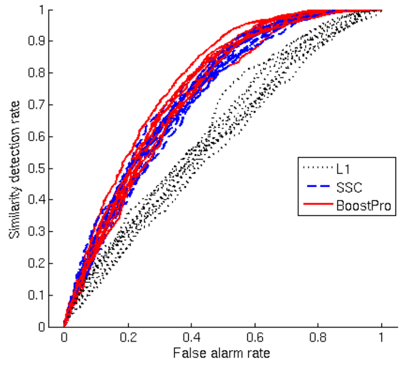
Figure 3-7: Results on Machine CPU. See legend for Figure 3-6.

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,

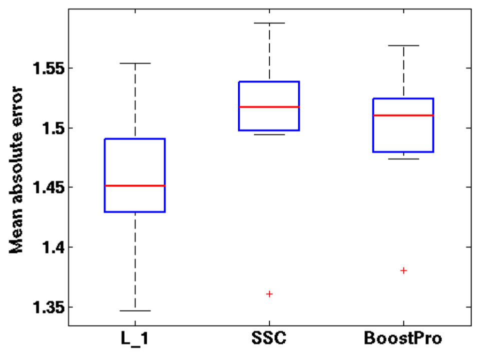


(a) Mean absolute error

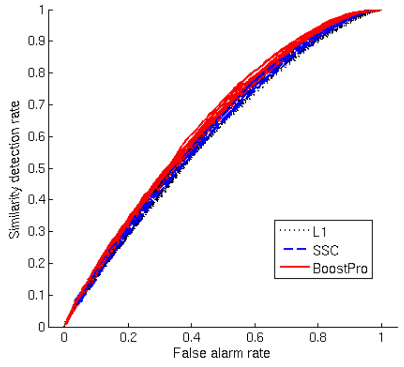


(b) ROC curve

Figure 3-8: Results on Boston Housing. See legend for Figure 3-6.



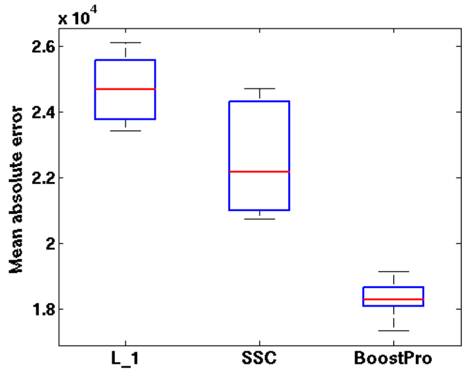
(a) Mean absolute error



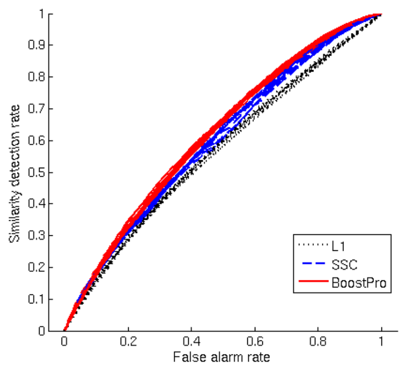
(b) ROC curve

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

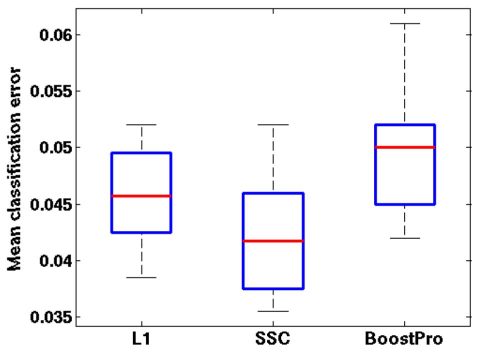


(a) Mean absolute error

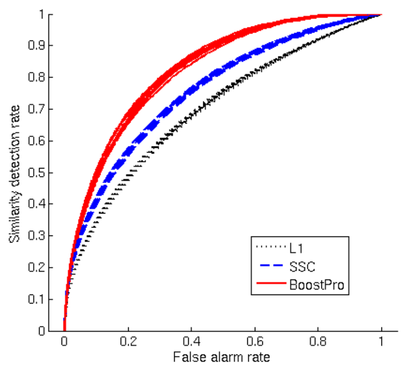


(b) ROC curve

Figure 3-10: Results on US Census. See legend for Figure 3-6.



