Similarity Learning for High-Dimensional Sparse Data

Kuan Liu
University of Southern California

Aurélien Bellet
Télécom ParisTech

Fei Sha
University of Southern California

Abstract

A good measure of similarity between data points is crucial to many tasks in machine learning. Similarity and metric learning methods learn such measures automatically from data, but they do not scale well respect to the dimensionality of the data. In this paper, we propose a method that can learn efficiently similarity measure from high-dimensional sparse data. The core idea is to parameterize the similarity measure as a convex combination of rank-one matrices with specific sparsity structures. The parameters are then optimized with an approximate Frank-Wolfe procedure to maximally satisfy relative similarity constraints on the training data. Our algorithm greedily incorporates one pair of features at a time into the similarity measure, providing an efficient way to control the number of active features and thus reduce overfitting. It enjoys very appealing convergence guarantees and its time and memory complexity depends on the sparsity of the data instead of the dimension of the feature space. Our experiments on real-world high-dimensional datasets demonstrate its potential for classification, dimensionality reduction and data exploration.

1 INTRODUCTION

In many applications, such as text processing, computer vision or biology, data is represented as very high-dimensional but sparse vectors. The ability to compute meaningful similarity scores between these objects is crucial to many tasks, such as classification, clustering or ranking. However, handcrafting a relevant similarity measure for such data is challenging because it is usually the case that only a small, often unknown subset of features is actually relevant to the task at hand. For instance, in drug discovery, chemical compounds can be represented as sparse features describing their 3D properties, and only a few of them play an role in determining whether the compound will bind to a target receptor (Guyon et al., 2004). In text classification, where each document is represented as a sparse bag of words, only a small subset of the words is generally sufficient to discriminate among documents of different topics.

A principled way to obtain a similarity measure tailored to the problem of interest is to learn it from data. This line of research, known as similarity and distance metric learning, has been successfully applied to many application domains (see Kulis, 2012; Bellet et al., 2013, for recent surveys). The basic idea is to learn the parameters of a similarity (or distance) function such that it satisfies proximity-based constraints, requiring for instance that some data instance $\boldsymbol{x}$ be more similar to $\boldsymbol{y}$ than to $\boldsymbol{z}$ according to the learned function. However, similarity learning typically requires estimating a matrix with $O(d^2)$ entries (where d is the data dimension) to account for correlation between pairs of features. For high-dimensional data (say, $d > 10^4$), this is problematic for at least three reasons: (i) training the metric is computationally expensive (quadratic or cubic in d), (ii) the matrix may not even fit in memory, and (iii) learning so many parameters is likely to lead to severe overfitting, especially for sparse data where some features are rarely observed.

To overcome this difficulty, a common practice is to first project data into a low-dimensional space (using PCA or random projections), and then learn a similarity function in the reduced space. Note that the projection intertwines useful features and irrelevant/noisy ones. Moreover, it is also difficult to interpret the reduced feature space, when we are interested in discovering what features are more important than others for discrimination.

In this paper, we propose a novel method to learn a bilinear similarity function $S_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{x}'$ directly in the original high-dimensional space while

Appearing in Proceedings of the 18th International Conference on Artificial Intelligence and Statistics (AISTATS) 2015, San Diego, CA, USA. JMLR: W&CP volume 38. Copyright 2015 by the authors.

avoiding the above-mentioned pitfalls. The main idea combines three ingredients: the sparsity of the data, the parameterization of $\boldsymbol{M}$ as a convex combination of rank-one matrices with special sparsity structures, and an approximate Frank-Wolfe procedure (Frank and Wolfe, 1956; Clarkson, 2010; Jaggi, 2013) to learn the similarity parameters. The resulting algorithm iteratively and greedily incorporates one pair of features at a time into the learned similarity, providing an efficient way to ignore irrelevant features as well as to guard against overfitting through early stopping. Our method has appealing approximation error guarantees, time and memory complexity independent of d and outputs extremely sparse similarity functions that are fast to compute and to interpret.

The usefulness of the proposed approach is evaluated on several datasets with up to 100,000 features, some of which have a large proportion of irrelevant features. To the best of our knowledge, this is the first time that a full similarity or distance metric is learned directly on such high-dimensional datasets without first reducing dimensionality. Our approach significantly outperforms both a diagonal similarity learned in the original space and a full similarity learned in a reduced space (after PCA or random projections). Furthermore, our similarity functions are extremely sparse (in the order of 0.0001% of nonzero entries), using a sparse subset of features and thus providing more economical analysis of the resulting model (for example, examining the importance of the original features and their pairwise interactions).

The rest of this paper is organized as follows. Section 2 briefly reviews some related work. Our approach is described in Section 3. We present our experimental results in Section 4 and conclude in Section 5.

2 RELATED WORK

Learning similarity and distance metric has attracted a lot of interests. In this section, we review previous efforts that focus on efficient algorithms for high-dimensional data – a comprehensive survey of existing approaches can be found in (Bellet et al., 2013).

A majority of learning similarity has focused on learning either a Mahalanobis distance $d_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}') = (\boldsymbol{x} - \boldsymbol{x}')^{\mathrm{T}}\boldsymbol{M}(\boldsymbol{x} - \boldsymbol{x}')$ where $\boldsymbol{M}$ is a symmetric positive semi-definite (PSD) matrix, or a bilinear similarity $S_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{x}'$ where $\boldsymbol{M}$ is an arbitrary $d \times d$ matrix. In both cases, it requires estimating $O(d^2)$ parameters, which is undesirable in the high-dimensional setting. Virtually all existing methods thus resort to dimensionality reduction (such as PCA or random projections) to preprocess the data when it has more than a few hundred dimensions, thereby incurring a potential loss of performance and interpretability of the resulting function (see e.g., Davis et al., 2007; Weinberger and Saul, 2009; Guillaumin et al., 2009; Ying and Li, 2012; Wang et al., 2012; Lim et al., 2013; Qian et al., 2014).

