Efficient and Numerically Stable Sparse Learning

Sihong Xie[1], Wei Fan[2], Olivier Verscheure[2], and Jiangtao Ren[1]

[1] Sun Yat-Sen University, Guangzhou, China
{xiesihong1, issrjt}@gmail.com
[2] IBM T.J. Watson Research Center, New York, USA
{weifan, ov1}@us.ibm.com

Abstract. We consider the problem of numerical stability and model density growth when training a sparse linear model from massive data. We focus on scalable algorithms that optimize certain loss function using gradient descent, with either ℓ_0 or ℓ_1 regularization. We observed numerical stability problems in several existing methods, leading to divergence and low accuracy. In addition, these methods typically have weak controls over sparsity, such that model density grows faster than necessary. We propose a framework to address the above problems. First, the update rule is numerically stable with convergence guarantee and results in more reasonable models. Second, besides ℓ_1 regularization, it exploits the sparsity of data distribution and achieves a higher degree of sparsity with a PAC generalization error bound. Lastly, it is parallelizable and suitable for training large margin classifiers on huge datasets. Experiments show that the proposed method converges consistently and outperforms other baselines using 10% of features by as much as 6% reduction in error rate on average. Datasets and software are available from the authors.

1 Introduction

In this paper, we focus on training a sparse large margin model. Assume that we are given m labeled examples $Z = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, where $\mathbf{x}_i \in \mathbb{R}^d, i = 1, \ldots, m$. We aim at solving the following optimization problem:

$$\min_{\mathbf{w}\in\mathbb{R}^d} \frac{1}{m}\sum_{i=1}^{m} L(\langle \mathbf{w}, \mathbf{x}_i\rangle, y_i) + \lambda\|\mathbf{w}\|_1 \tag{1}$$

L is any smooth and differentiable loss function such as logistic or hinge loss. λ is the parameter for trading off between loss and ℓ_1 regularization. We are interested in scalable algorithm, with parallelizable data accesses and small communication cost. One can find such application in text mining and webspam detection, where the number of features could be in millions.

Sparse learning aims at accurate models using a small number of non-zero elements, with the advantages of efficiency and generalizability [13]. Some existing methods produce sparse models by forward-backward feature selection [13] or boosting [3]. Though effective, these methods have to scan the training set

Table 1: Notations and definitions

Notation	definition	Notation	Definition
$\mathbf{x}$	Example in $\mathbb{R}^d$	X	Data matrix
y	Label of $\mathbf{x}$	Z	A series of m labeled examples
$\mathbf{w}$	Linear model in $\mathbb{R}^d$	$\boldsymbol{\theta}$	Dual vector of $\mathbf{w}$ in Eq. (8)
$\mathbf{w}_Z$	Linear model learned from Z	$\mathbf{w}^*$	Optimal linear model
$\tilde{\mathbf{w}}$	Vector to be thresholded	$H(\cdot, s)$	Hard-thresholding function
$S(\cdot, \lambda)$	Soft-thresholding function	$\nabla f(\cdot)$	Gradient of $f(\cdot)$
λ	ℓ_1 regularization parameter	t	Index of iterations
η	Learning rate	$[d]$	Set of indices $\{1, \ldots, d\}$
J	Set of indices, $J \subseteq [d]$	$\|\mathbf{v}\|_0$	The support of $\mathbf{v}$
$\|\mathbf{v}\|_1$	The sum of $|v_i|$, $i \in [d]$	$\|\mathbf{v}\|_2$	The Euclidean length of $\mathbf{v}$
$\|\mathbf{v}\|_\infty$	The maximum of $|v_i|$	σ_i	Eigenvalues

at each iteration, which is expensive or even impossible for large datasets. Under some restrictive assumptions [14], one can achieve sparse models via convex programming with ℓ_1 regularization or constraint. Though scalable as methods in [8, 12], there are two main drawbacks. First, numerical problems can lead to iteration divergence or summation cancellation, making the algorithm less useful. Second, ℓ_1 regularization only encourages sparsity and may not be enough for sparse learning. As the training proceeds, model complexity can grow faster than necessary and result in dense models, regardless of the regularization.

In this paper, we propose a perceptron-based algorithm to address the above problems. First, it is numerically stable with convergence guarantee. Second, in addition to ℓ_1 regularization, it further takes advantage of the sparsity of training examples to achieve a higher degree of model sparsity, and furthermore, a generalization error bound which is not provided in [8, 12]. Experiments show that the proposed method converges with steadily reduced test error rates as the training proceeds. and consistently outperforms previous state-of-the-art algorithms by as much as 8.4% in accuracy using only 10% of all features in one of the tasks.