(a) Mean classification error

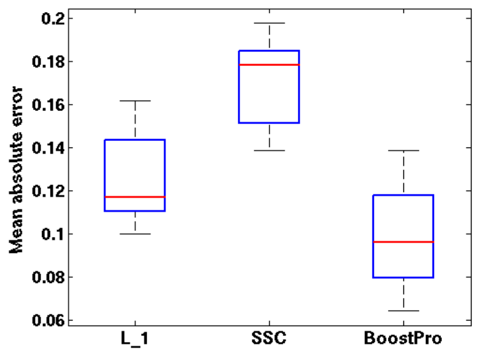


(b) ROC curve

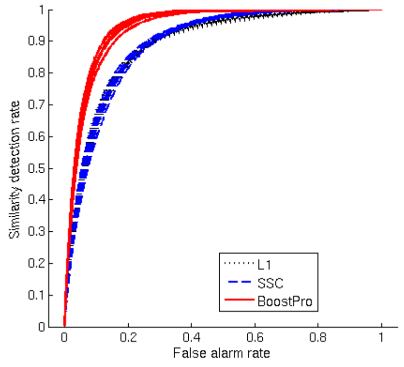
Figure 3-11: Results on Letter. Plots on the left show classification error distributions; otherwise, see legend for Figure 3-6.

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.

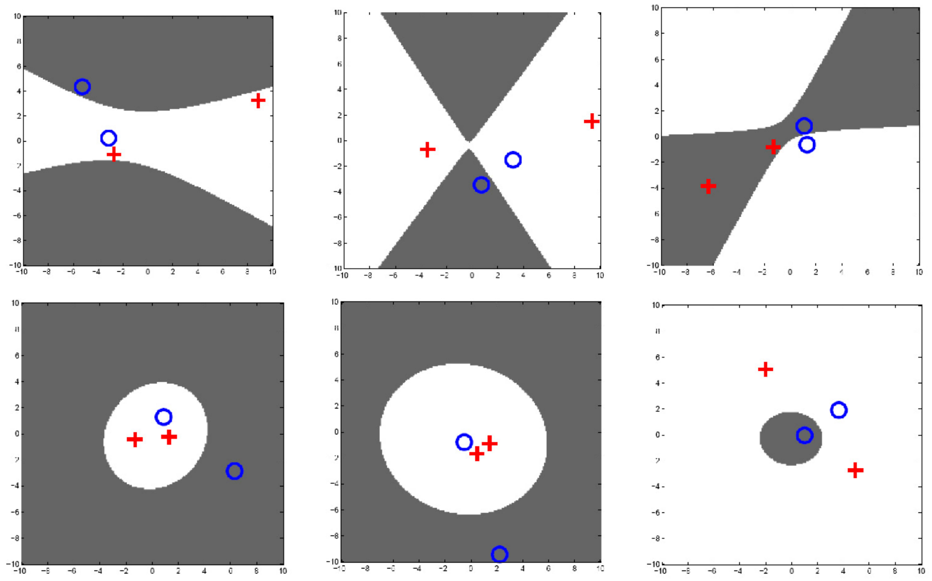


Figure 3-13: Typical weak classifiers (thresholded projections) learned for the angle (top three) and norm (bottom three) similarity on the synthetic 2D data. The darker shade corresponds to the area where $h = +1$. Crosses and circles show pairs that would be classified by the projection as similar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