There have been a few solutions to this essential limitation. The most drastic strategy is to learn a diagonal matrix $\boldsymbol{M}$ (Schultz and Joachims, 2003; Gao et al., 2014), which is very restrictive as it amounts to a simple weighting of the features. Instead, some approaches assume an explicit low-rank decomposition $\boldsymbol{M} = \boldsymbol{L}^{\mathrm{T}}\boldsymbol{L}$ and learn $\boldsymbol{L} \in \mathbb{R}^{r \times d}$ in order to reduce the number of parameters to learn (Goldberger et al., 2004; Weinberger and Saul, 2009; Kedem et al., 2012). But this results in nonconvex formulations with many bad local optima (Kulis, 2012) and requires to tune r carefully. Moreover, the training complexity still depends on d and can thus remain quite large. Another direction is to learn $\boldsymbol{M}$ as a combination of rank-one matrices. In (Shen et al., 2012), the combining elements are selected greedily in a boosting manner but each iteration has an $O(d^2)$ complexity. To go around this limitation, Shi et al. (2014) generate a set of rank-one matrices before training and learn a sparse combination. However, as the dimension increases, a larger dictionary is needed and can be expensive to generate. Some work have also gone into sparse and/or low-rank regularization to reduce overfitting in high dimensions (Rosales and Fung, 2006; Qi et al., 2009; Ying et al., 2009) but those do not reduce the training complexity of the algorithm.

To the best of our knowledge, DML-eig (Ying and Li, 2012) and its extension DML-ρ (Cao et al., 2012) are the only prior attempts to use a Frank-Wolfe procedure for metric or similarity learning. However, their formulation requires finding the largest eigenvector of the gradient matrix at each iteration, which scales in $O(d^2)$ and is thus unsuitable for the high-dimensional setting we consider in this work.

3 PROPOSED APPROACH

This section introduces HDSL (High-Dimensional Similarity Learning), the approach proposed in this paper. We first describe our problem formulation (Section 3.1), then derive an efficient algorithm to solve it (Section 3.2).

3.1 Problem Formulation

In this work, we propose to learn a similarity function for high-dimensional sparse data. We assume the data points lie in some space $\mathcal{X} \subseteq \mathbb{R}^d$, where d is large ($d > 10^4$), and are D-sparse on average ($D \ll d$). Namely, the number of nonzero entries is typically much smaller than d. We focus on learning a similarity function $S_{\boldsymbol{M}} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ of the form $S_{\boldsymbol{M}}(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{M}\boldsymbol{x}'$, where $\boldsymbol{M} \in \mathbb{R}^{d\times d}$. Note that for any $\boldsymbol{M}$, $S_{\boldsymbol{M}}$ can be computed in $O(D^2)$ time on average.

Feasible domain Our goal is to derive an algorithm to learn a very sparse $\boldsymbol{M}$ with time and memory requirements that depend on D but not on d. To this end, given a scale parameter $\lambda > 0$, we will parameterize $\boldsymbol{M}$ as a convex combination of 4-sparse $d \times d$ bases:

$$\boldsymbol{M} \in \mathcal{D}_\lambda = \mathrm{conv}(\mathcal{B}_\lambda), \text{ with } \mathcal{B}_\lambda = \bigcup_{ij} \left\{ \boldsymbol{P}_\lambda^{(ij)}, \boldsymbol{N}_\lambda^{(ij)} \right\},$$

where for any pair of features $i, j \in \{1, \dots, d\}$, $i \neq j$,

$$\boldsymbol{P}_\lambda^{(ij)} = \lambda(\boldsymbol{e}_i + \boldsymbol{e}_j)(\boldsymbol{e}_i + \boldsymbol{e}_j)^T = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \lambda & \cdot & \lambda & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \lambda & \cdot & \lambda & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix},$$

$$\boldsymbol{N}_\lambda^{(ij)} = \lambda(\boldsymbol{e}_i - \boldsymbol{e}_j)(\boldsymbol{e}_i - \boldsymbol{e}_j)^T = \begin{pmatrix} \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & \lambda & \cdot & -\lambda & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \\ \cdot & -\lambda & \cdot & \lambda & \cdot \\ \cdot & \cdot & \cdot & \cdot & \cdot \end{pmatrix}.$$

The use of such sparse matrices was suggested by Jaggi (2011). Besides the fact that they are instrumental to the efficiency of our algorithm (see Section 3.2), we give some additional motivation for their use in the context of similarity learning.

First, any $\boldsymbol{M} \in \mathcal{D}_\lambda$ is a convex combination of symmetric PSD matrices and is thus also symmetric PSD. Unlike many metric learning algorithms, we thus avoid the $O(d^3)$ cost of projecting onto the PSD cone. Furthermore, constraining $\boldsymbol{M}$ to be symmetric PSD provides useful regularization to prevent overfitting (Chechik et al., 2009) and allows the use of the square root of $\boldsymbol{M}$ to project the data into a new space where the dot product is equivalent to $S_{\boldsymbol{M}}$. Because the bases in $\mathcal{B}_\lambda$ are rank-one, the dimensionality of this projection space is at most the number of bases composing $\boldsymbol{M}$.

Second, each basis operates on two features only. In particular, $S_{\boldsymbol{P}_\lambda^{(ij)}}(\boldsymbol{x}, \boldsymbol{x}') = \lambda(x_i x'_i + x_j x'_j + x_i x'_j + x_j x'_i)$ assigns a higher score when feature i appears jointly in $\boldsymbol{x}$ and $\boldsymbol{x}'$ (likewise for j), as well as when feature i in $\boldsymbol{x}$ and feature j in $\boldsymbol{y}$ (and vice versa) co-occur. Conversely, $S_{\boldsymbol{N}_\lambda^{(ij)}}$ penalizes the co-occurrence of features i and j. This will allow us to easily control the number of active features and learn a very compact similarity representation.