2 Numerical Challenges in Sparse Learning

Notations are summarized in Table 1. Note that an element of a vector is in normal font, for example, w_j is the j-th element of the vector $\mathbf{w}$. About norms, for any $\mathbf{v} \in \mathbb{R}^d$, $\|\mathbf{v}\|_{p_1} \leq \|\mathbf{v}\|_{p_2}$ for any $p_1 \geq p_2$ [6]. If not otherwise specified, $\|\cdot\|$ is equivalent to $\|\cdot\|_2$

We have observed numerical problems in two sparse learning algorithms using either ℓ_1 or ℓ_0 regularization.

2.1 Numerical Problems of Direct Iterative Methods

The first problem arises when directly applying matrix iterative methods to solve a system of linear equations

$$\mathbf{y} = X\mathbf{w} \tag{2}$$

where $X = (\mathbf{x}_1, \ldots, \mathbf{x}_m)^\top$. ℓ_1 or ℓ_0 regularization are used to obtain sparse model, as in compressive sensing or sparse learning. Suppose there exists an optimal solution $\mathbf{w}^*$ satisfying Eq. (2), namely $\mathbf{y} - X\mathbf{w}^* = \mathbf{0}$ ($\mathbf{0}$ denotes the zero vector). The direct method tries to find $\mathbf{w}^*$ by minimizing the potential function

$$\Psi(\mathbf{w}) = \frac{1}{2}\|\mathbf{y} - X\mathbf{w}\|^2 \tag{3}$$

Starting with an arbitrary vector $\mathbf{w}^{(0)}$ (usually $\mathbf{0}$), it follows the gradient direction at each iteration to reduce the potential function value. The gradient of Eq. (3) is $\nabla\Psi(\mathbf{w}) = -X^\top(\mathbf{y} - X\mathbf{w})$ and we update $\mathbf{w}^{(t)}$ by

$$\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} + \eta X^\top(\mathbf{y} - X\mathbf{w}^{(t-1)}) \tag{4}$$

where η is the learning rate. To see when the above iteration fails to converge, let $\varepsilon^{(t)} = \mathbf{w}^{(t)} - \mathbf{w}^*$ be the error vector.

$$\begin{aligned}\varepsilon^{(t)} &= \mathbf{w}^{(t-1)} + \eta X^\top(\mathbf{y} - X\mathbf{w}^{(t-1)}) - \mathbf{w}^* - \eta X^\top(\mathbf{y} - X\mathbf{w}^*) \\ &= (I - \eta X^\top X)(\mathbf{w}^{(t-1)} - \mathbf{w}^*) = M\varepsilon^{(t-1)}\end{aligned}$$

where $M = I - \eta X^\top X$ is the iteration matrix. Thus for $t > 0$ we have

$$\|\varepsilon^{(t)}\| = \|\mathbf{w}^{(t)} - \mathbf{w}^*\| = \|M\varepsilon^{(t-1)}\| = \cdots = \|M^t\varepsilon^{(0)}\|$$

The sufficient and necessary condition for the update rule (4) to converge is that the spectral radius of M, $\rho(M) = \max_i\{|\sigma_i|\}$less than 1, where σ_i is the eigenvalues of M. Formally,

$$\lim_{t\to\infty} \|M^t\| = 0 \Leftrightarrow \rho(M) < 1 \tag{5}$$

The above analysis shows that if one cannot guarantee $\rho(M) < 1$ at each iteration, then update rule (4) may not be applicable in reducing the potential function.

In addition, we would like $\mathbf{w}^{(t)}$ to be sparse. It is shown in [5] that under the RIP condition [1], a truncated version of update rule (4) reduces the potential function at each iteration while maintaining a sparse solution. We show that the truncation does not automatically guarantee the convergence of ℓ_0 constrained gradient descent. Consider the following iteration

$$\mathbf{w}^{(t)} = H(\tilde{\mathbf{w}}^{(t)}, s) = D_{J^{(t)}}(\mathbf{w}^{(t-1)} + \eta X^\top(\mathbf{y} - X\mathbf{w}^{(t-1)})) \tag{6}$$

with $\mathbf{w}^{(0)}$ being a zero vector. $\tilde{\mathbf{w}}^{(t)}$ is the t-th solution before thresholding. $H(\mathbf{v}, s) : \mathbb{R}^d \to \mathbb{R}^d$ keeps only s elements with largest absolute values in $\mathbf{v}$

and set other elements to zero. $H(\mathbf{v}, s)$ equals the row selection operation as shown in Eq. (6). $J^{(t)} \subseteq [d]$ ($\left|J^{(t)}\right| = s$, $\forall t = 1, 2, \dots$) represents the set of indices of s remaining elements in $H(\tilde{\mathbf{w}}, s)$. $D_{J^{(t)}}$ is a diagonal matrix with $D_{j,j} = 1, \forall j \in J^{(t)}$ and 0 otherwise. The error vector of iteration rule (6) can then be written as