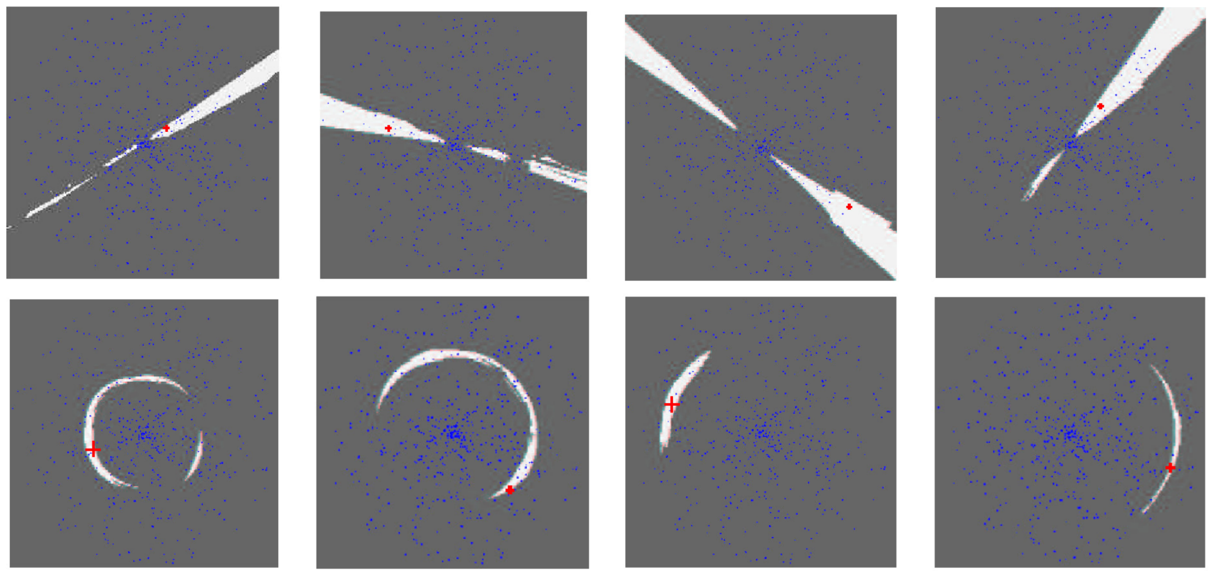
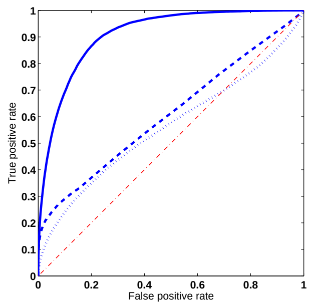
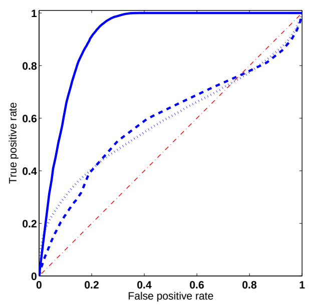


Figure 3-14: Synthetic data: example similarity regions. Light areas correspond to $\mathbf{x}$ such that $\|H(\mathbf{x}) - H(\mathbf{q})\|_H \leq R$, with the query $\mathbf{q}$ shown by the red cross and $R = 20$ set for 0.5 recall. The dots show the training data. Top: angle similarity, Bottom: norm similarity.



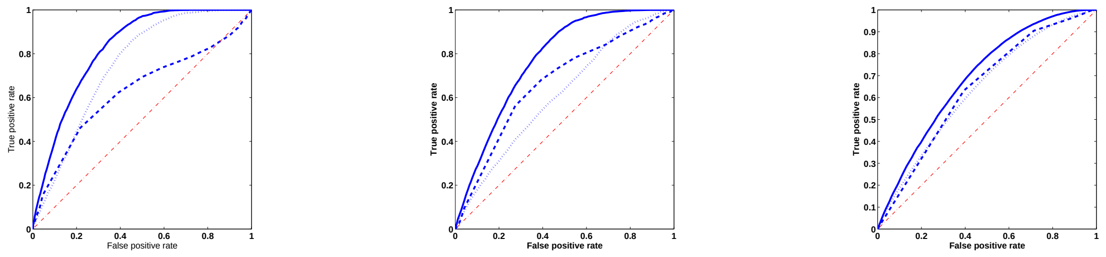
(a) Angle



(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.



(a) MPG (b) Housing (c) Abalone

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

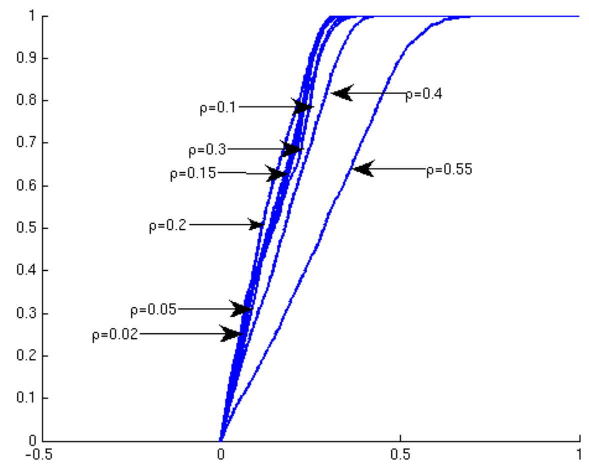


Figure 3-17: Effect of similarity rate on the performance of the semi-supervised BoostPro, on synthetic norm similarity data. The ROC curves are shown for the retrieval task on 1000 test points, using 600 unlabeled points and 2,000 similar pairs, with M=100.

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.