Finally, notice that in the context of text data represented as bags-of-words (or other count data), the bases in $\mathcal{B}_\lambda$ are quite natural: they can be intuitively thought of as encoding the fact that a term i or j present in both documents makes them more similar, and that two terms i and j are associated with the same/different class or topic.

Optimization problem We now describe the optimization problem to learn the similarity parameters. Following previous work (see for instance Schultz and Joachims, 2003; Weinberger and Saul, 2009; Chechik et al., 2009), our training data consist of side information in the form of triplet constraints:

$$\mathcal{T} = \{\boldsymbol{x}_t \text{ should be more similar to } \boldsymbol{y}_t \text{ than to } \boldsymbol{z}_t\}_{t=1}^{T}.$$

Such constraints can be built from a labeled training sample, provided directly by a domain expert, or obtained through implicit feedback such as clicks on search engine results. For notational convenience, we write $\boldsymbol{A}^t = \boldsymbol{x}_t(\boldsymbol{y}_t - \boldsymbol{z}_t)^{\mathrm{T}} \in \mathbb{R}^{d\times d}$ for each constraint $t = 1, \dots, T$. We want to define an objective function that applies a penalty when a constraint t is not satisfied with margin at least 1, i.e. whenever $\langle \boldsymbol{A}^t, \boldsymbol{M}\rangle = S_{\boldsymbol{M}}(\boldsymbol{x}_t, \boldsymbol{y}_t) - S_{\boldsymbol{M}}(\boldsymbol{x}_t, \boldsymbol{z}_t) < 1$. To this end, we use the smoothed hinge loss $\ell : \mathbb{R} \to \mathbb{R}^+$:

$$\ell\left(\langle \boldsymbol{A}^t, \boldsymbol{M}\rangle\right) = \begin{cases} 0 & \text{if } \langle \boldsymbol{A}^t, \boldsymbol{M}\rangle \geq 1 \\ \frac{1}{2} - \langle \boldsymbol{A}^t, \boldsymbol{M}\rangle & \text{if } \langle \boldsymbol{A}^t, \boldsymbol{M}\rangle \leq 0 \\ \frac{1}{2}\left(1 - \langle \boldsymbol{A}^t, \boldsymbol{M}\rangle\right)^2 & \text{otherwise} \end{cases},$$

where $\langle \cdot, \cdot \rangle$ denotes the Frobenius inner product.[1] Given $\lambda > 0$, our similarity learning formulation aims at finding the matrix $\boldsymbol{M} \in \mathcal{D}_\lambda$ that minimizes the average penalty over the triplet constraints in $\mathcal{T}$:

$$\min_{\boldsymbol{M}\in\mathcal{D}_\lambda} f(\boldsymbol{M}) = \frac{1}{T}\sum_{t=1}^{T} \ell\left(\langle \boldsymbol{A}^t, \boldsymbol{M}\rangle\right) \qquad (1)$$

Due to the convexity of the smoothed hinge loss, Problem (1) involves minimizing a convex function over the convex domain $\mathcal{D}_\lambda$. In the next section, we propose a greedy algorithm to solve this problem.

3.2 Algorithm

3.2.1 Exact Frank-Wolfe Algorithm

We propose to use a Frank-Wolfe (FW) algorithm (Frank and Wolfe, 1956; Clarkson, 2010; Jaggi, 2013) to learn the similarity. FW is a general procedure to minimize a convex and continuously differentiable function over a compact and convex set. At each iteration, it moves towards a feasible point that minimizes a linearization of the objective function at the current

[1] In principle, any other convex and continuously differentiable loss function can be used in our framework, such as the squared hinge loss, logistic loss or exponential loss.

Algorithm 1 Frank Wolfe algorithm for problem (1)

```
1: initialize $\boldsymbol{M}^{(0)}$ to an arbitrary $\boldsymbol{B} \in \mathcal{B}_\lambda$
2: for $k = 0, 1, 2, \dots$ do
3:     let $\boldsymbol{B}_F^{(k)} \in \arg\min_{\boldsymbol{B}\in\mathcal{B}_\lambda} \langle \boldsymbol{B}, \nabla f(\boldsymbol{M}^{(k)})\rangle$ and $\boldsymbol{D}_F^{(k)} = \boldsymbol{B}_F^{(k)} - \boldsymbol{M}^{(k)}$    // compute forward direction
4:     let $\boldsymbol{B}_A^{(k)} \in \arg\max_{\boldsymbol{B}\in\mathcal{S}^{(k)}} \langle \boldsymbol{B}, \nabla f(\boldsymbol{M}^{(k)})\rangle$ and $\boldsymbol{D}_A^{(k)} = \boldsymbol{M}^{(k)} - \boldsymbol{B}_A^{(k)}$    // compute away direction
5:     if $\langle \boldsymbol{D}_F^{(k)}, \nabla f(\boldsymbol{M}^{(k)})\rangle \leq \langle \boldsymbol{D}_A^{(k)}, \nabla f(\boldsymbol{M}^{(k)})\rangle$ then
6:         $\boldsymbol{D}^{(k)} = \boldsymbol{D}_F^{(k)}$ and $\gamma_{\max} = 1$    // choose forward step
7:     else
8:         $\boldsymbol{D}^{(k)} = \boldsymbol{D}_A^{(k)}$ and $\gamma_{\max} = \alpha^{(k)}_{\boldsymbol{B}_A^{(k)}}/(1 - \alpha^{(k)}_{\boldsymbol{B}_A^{(k)}})$    // choose away step
9:     end if
10:    let $\gamma^{(k)} \in \arg\min_{\gamma\in[0,\gamma_{\max}]} f(\boldsymbol{M}^{(k)} + \gamma \boldsymbol{D}^{(k)})$    // perform line search
11:    $\boldsymbol{M}^{(k+1)} = \boldsymbol{M}^{(k)} + \gamma^{(k)}\boldsymbol{D}^{(k)}$    // update iterate towards direction
12: end for
```

iterate. Note that a minimizer of this linear function must be at a vertex of the feasible domain. We will exploit the fact that in our formulation (1), the vertices of the feasible domain $\mathcal{D}_\lambda$ are the elements of $\mathcal{B}_\lambda$ and have special structure.