$$\begin{aligned}
\|\varepsilon^{(t)}\|^2 &= \|\mathbf{w}^{(t)} - \mathbf{w}^*\|^2 = \|D_{J^{(t)}}\tilde{\mathbf{w}}^{(t)} - \mathbf{w}^*\|^2 \\
&\geq \|D_{J^{(t)}}\tilde{\mathbf{w}}^{(t)} - D_{J^{(t)}}\mathbf{w}^*\|^2 \\
&= \|D_{J^{(t)}}[\mathbf{w}^{(t-1)} + \eta X^\top(y - X\mathbf{w}^{(t-1)})] \\
&\quad - D_{J^{(t)}}[\mathbf{w}^* + \eta X^\top(y - X\mathbf{w}^*)]\|^2 \\
&= \|D_{J^{(t)}}M(\mathbf{w}^{(t-1)} - \mathbf{w}^*)\|^2 \\
&= \|D_{J^{(t)}}M\varepsilon^{(t-1)}\|^2
\end{aligned} \tag{7}$$

Inequality (7) follows since w_j^*, $j \notin J^{(t)}$ are held out of the sum of the ℓ_2-norm. Therefore, $\|\varepsilon^{(t)}\|^2$ is lower-bounded by $\|D_{J^{(t)}}M\varepsilon^{(t-1)}\|^2$. For the ℓ_0 projected gradient descent to converge, $D_{J^{(t)}}M$ must satisfy $\rho(D_{J^{(t)}}M) < 1$, otherwise, update rule (6) diverges. Simple truncation using $H(\mathbf{v}, s)$ doesn't guarantee such condition, as we demonstrated in experiments.

2.2 Numerical Problems of Mirror Descent

The mirror descent algorithm (MDA) has been recognized as an effective online learning algorithm. For example, MDA using Bregman divergence is proposed to approximate the exponentiated gradient (EG) algorithm, which have cumulative loss bound logarithmically to the number of irrelevant features in the target weight vector [6]. In convergence rate, it is proved that MDA is superior to the usual stochastic gradient descent methods [12], where MDA is adopted in online sparse learning. However, the price MDA pays for these advantages is numerical unstability, especially when the dimensionality of data is high, leading to less discriminative models. As proposed in [12], the main component of MDA for sparse learning is the alternative updates of two vectors $\boldsymbol{\theta}$, $\mathbf{w} \in \mathbb{R}^n$ by the following rules:

$$w_j^{(t)} = f_j^{-1}(\boldsymbol{\theta}^{(t)}) = \frac{\operatorname{sgn}(\theta_j^{(t)})|\theta_j^{(t)}|^{p-1}}{\|\boldsymbol{\theta}^{(t)}\|_p^{p-2}}, \forall j \tag{8}$$

$$\boldsymbol{\theta}^{(t+1)} = S(\boldsymbol{\theta}^{(t)} - \eta\nabla L(\mathbf{w}^{(t)}), \lambda) \tag{9}$$

where $S(\cdot, \lambda)$ is the soft-thresholding operator [2]:

$$\begin{aligned}
S(w_j, \lambda) &= \operatorname{sgn}(w_j)(w_j - \lambda)_+ \\
&= \begin{cases} w_j - \lambda, \text{ if } w_j > 0 \text{ and } |w_j| > \lambda \\ w_j + \lambda, \text{ if } w_j < 0 \text{ and } |w_j| > \lambda \\ 0, \text{ if } |w_j| < \lambda \end{cases}
\end{aligned} \tag{10}$$

$S(\mathbf{v}, \lambda)$ means applying Eq. (10) at each element of $\mathbf{v}$. As suggested in [6], the parameter p in Eq. (8) is set to $O(\ln(d))$ (Note that different p lead to different "algorithms", for example, in binary classification, one obtain Perceptron when $p = 2$ and Weighted Majority as $p \to \infty$. However, for MDA to approximate the EG algorithm, p should be sufficiently large such as logarithmic p). $\mathbf{w}$ and $\boldsymbol{\theta}$ are called primal and dual vector, respectively. In the t-th iteration, MDA first computes $\mathbf{w}^{(t)}$ using $\mathbf{w}^{(t)} = f^{-1}(\boldsymbol{\theta}^{(t)})$, with $\boldsymbol{\theta}^{(0)} = \mathbf{0}$. Then the stochastic gradient of the loss function $\nabla L(\mathbf{w}^{(t)})$ is estimated using $(\mathbf{x}_i, y_i)$ at $\mathbf{w}^{(t)}$. Finally gradient descent and soft-thresholding update $\boldsymbol{\theta}^{(t)}$ to get $\boldsymbol{\theta}^{(t+1)}$. Converting $\boldsymbol{\theta}$ to $\mathbf{w}$ using Eq. (8) can cause numerical problem in MDA. Specifically, elements of $\mathbf{w}$ could become very small and sensitive to difference of θ_j's scale. The following lemma reveals the numerical problem that update rules (8) and (9) can bring.