The FW algorithm applied to (1) and enhanced with so-called away steps (Guélat and Marcotte, 1986) is described in details in Algorithm 1. During the course of the algorithm, we explicitly maintain a representation of each iterate $\boldsymbol{M}^{(k)}$ as a convex combination of basis elements:

$$\boldsymbol{M}^{(k)} = \sum_{\boldsymbol{B}\in\mathcal{B}_\lambda} \alpha_{\boldsymbol{B}}^{(k)}\boldsymbol{B}, \text{ where } \sum_{\boldsymbol{B}\in\mathcal{B}_\lambda} \alpha_{\boldsymbol{B}}^{(k)} = 1, \alpha_{\boldsymbol{B}}^{(k)} \geq 0.$$

We denote the set of active basis elements in $\boldsymbol{M}^{(k)}$ as $\mathcal{S}^{(k)} = \{\boldsymbol{B} \in \mathcal{B}_\lambda : \alpha_{\boldsymbol{B}}^{(k)} > 0\}$. The algorithm goes as follows. We initialize $\boldsymbol{M}^{(0)}$ to a random basis element. Then, at each iteration, we greedily choose between moving towards a (possibly) new basis (forward step) or reducing the weight of an active one (away step). The extent of the step is determined by line search. As a result, Algorithm 1 adds only one basis (at most 2 new features) at each iteration, which provides a convenient way to control the number of active features and maintain a compact representation of $\boldsymbol{M}^{(k)}$ in $O(k)$ memory cost. Furthermore, away steps provide a way to reduce the importance of a potentially "bad" basis element added at an earlier iteration (or even remove it completely when $\gamma^{(k)} = \gamma_{\max}$). Note that throughout the execution of the algorithm, all iterates $\boldsymbol{M}^{(k)}$ remain convex combinations of basis elements and are thus feasible. The following lemma shows that the iterates of Algorithm 1 converge to an optimal solution of (1) with a rate of $O(1/k)$.

Lemma 1. *Let $\lambda > 0$, $\boldsymbol{M}^*$ be an optimal solution to (1) and $L = \frac{1}{T}\sum_{t=1}^T \|\boldsymbol{A}^t\|_F^2$. At any iteration $k \geq 1$ of Algorithm 1, the iterate $\boldsymbol{M}^{(k)} \in \mathcal{D}_\lambda$ satisfies $f(\boldsymbol{M}^{(k)}) - f(\boldsymbol{M}^*) \leq 16L\lambda^2/(k+2)$. Furthermore, it has at most rank $k+1$ with $4(k+1)$ nonzero entries, and uses at most $2(k+1)$ distinct features.*

Proof. The result follows from the analysis of the general FW algorithm (Jaggi, 2013), the fact that f has L-Lipschitz continuous gradient and observing that $\text{diam}_{\|\cdot\|_F}(\mathcal{D}_\lambda) = \sqrt{8}\lambda$. □

Note that the optimality gap in Lemma 1 is independent from d. This means that Algorithm 1 is able to find a good approximate solution based on a small number of features, which is very appealing in the high-dimensional setting.

3.2.2 Complexity Analysis

We now analyze the time and memory complexity of Algorithm 1. Observe that the gradient has the form $\nabla f(\boldsymbol{M}) = \frac{1}{T}\sum_{t=1}^T \boldsymbol{G}^t$, where

$$\boldsymbol{G}^t = \begin{cases} \boldsymbol{0} & \text{if } \langle \boldsymbol{A}^t, \boldsymbol{M}\rangle \geq 1 \\ -\boldsymbol{A}^t & \text{if } \langle \boldsymbol{A}^t, \boldsymbol{M}\rangle \leq 0 \\ (\langle \boldsymbol{A}^t, \boldsymbol{M}\rangle - 1)\,\boldsymbol{A}^t & \text{otherwise} \end{cases}.$$

The structure of the algorithm's updates is crucial to its efficiency: since $\boldsymbol{M}^{(k+1)}$ is a convex combination of $\boldsymbol{M}^{(k)}$ and a 4-sparse matrix $\boldsymbol{B}^{(k)}$, we can efficiently compute most of the quantities of interest through careful book-keeping.

In particular, storing $\boldsymbol{M}^{(k)}$ at iteration k requires $O(k)$ memory. We can also recursively compute $\langle \boldsymbol{A}^t, \boldsymbol{M}^{(k+1)}\rangle$ for all constraints in only $O(T)$ time and $O(T)$ memory based on $\langle \boldsymbol{A}^t, \boldsymbol{M}^{(k)}\rangle$ and $\langle \boldsymbol{A}^t, \boldsymbol{B}^{(k)}\rangle$. This allows us, for instance, to efficiently compute the objective value as well identify the set of satisfied constraints (those with $\langle \boldsymbol{A}^t, \boldsymbol{M}^{(k)}\rangle \geq 1$) and ignore them when computing the gradient. Finding the away direction at iteration k can be done in $O(Tk)$ time. For the line search, we use a bisection algorithm to find a

Table 1: Complexity of iteration k (ignoring logarithmic factors) for different variants of the algorithm.

Variant	Time	Memory
Exact	$\tilde{O}(TD^2 + Tk)$	$\tilde{O}(TD^2 + k)$
Mini-batch	$\tilde{O}(MD^2 + Tk)$	$\tilde{O}(T + MD^2 + k)$
Heuristic	$\tilde{O}(MD + Tk)$	$\tilde{O}(T + MD + k)$

root of the gradient of the 1-dimensional function of γ, which only depends on $\langle \boldsymbol{A}^t, \boldsymbol{M}^{(k)}\rangle$ and $\langle \boldsymbol{A}^t, \boldsymbol{B}^{(k)}\rangle$, both of which are readily available. Its time complexity is $O(T \log \frac{1}{\epsilon})$ where ϵ is the precision of the line-search, and requires constant memory.