Lemma 1. *Assume that the values of features and λ in Eq. (9) are in the scale of $O(1)$. At the t-th iteration of MDA minimizing logistic loss, where $t = O(p)$, $\theta_j^{(t)}$ is also in $O(p)$ and $w_j^{(t)}$ is at most in the order of $O(a^{-p})$ for some $a > 1$.*

Proof. Elements of the gradient of the logistic loss $\nabla_j L(\cdot,\cdot)$ with respect to the first argument is $L'(\langle \mathbf{w}, \mathbf{x}\rangle, y)\mathbf{x}$ and thus in the order of $O(1)$. $\theta_j^{(t)}$ is the summation of t terms in $O(1)$ and thus in the order of $O(p)$. Since $\|\boldsymbol{\theta}\|_p > \|\boldsymbol{\theta}\|_\infty$, $\exists \epsilon > 0$ s.t. $\|\boldsymbol{\theta}\|_p > (1+\epsilon)\|\boldsymbol{\theta}\|_\infty = (1+\epsilon)\max_i |\theta_i|$. By Eq. (8),

$$\begin{aligned}|w_j| &= \frac{|\theta_j|^{p-1}}{\|\boldsymbol{\theta}\|_p^{p-2}} < \frac{|\theta_j|^{p-1}}{(1+\epsilon)^{p-2}(\max_i |\theta_i|)^{p-2}} \\ &= |\theta_j| \left(\frac{|\theta_j|}{(1+\epsilon)\max_i |\theta_i|}\right)^{p-2} \\ &= O(p)O(a^{2-p}) = O(a^{-p})\end{aligned}$$

where $a = (1+\epsilon)\max_i |\theta_i| / |\theta_j| > 1$.

Particularly, because w_j is exponential in $p = O(\ln(d))$, small difference between exponents of dimensions could be greatly amplified in the resulting model. Consider two entries of $\boldsymbol{\theta}$: θ_1 and θ_2. Without loss of generality, assume that θ_1 is only one order smaller than θ_2 in magnitude: $|\theta_1| \approx 10^{-1}|\theta_2|$. Reconstruct the primal vector $\mathbf{w}$ from $\boldsymbol{\theta}$, we can see that w_1 is p order smaller than w_2 in magnitude : $|w_1| \approx 10^{-p}|w_2|$. When computing the inner product between $\mathbf{w}$ and $\mathbf{x}$, if the difference between exponents of w_1 and w_2 is larger than the precision that the data type supports, then the 2nd element in $\mathbf{x}$ would be totally lost. For example, double precision floating point number supported by C++ (64 bits in length according to IEEE 754-2008) typically has precision of 10^{-16}. Therefore $1.0 + 1.0^{-17} = 1.0$ in machine addition. In general, the larger p is, the more difference between w_j's exponents. We'll show in experiment (Section 4.3) how the parameter p affects the performance of MDA.

3 Efficient and Numerically Stable Sparse Learning

We present the proposed sparse learning algorithm in Section 3.1. In Section 3.2 we show that the method is guaranteed to converge, with numerical errors taken into account. Finally we show generalization error bound in Section 3.3.

3.1 The Proposed Method

Given the labeled data $Z = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)\}$, drawn from some distribution D over $\mathbb{R}^d \times \{-1, 1\}$, we consider learning a sparse large margin linear classifier. This reduces to minimizing certain loss function. There are three concerns. First, the algorithm should be scalable. One of the best practical methods known is online learning or stochastic gradient descent, which iteratively minimizes the loss function and keeps only the footprint of the sparse model without storing any example in memory. Second, the algorithm should be numerically stable. We prefer algorithms which rely on practical assumption and are robust to small difference in the scale between dimensions of data. Finally, for efficiency consideration, algorithms achieve better classification performance using less features are desirable.