The bottleneck is to find the forward direction. Indeed, sequentially considering each basis element is intractable as it takes $O(Td^2)$ time. A more efficient strategy is to sequentially consider each constraint, which requires $O(TD^2)$ time and $O(TD^2)$ memory. The overall iteration complexity of Algorithm 1 is given in Table 1.

3.2.3 Approximate Forward Step

Finding the forward direction can be expensive when T and D are both large. We propose two strategies to alleviate this cost by finding an approximately optimal basis (see Table 1 for iteration complexity).

Mini-Batch Approximation Instead of finding the forward and away directions based on the full gradient at each iteration, we estimate it on a mini-batch of $M \ll T$ constraints drawn uniformly at random (without replacement). The complexity of finding the forward direction is thus reduced to $O(MD^2)$ time and $O(MD^2)$ memory. Under mild assumptions, concentration bounds such as Hoeffding's inequality without replacement (Serfling, 1974) can be used to show that with high probability, the deviation between the "utility" of any basis element $\boldsymbol{B}$ on the full set of constraints and its estimation on the mini-batch, namely:

$$\left| \frac{1}{M} \sum_{t=1}^{M} \langle \boldsymbol{B}, \boldsymbol{G}^t\rangle - \frac{1}{T} \sum_{t=1}^{T} \langle \boldsymbol{B}, \boldsymbol{G}^t\rangle \right|,$$

decreases as $O(1/\sqrt{M})$. In other words, the mini-batch variant of Algorithm 1 finds a forward direction which is approximately optimal. The FW algorithm is known to be robust to this setting, and convergence guarantees similar to Lemma 1 can be obtained following (Jaggi, 2013; Freund and Grigas, 2013).

Fast Heuristic To avoid the quadratic dependence on D, we propose to use the following heuristic to find a good forward basis. We first pick a feature $i \in \{1, \dots, d\}$ uniformly at random, and solve the linear problem over the restricted set $\bigcup_j \{\boldsymbol{P}_\lambda^{(ij)}, \boldsymbol{N}_\lambda^{(ij)}\}$. We then fix j and solve the problem again over the set $\bigcup_k \{\boldsymbol{P}_\lambda^{(kj)}, \boldsymbol{N}_\lambda^{(kj)}\}$ and use the resulting basis for the forward direction. This can be done in only $O(MD)$ time and $O(MD)$ memory and gives good performance in practice, as we shall see in the next section.

4 EXPERIMENTS

In this section, we present experiments to study the performance of HDSL in classification, dimensionality reduction and data exploration against competing approaches.

4.1 Experimental Setup

Datasets We report experimental results on several real-world classification datasets with up to 100,000 features. Dorothea and dexter come from the NIPS 2003 feature selection challenge (Guyon et al., 2004) and are respectively pharmaceutical and text data with predefined splitting into training, validation and test sets. They both contain a large proportion of noisy/irrelevant features. Reuters CV1 is a popular text classification dataset with bag-of-words representation. We use the binary version from the LIBSVM dataset collection[2] (with 60%/20%/20% random splits) and the 4-classes version (with 40%/30%/30% random splits) introduced in (Cai and He, 2012). Detailed information on the datasets and splits is given in Table 2. All datasets are normalized such that each feature takes values in $[0, 1]$.

Competing Methods We compare the proposed approach (HDSL) to several methods:

- IDENTITY: The standard dot product as a baseline, which corresponds to using $\boldsymbol{M} = \boldsymbol{I}$.

- DIAG: Diagonal similarity learning (i.e., a weighting of the features), as done in Gao et al. (2014). We obtain it by minimizing the same loss as in HDSL with ℓ_2 and ℓ_1 regularization, i.e.,

$$\min_{\boldsymbol{w} \in \mathbb{R}^d} \quad f(\boldsymbol{w}) = \frac{1}{T} \sum_{t=1}^{T} \ell\left(\langle \boldsymbol{A}^t, \mathrm{diag}(\boldsymbol{w})\rangle\right) + \lambda \Omega(\boldsymbol{w}),$$

  where $\Omega(\boldsymbol{w}) \in \{\|\boldsymbol{w}\|_2^2, \|\boldsymbol{w}\|_1\}$ and λ is the regularization parameter. Optimization is done using (proximal) gradient descent.

[2]http://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

Table 2: Datasets used in the experiments.

Datasets	Dimension	Sparsity	Training size	Validation size	Test size
dexter	20,000	0.48%	300	300	2,000
dorothea	100,000	0.91%	800	350	800
rcv1_2	47,236	0.16%	12,145	4,048	4,049
rcv1_4	29,992	0.26%	3,850	2,888	2,887

- RP+OASIS: Similarity learning in random projected space. Given $r \ll d$, let $\boldsymbol{R} \in \mathbb{R}^{d\times r}$ be a matrix where each entry r_{ij} is randomly drawn from $\mathcal{N}(0,1)$. For each data instance $\boldsymbol{x} \in \mathbb{R}^d$, we generate $\tilde{\boldsymbol{x}} = \frac{1}{\sqrt{r}}\boldsymbol{R}^{\mathrm{T}}\boldsymbol{x} \in \mathbb{R}^r$ and use this reduced data in OASIS (Chechik et al., 2009), a fast online method to learn a bilinear similarity from triplet constraints.

- PCA+OASIS: Similarity learning in PCA space. Same as RP+OASIS, except that PCA is used instead of random projections to project the data into $\mathbb{R}^r$.

- SVM: Support Vector Machines. We use linear SVM, which is known to perform well for sparse high-dimensional data (Caruana et al., 2008), with ℓ_2 and ℓ_1 regularization. We also use nonlinear SVM with the polynomial kernel (2nd and 3rd degree) popular in text classification (Chang et al., 2010). The SVM models are trained using liblinear (Fan et al., 2008) and libsvm (Chang and Lin, 2011) with 1vs1 paradigm for multiclass.