Algorithm 1 Numerically Stable Sparse Learning

```
 1: **Input**: margin threshold $\tau$, learning rate $\eta$,
        regularization parameter $\lambda$, density upper bound $s$
 2: **initialize** linear model $\mathbf{w} = \mathbf{0}$, model density=0
 3: **while** model density less than $s$ **do**
 4:   Receive instance $(\mathbf{x}_i, y_i)$ and compute $z = y_i \mathbf{w} \mathbf{x}_i$
 5:   **if** $z \leq \tau$ **then**
 6:     **for** Each non-zero element $x_{ij}$ **do**
 7:       Update $w_j$ by $w_j = w_j + \eta x_{ij} y_i$
 8:       Soft threshold $w_j$ using Eq. (10)
 9:     **end for**
10:   **end if**
11: **end while**
```

We alter the perceptron algorithm such that it robustly produces sparse model with convergence and generalization error guarantees (see Section 3.2 and 3.3). The proposed method is described in Algorithm 1. It begins with a zero vector. During each iteration, it loads an example from secondary storage and updates the model as the original perceptron (line 4-7). The difference is that after each online update, the algorithm suppresses the weights in the model (line 8) using the soft-thresholding operator (Eq. (10)). Since we assume that the data is separable by a sparse linear classifier, by the fact that the data is in high dimensional space, it is expected that there would be many noisy or irrelevant features. For an example classified with margin higher than τ, it is more likely that the features in the current model $\mathbf{w}^{(t)}$ are sufficient to make correct decision, and the example is not used for update. This prevents unnecessarily introducing new features into the current model and maximally preserves the sparsity without increasing regret. In general, perceptron algorithm exploits the sparsity of the instance space, and further leads to a model with fewer features.

The above framework can be applied to very large datasets. The main task at each iteration is computing $\langle \mathbf{w}^{(t)}, \mathbf{x} \rangle$, which is in the form of summation and can be distributed to r machines. One can partition the dataset on a per-feature basis into r partitions, with each assigned to a machine. Each machine computes $O(\lceil d/r \rceil)$ components in $\langle \mathbf{w}^{(t)}, \mathbf{x} \rangle$. Adding these parts up obtains the inner product. There are $O(\lceil d/r \rceil)$ cost on each machine for I/O and $O(r)$ cost for communication. Thus the algorithm is scalable via parallelization on columns of the data matrix X.

3.2 Convergence of the Proposed Method

Without loss of generality, we assume that the learning rate equals 1 in the following analysis. We alter the convergence theorem in [10] and prove similar results of perceptron with soft-thresholding, and analyze conditions under which the soft-thresholding version perceptron converges. Assume that the t-th mistake happens on example $(\mathbf{x}, y)$. Let $J_0 = \{j | x_j = 0\}$ and $J_1 = \{j | x_j \neq 0\}$. Then $w_j^{(t)} = w_j^{(t-1)}$ for $j \in J_0$ and we can only pay attention to the difference between $w_j^{(t)}$ and $w_j^{(t-1)}$ for $j \in J_1$. By Algorithm 1 and Eq. (10),

$$\mathbf{w}^{(t)} = S(\tilde{\mathbf{w}}^{(t)}, \lambda) = S(\mathbf{w}^{(t-1)} + y\mathbf{x}, \lambda)$$

Project $\mathbf{w}^{(t-1)}$, $\mathbf{w}^{(t)}$ and $\mathbf{x}$ onto J_0 and J_1 respectively, and let the projection of a vector $\mathbf{v}$ denoted by the vector with the index set as its subscript (for example, $\mathbf{x}_{J_0}$ means keep the elements on indices in J_0 unchanged and set the other elements to zero). By simple algebra, $\mathbf{w}^{(t)}$ can be decomposed into the sum of two orthogonal vectors $\mathbf{w}^{(t)} = \mathbf{w}_{J_0}^{(t-1)} + \mathbf{w}_{J_1}^{(t)}$. We have the following lemma.

Lemma 2. *Assume that* $\forall(\mathbf{x}, y) \in Z$, $\|\mathbf{x}\| < R$. *Let* $J_+^{(t)}$ *denote the set* $\{j : |\tilde{w}_j^{(t)}| \geq \lambda\} \cap J_1$ *and* $J_-^{(t)}$ *be the set* $\{j : |\tilde{w}_j^{(t)}| < \lambda\} \cap J_1$. *Let* $\epsilon_\lambda^{(t)} = \lambda \sum_{j \in J_+^{(t)}} |\tilde{w}_j^{(t)}| + \sum_{j \in J_-^{(t)}} |\tilde{w}_j^{(t)}|^2$. *If* $\tau \leq \epsilon_\lambda^{(t)}/2$, $\|\mathbf{w}^{(t)}\|^2 \leq tR^2$.

Proof. Prove by induction,

$$\begin{aligned}
\|\mathbf{w}^{(t)}\|^2 &= \|\mathbf{w}_{J_0}^{(t-1)} + \mathbf{w}_{J_1}^{(t)}\|^2 = \|\mathbf{w}_{J_0}^{(t-1)}\|^2 + \|\mathbf{w}_{J_1}^{(t)}\|^2 \\
&\leq \|\mathbf{w}_{J_0}^{(t-1)}\|^2 + \|\tilde{\mathbf{w}}_{J_1}^{(t)}\|^2 - \epsilon_\lambda^{(t)} && (11) \\
&= \|\mathbf{w}_{J_0}^{(t-1)}\|^2 + \|\mathbf{w}_{J_1}^{(t-1)} + y\mathbf{x}\|^2 - \epsilon_\lambda^{(t)} && (12) \\
&\leq \|\mathbf{w}_{J_0}^{(t-1)}\|^2 - \epsilon_\lambda^{(t)} + \|\mathbf{w}_{J_1}^{(t-1)}\|^2 + \|y\mathbf{x}\|^2 + 2\tau && (13) \\
&\leq \|\mathbf{w}^{(t-1)}\|^2 + R^2 - \epsilon_\lambda^{(t)} + 2\tau && (14)
\end{aligned}$$

Inequality (11) follows from the fact that at each iteration t, $\tilde{\mathbf{w}}^{(t)}$ suffered from at least $\epsilon_\lambda^{(t)}$ of shrinkage. Inequality (13) holds because update occurs only when $\langle \mathbf{w}, y\mathbf{x} \rangle \leq \tau$. Since $\tau \leq \epsilon_\lambda^{(t)}/2$, then $\|\mathbf{w}^{(t)}\|^2 \leq \|\mathbf{w}^{(t-1)}\|^2 + R^2$.

Discussion We assume the condition $\tau \leq \epsilon_{\lambda}^{(t)}/2$ for the convenience of proof. If $\tau = 0$, then the condition is satisfied for $\forall\lambda > 0$. Particularly, if λ is set too small, then the algorithm asymptotically becomes Perceptron; By using a large λ, one can achieve a sparser model with sacrifice of accuracy. This is a more general problem of model selection and one need to trade off sparsity for accuracy[11]. In the experiment, we choose the parameters λ and τ via validation, then they're fixed during iterations.

We proved the above lemma without considering numerical errors. Let the machine precision be $eps = 2^{-r}$. Soft-thresholding introduces errors if λ and the truncated elements are rounded-off. This gives an error up to $|\varepsilon_1| \leq 2eps$ which is minor and can be ignored. Note that $\|\mathbf{w}^{(t)}\|$ usually grows with the iterations, thus more importantly, we must consider the numerical errors associated with this term. With $|\varepsilon_2| \leq eps$, the exact term $\|\mathbf{w}_{J_1}^{(t-1)} + y\mathbf{x}\|^2$ in Eq. (12) becomes $\|(1+\varepsilon_2)(\mathbf{w}_{J_1}^{(t-1)} + y\mathbf{x})\|^2$ and can be approximately upper-bounded by $(\|\mathbf{w}_{J_1}^{(t-1)}\|^2 + \|y\mathbf{x}\|^2 + 2\tau)(1+2\varepsilon_2)$ by ignoring the higher order term ε_2^2 (an exact upper-bound should be $(\|\mathbf{w}_{J_1}^{(t-1)}\|^2 + \|y\mathbf{x}\|^2 + 2\tau)(1+2\varepsilon_2+\varepsilon_2^2)$). Substitute this error into Eq. (12), we can prove the following theorem, which is the key to the convergence proof.

Theorem 1. *Given $\lambda \geq 0$, let $\epsilon_{\lambda}^{*} = \min_t \epsilon_{\lambda}^{(t)}$, then $\|\mathbf{w}^{(t)}\|^2 < tR^2$ holds, $\forall$ $t \in Z^+$ such that*

$$t \leq \frac{\epsilon_{\lambda}^{*} - 2\tau}{2\varepsilon_2 R^2} - \frac{2\tau}{R^2} \tag{15}$$

Proof. The inequality (14) becomes

$$\|\mathbf{w}^{(t-1)}\|^2 \underbrace{+R^2 - \epsilon_{\lambda}^{*} + 2\tau + 2\varepsilon_2(\|\mathbf{w}_{J_1}^{(t-1)}\|^2 + \|y\mathbf{x}\|^2 + 2\tau)}$$

As in the proof of Lemma 2, we need to prove that the under-braced sum is smaller than R^2. Since by induction, $\|\mathbf{w}_{J_1}^{(t-1)}\|^2 \leq \|\mathbf{w}^{(t-1)}\|^2 \leq (t-1)R^2$ and $\|y\mathbf{x}\|^2 \leq R^2$ by assumption, we need only to prove that

$$-\epsilon_{\lambda}^{*} + 2\tau + 2\varepsilon_2(tR^2 + 2\tau) \leq 0$$

Solving for t gets the conclusion.

In C++, $\varepsilon_2 \approx 10^{-16}$ in double-precision. If we set λ and τ such that $\epsilon_{\lambda}^{*} - 2\tau$ is not too small (for example, in the order of $o(R^2)$), then $\|\mathbf{w}^{(t)}\|^2 < tR^2$ holds for a sufficient large t in a reasonable training process.