Training Procedure For all similarity learning algorithms, we generate 15 training constraints for each instance by identifying its 3 target neighbors (nearest neighbors with same label) and 5 impostors (nearest neighbors with different label), following Weinberger and Saul (2009). Due to the very small number of training instances in dexter, we found that better performance is achieved using 20 constraints per instance $\boldsymbol{x}$, each of them constructed by randomly drawing a point from the class of $\boldsymbol{x}$ and a point from a different class. All parameters are tuned using the accuracy on the validation set. For HDSL, we use the fast heuristic described in Section 3.2.3 and tune the scale parameter $\lambda \in \{1, 10, \dots, 10^9\}$. The regularization parameters of DIAG and SVM are tuned in $\{10^{-9}, \dots, 10^8\}$ and the "aggressiveness" parameter of OASIS is tuned in $\{10^{-9}, \dots, 10^2\}$.

4.2 Results

Classification Performance We first investigate the performance of each similarity learning approach in k-NN classification (k was set to 3 for all experiments). For RP+OASIS and PCA+OASIS, we choose the dimension r of the reduced space based on the accuracy of the learned similarity on the validation set, limiting our search to $r \leq 2000$ because OASIS is extremely slow beyond this point.[3] Similarly, we use the performance on validation data to do early stopping in HDSL, which also has the effect of restricting the number of features used by the learned similarity.

Table 3 shows the k-NN classification performance. First, notice that RP+OASIS often performs worse than IDENTITY, which is consistent with previous observations that a large number of random projections may be needed to obtain good performance (Fradkin and Madigan, 2003). PCA+OASIS gives much better results, but is generally outperformed by a simple diagonal similarity learned directly in the original high-dimensional space. HDSL, however, outperforms all other algorithms on these datasets, including DIAG. This shows the good generalization performance of the proposed approach, even though the number of training samples is sometimes very small compared to the number of features, as in dexter and dorothea. It also shows the relevance of encoding "second order" information (pairwise interactions between the original features) in the similarity instead of simply considering a feature weighting as in DIAG.

Table 4 shows the comparison with SVMs. Interestingly, HDSL with k-NN outperforms all SVM variants on dexter and dorothea, both of which have a large proportion of irrelevant features. This shows that its greedy strategy and early stopping mechanism achieves better feature selection and generalization than the ℓ_1 version of linear SVM. On the other two datasets, HDSL is competitive with SVM, although it is outperformed slightly by one variant (SVM-poly-3 on rcv1_2 and SVM-linear-ℓ_2 on rcv1_4), both of which rely on all features.

Feature Selection and Sparsity We now focus on the ability of HDSL to perform feature selection and more generally to learn sparse similarity functions. To

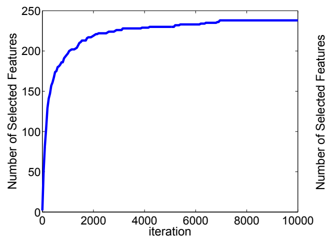
[3]Note that the number of PCA dimensions is at most the number of training examples. Therefore, for dexter and dorothea, r is at most 300 and 800 respectively.

Table 3: k-NN test error (%) of the similarities learned with each method. The number of features used by each similarity (when smaller than d) is given in brackets. Best accuracy on each dataset is shown in bold.

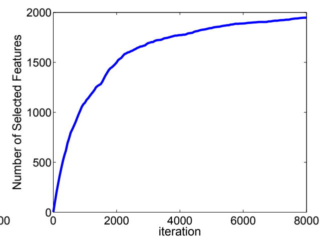
Datasets	IDENTITY	RP+OASIS	PCA+OASIS	DIAG-ℓ_2	DIAG-ℓ_1	HDSL
dexter	20.1	24.0 [1000]	9.3 [50]	8.4	8.4 [773]	**6.5** [183]
dorothea	9.3	11.4 [150]	9.9 [800]	6.8	6.6 [860]	**6.5** [731]
rcv1_2	6.9	7.0 [2000]	4.5 [1500]	3.5	3.7 [5289]	**3.4** [2126]
rcv1_4	11.2	10.6 [1000]	6.1 [800]	6.2	7.2 [3878]	**5.7** [1888]

Table 4: Test error (%) of several SVM variants compared to HDSL. As in Table 3, the number of features is given in brackets and best accuracies are shown in bold.

Datasets	SVM-poly-2	SVM-poly-3	SVM-linear-ℓ_2	SVM-linear-ℓ_1	HDSL
dexter	9.4	9.2	8.9	8.9 [281]	**6.5** [183]
dorothea	7	6.6	8.1	6.6 [366]	**6.5** [731]
rcv1_2	3.4	**3.3**	3.5	4.0 [1915]	3.4 [2126]
rcv1_4	5.7	5.7	**5.1**	5.7 [2770]	5.7 [1888]



(a) dexter dataset



(b) rcv1_4 dataset

Figure 1: Number of active features learned by HDSL as a function of the iteration number.

better understand the behavior of HDSL, we show in Figure 1 the number of selected features as a function of the iteration number for two of the datasets. Remember that at most two new features can be added at each iteration. Figure 1 shows that HDSL incorporates many features early on but tends to eventually converge to a modest fraction of features (the same observation holds for the other two datasets). This may explain why HDSL does not suffer much from overfitting even when training data is scarce as in dexter.

Another attractive characteristic of HDSL is its ability to learn a matrix that is sparse not only on the diagonal but also off-diagonal (the proportion of nonzero entries is in the order of 0.0001% for all datasets). In other words, it only relies on a few relevant pairwise interactions between features. Figure 2 shows two examples, where we can see that HDSL is able to exploit the product of two features as either a positive or negative contribution to the similarity score. This opens the door to an analysis of the importance of pairs of features (for instance, word co-occurrence) for the application at hand. Finally, the extreme sparsity of the matrices allows very fast similarity computation.