Theorem 2. *Assume for all examples $\mathbf{x}_i \in Z$, $\|\mathbf{x}_i\| < R$. If there exists a linear model $\mathbf{u}$ such that $\|\mathbf{u}\| = 1$ and $0 < \epsilon_{\lambda,\gamma} < 1$ such that $\langle(y_i\mathbf{x}_i - \mathbf{v}), \mathbf{u}\rangle \geq (1-\epsilon_{\lambda,\gamma})\gamma$ for $\forall(\mathbf{x}, y) \in Z$ and any $\mathbf{v}$, $\|\mathbf{v}\|_\infty < \lambda$, then the number of mistakes made by the online perceptron algorithm on Z is at most $k = (R/(1-\epsilon_{\lambda,\gamma})\gamma)^2$.*

Proof. For any t satisfying Eq. (15),

$$\begin{aligned}\mathbf{w}^{(t)} \cdot \mathbf{u} &= \mathbf{w}^{(t-1)} \cdot \mathbf{u} + (\Delta^{(t)} + y_i \mathbf{x}_i) \cdot \mathbf{u} \\ &\geq \mathbf{w}^{(t-1)} \cdot \mathbf{u} + (1 - \epsilon_{\lambda,\gamma})\gamma\end{aligned} \tag{16}$$

where $\Delta^{(t)} = S(\tilde{\mathbf{w}}^{(t)}, \lambda) - \tilde{\mathbf{w}}^{(t)}$. Thus $\mathbf{w}^{(t)} \cdot \mathbf{u} \geq (1 - \epsilon_{\lambda,\gamma})t\gamma$. By Lemma (2), $\|\mathbf{w}^{(t)}\|^2 \leq tR^2$.

$$(1 - \epsilon_{\lambda,\gamma})t\gamma \leq \mathbf{w}^{(t)} \cdot \mathbf{u} \leq \|\mathbf{w}^{(t)}\| \leq \sqrt{t}R$$

This theorem indicates that if data in Z can be separated with margin at least γ, and this margin does not shrink too much compared with γ (up to $\epsilon_{\lambda,\gamma}$) given the examples are soft-thresholded, then the Algorithm 1 needs at most $(R/(1-\epsilon_{\lambda,\gamma})\gamma)^2$ examples among Z to learn a linear classifier consistent with all examples in Z. For the inseparable case, according to the method in [4], one can extend all the examples $\mathbf{x}$ to $\mathbf{x}'$ and the linear model $\mathbf{u}$ to $\mathbf{u}'$. In this extended space, $(\mathbf{x}', y)$ is linearly separable by $\mathbf{u}'$.

3.3 Generalization Error Bound of the Proposed Method

In Algorithm 1, features enter the model only when necessary. Such strategy also allow us to construct a consistent classifier using a subset of the training set. Combining these two properties, it is possible to learn a sparse model with generalization error guarantee. Algorithm 1 can be seen as a compression scheme [9] consisting of two mappings. The first mapping κ maps any training set Z of size m to $Z_k \subset Z$, called the "kernel" of Z, where k is the kernel size. The second mapping π uses Z_k to reconstruct the labels of all the examples in Z. κ can be seen as an algorithm, learning from m training examples and encoding the produced hypothesis h using only k of them. The mapping π requires that h is consistent with all the training examples. An algorithm with the data compression property is guaranteed with generalization error bound [9].

Lemma 3. *For any compression scheme with kernel size k, the probability that the generalization error of the learned hypothesis (with respect to distribution D) being larger than ϵ is less than $\binom{m}{k}(1-\epsilon)^{m-k}$*

With a smaller k where $k < \lfloor m/2 \rfloor$, the bound on generalization error becomes smaller. Combining Theorem 2 and Lemma 3, with high probability over the randomly drawn training set Z, the soft-thresholding perceptron algorithm can obtain a classifier $\mathbf{w}_Z$ of good generalizability using only k examples out of the total m training examples.

Theorem 3. *With probability at least $1-\delta$ over the random draw of the training set Z of size m, given the conditions in Theorem 2, the generalization error of the classifier found by the proposed algorithm is less than*

$$\frac{1}{m-k}\left(\ln\binom{m}{k} + \ln(m) + \ln\frac{1}{\delta}\right) \tag{17}$$

where $k = (R/(1-\epsilon_{\lambda,\gamma})\gamma)^2$ for some $0 < \epsilon_{\lambda,\gamma} < 1$.