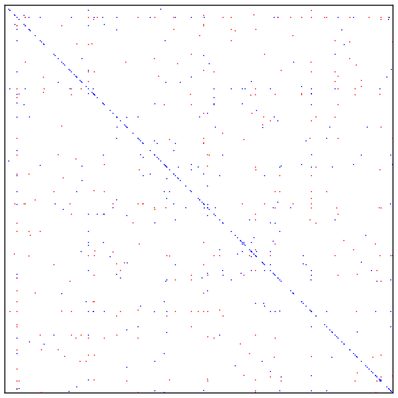
Finally, it is also worth noticing that HDSL uses significantly less features than DIAG-ℓ_1 (see numbers in brackets in Table 3). We attribute this to the extra modeling capability brought by the non-diagonal similarity observed in Figure 2.[4]

Dimension Reduction We now investigate the potential of HDSL for dimensionality reduction. Recall that HDSL learns a sequence of PSD matrices $\boldsymbol{M}^{(k)}$. We can use the square root of $\boldsymbol{M}^{(k)}$ to project the data into a new space where the dot product is equivalent to $S_{\boldsymbol{M}^{(k)}}$ in the original space. The dimension of the projection space is equal to the rank of $\boldsymbol{M}^{(k)}$, which is upper bounded by $k+1$. A single run of HDSL can thus be seen as incrementally building projection spaces of increasing dimensionality.

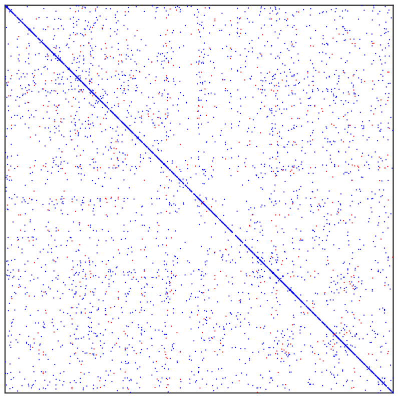
To assess the dimensionality reduction quality of HDSL (measured by k-NN classification error on the test set), we plot its performance at various iterations during the runs that gave the results in Table 3. We compare it to two standard dimensionality reduction techniques: random projection and PCA. We also evaluate RP+OASIS and PCA+OASIS, i.e., learn a similarity with OASIS on top of the RP and PCA features.[5] Note that OASIS was tuned separately for each projection size, making the comparison a bit unfair to HDSL. The results are shown in Figure 3. As observed ear-

[4]Note that HDSL uses roughly the same number of features as SVM-linear-ℓ_1 (Table 4), but drawing any conclusion is harder because the objective and training data for each method are different. Moreover, 1-vs-1 SVM combines several binary models to deal with the multiclass setting.

[5]Again, we were not able to run OASIS beyond a certain dimension due to computational complexity.

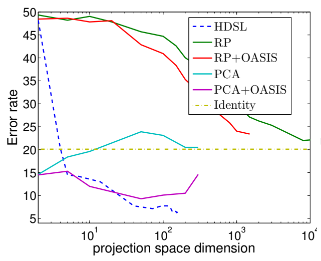


(a) dexter ($20,000 \times 20,000$ matrix, 712 nonzeros)

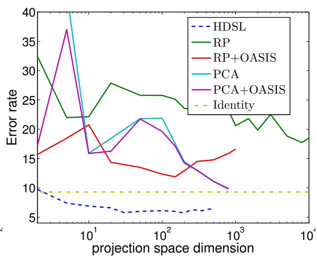


(b) rcv1_4 ($29,992 \times 29,992$ matrix, 5263 nonzeros)

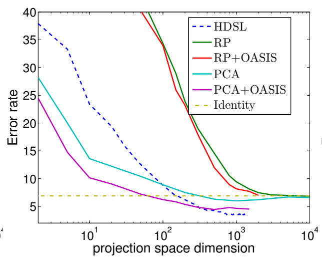
Figure 2: Sparsity structure of the matrix $\boldsymbol{M}$ learned by HDSL. Positive and negative entries are shown in blue and red, respectively (best seen in color).



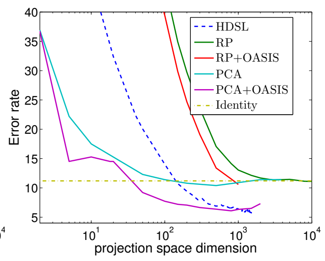
(a) dexter dataset



(b) dorothea dataset



(c) rcv1_2 dataset



(d) rcv1_4 dataset

Figure 3: k-NN test error as a function of the dimensionality of the space (in log scale). Best seen in color.

lier, random projection-based approaches achieve poor performance. When the features are not too noisy (as in rcv1_2 and rcv1_4), PCA-based methods are better than HDSL at compressing the space into very few dimensions, but HDSL eventually catches up. On the other hand, PCA suffers heavily from the presence of noise (dexter and dorothea), while HDSL is able to quickly improve upon the standard similarity in the original space. Finally, on all datasets, we observe that HDSL converges to a stationary dimension without overfitting, unlike PCA+OASIS which exhibits signs of overfitting on dexter and rcv1_4 especially.

5 CONCLUSION

In this work, we proposed an efficient approach to learn similarity functions from high-dimensional sparse data. This is achieved by forming the similarity as a combination of simple sparse basis elements that operate on only two features and the use of an (approximate) Frank-Wolfe algorithm. Experiments on real-world datasets confirmed the robustness of the approach to noisy features and its usefulness for classification and dimensionality reduction. Together with the extreme sparsity of the learned similarity, this makes our approach potentially useful in a variety of other contexts, from data exploration to clustering and ranking, and more generally as a way to preprocess the data before applying any learning algorithm.

Acknowledgements

Most of the work was conducted when A. Bellet was affiliated with the U. of Southern California.

This work was in part supported by the Intelligence Advanced Research Projects Activity (IARPA) via Department of Defense U.S. Army Research Laboratory (DoD / ARL) contract number W911NF-12-C-0012, a NSF IIS-1065243, an Alfred. P. Sloan Research Fellowship, DARPA award D11AP00278, and an ARO YIP Award (W911NF-12-1-0241). The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of IARPA, DoD/ARL, or the U.S. Government.

References

Aurélien Bellet, Amaury Habrard, and Marc Sebban. A Survey on Metric Learning for Feature Vectors and Structured Data. Technical report, arXiv:1306.6709, June 2013.

Deng Cai and Xiaofei He. Manifold Adaptive Experimental Design for Text Categorization. *TKDE*, 24 (4):707–719, 2012.

Qiong Cao, Yiming Ying, and Peng Li. Distance Metric Learning Revisited. In *ECML/PKDD*, pages 283–298, 2012.

Rich Caruana, Nikolaos Karampatziakis, and Ainur Yessenalina. An empirical evaluation of supervised learning in high dimensions. In *ICML*, pages 96–103, 2008.

Chih-Chung Chang and Chih-Jen Lin. LIBSVM : a library for support vector machines. *ACM TIST*, 2 (3):27–27, 2011.

Yin-Wen Chang, Cho-Jui Hsieh, Kai-Wei Chang, Michael Ringgaard, and Chih-Jen Lin. Training and Testing Low-degree Polynomial Data Mappings via Linear SVM. *JMLR*, 11:1471–1490, 2010.

Gal Chechik, Uri Shalit, Varun Sharma, and Samy Bengio. An online algorithm for large scale image similarity learning. In *NIPS*, pages 306–314, 2009.

Kenneth L. Clarkson. Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. *ACM Transactions on Algorithms*, 6(4):1–30, 2010.

Jason V. Davis, Brian Kulis, Prateek Jain, Suvrit Sra, and Inderjit S. Dhillon. Information-theoretic metric learning. In *ICML*, pages 209–216, 2007.

Rong-En Fan, Kai-Wei Chang, Cho-Jui Hsieh, Xiang-Rui Wang, and Chih-Jen Lin. LIBLINEAR: A Library for Large Linear Classification. *JMLR*, 9: 1871–1874, 2008.

Dmitriy Fradkin and David Madigan. Experiments with random projections for machine learning. In *KDD*, pages 517–522, 2003.

Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, 1956.

Robert M. Freund and Paul Grigas. New Analysis and Results for the Conditional Gradient Method. Technical report, arXiv:1307.0873, 2013.

Xingyu Gao, Steven C.H. Hoi, Yongdong Zhang, Ji Wan, and Jintao Li. SOML: Sparse Online Metric Learning with Application to Image Retrieval. In *AAAI*, pages 1206–1212, 2014.

Jacob Goldberger, Sam Roweis, Geoff Hinton, and Ruslan Salakhutdinov. Neighbourhood Components Analysis. In *NIPS*, pages 513–520, 2004.

Jacques Guélat and Patrice Marcotte. Some comments on Wolfe's away step. *Mathematical Programming*, 35(1):110–119, 1986.

Matthieu Guillaumin, Jakob J. Verbeek, and Cordelia Schmid. Is that you? Metric learning approaches for face identification. In *ICCV*, pages 498–505, 2009.

Isabelle Guyon, Steve R. Gunn, Asa Ben-Hur, and Gideon Dror. Result Analysis of the NIPS 2003 Feature Selection Challenge. In *NIPS*, 2004.

Martin Jaggi. *Sparse Convex Optimization Methods for Machine Learning.* PhD thesis, ETH Zurich, 2011.

Martin Jaggi. Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization. In *ICML*, 2013.

Dor Kedem, Stephen Tyree, Kilian Weinberger, Fei Sha, and Gert Lanckriet. Non-linear Metric Learning. In *NIPS*, pages 2582–2590, 2012.

Brian Kulis. Metric Learning: A Survey. *Foundations and Trends in Machine Learning*, 5(4):287–364, 2012.

Daryl K. Lim, Brian McFee, and Gert Lanckriet. Robust Structural Metric Learning. In *ICML*, 2013.

Guo-Jun Qi, Jinhui Tang, Zheng-Jun Zha, Tat-Seng Chua, and Hong-Jiang Zhang. An Efficient Sparse Metric Learning in High-Dimensional Space via l1-Penalized Log-Determinant Regularization. In *ICML*, 2009.

Qi Qian, Rong Jin, Shenghuo Zhu, and Yuanqing Lin. An Integrated Framework for High Dimensional Distance Metric Learning and Its Application to Fine-Grained Visual Categorization. Technical report, arXiv:1402.0453, 2014.

Romer Rosales and Glenn Fung. Learning Sparse Metrics via Linear Programming. In *KDD*, pages 367–373, 2006.

Matthew Schultz and Thorsten Joachims. Learning a Distance Metric from Relative Comparisons. In *NIPS*, 2003.

Robert J. Serfling. Probability inequalities for the sum in sampling without replacement. *The Annals of Statistics*, 2(1):39–48, 1974.

Chunhua Shen, Junae Kim, Lei Wang, and Anton van den Hengel. Positive Semidefinite Metric Learning Using Boosting-like Algorithms. *JMLR*, 13: 1007–1036, 2012.

Yuan Shi, Aurélien Bellet, and Fei Sha. Sparse Compositional Metric Learning. In *AAAI*, pages 2078–2084, 2014.

Jun Wang, Adam Woznica, and Alexandros Kalousis. Parametric Local Metric Learning for Nearest

Neighbor Classification. In *NIPS*, pages 1610–1618, 2012.

Kilian Q. Weinberger and Lawrence K. Saul. Distance Metric Learning for Large Margin Nearest Neighbor Classification. *JMLR*, 10:207–244, 2009.

Yiming Ying and Peng Li. Distance Metric Learning with Eigenvalue Optimization. *JMLR*, 13:1–26, 2012.

Yiming Ying, Kaizhu Huang, and Colin Campbell. Sparse Metric Learning via Smooth Optimization. In *NIPS*, pages 2214–2222, 2009.