Table 2: Description of classification tasks. Columns sequentially show the task ids, the tasks' names, the size of the training sets (#Tr), validation set (#Valid), test set (#Test) and number of total features (#Dim)

No.	Task	#Tr	#Valid	#Test	#Dim
1	comp_vs_rec	4278	1644	2948	21317
2	comp_vs_sci	4248	1766	2829	22944
3	comp_vs_talk	3960	1577	2607	24178
4	rec_vs_talk	3456	1423	2353	23253
5	sci_vs_rec	3772	1598	2561	22839
6	sci_vs_talk	3397	1445	2363	24777
7	rcv1	20242	3357	6854	47236
8	webspam	50000	4999	50000	16609143

Proof. The kernel size of the model learned from Z by the soft-thresholding algorithm is bounded by $k = (R/(1-\epsilon_{\lambda,\gamma})\gamma)^2$, for some $0 < \epsilon_{\lambda,\gamma} < 1$. By Lemma 3,

$$\binom{m}{k}(1-\epsilon)^{m-k} \leq \binom{m}{k} e^{(k-m)\epsilon} \tag{18}$$

Let the right hand size of the above inequality equal to δ and solve for ϵ, we obtain the conclusion.

This theorem is similar to the results given in [7] where the perceptron algorithm is run in the dual form and the output is a linear classifier in the kernel space. The difference is that the dual perceptron algorithm needs to store k training examples for prediction while the proposed method stores only a sparse vector.

4 Experiment

In this section, we present experiment results of the proposed and other methods, focusing on the numerical stability and efficiency of sparse models. After briefing the experiment settings, three subsections address the problems listed below:

i Does it converge when applying gradient descent directly on real-world data.
ii How numerical unstability affects MDA's convergence and model accuracy.
iii To what extent of sparsity can one achieve with performance guarantee.

4.1 Experiment Settings

We conducted experiments on three datasets. The first dataset is 20newsgroups, from which we construct six binary classification tasks. We used word vector representation with TFIDF weighting. The seventh and eighth tasks are constructed from rcv1 and webspam datasets, respectively. Note that the webspam dataset is from *Pascal Large Scale Learning Challenge*[3]. See Table 2 for details

[3] http://largescale.first.fraunhofer.de/instructions/

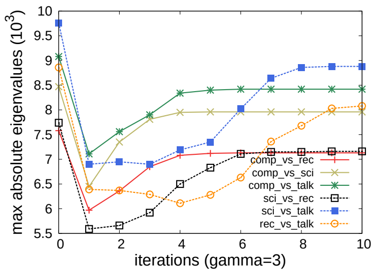

Fig. 1: Spectral Radius of Iteration Matrices of GraDes [5] ($\gamma = 3$)

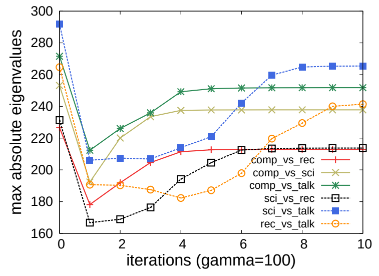

Fig. 2: Spectral Radius of Iteration Matrices of GraDes ($\gamma = 100$)

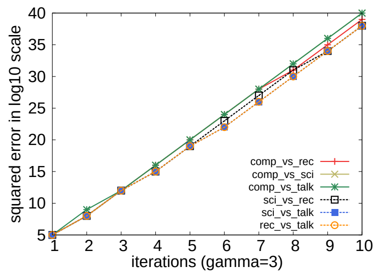

Fig. 3: Potential function value of GraDes of the first 10 iterations ($\gamma = 3$)

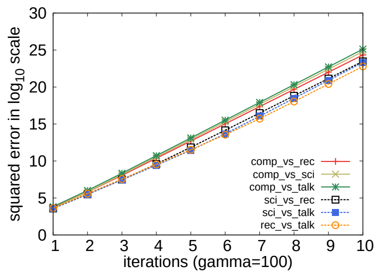

Fig. 4: Potential function value of GraDes of the first 10 iterations ($\gamma = 100$)

of tasks. We compared 4 baselines with the proposed method, focusing on numerical stability, model sparsity and accuracy. The baselines are: GraDes [5], SMIDAS (Stochastic Mirror Descent Algorithm made Sparse), SCD (Stochastic Coordinate Descent) [12] and TG(Truncated Gradient) [8]. We next demonstrate the numerical problems of GraDes and SMIDAS, shown in Section 2.1 and 2.2.

4.2 Numerical Problems of Direct Gradient Descent

The GraDes algorithm attempts to find a sparse solution for the least squared error regression problem using ℓ_0 regularization. The accuracy of GraDes is not directly comparable to the proposed method due to the difference of the loss functions (GraDes uses squared loss while other three use logistic loss in our experiments), therefore we report the numerical problem of GraDes without comparing to other algorithms. As we analyzed in Section 2.1, for the direct gradient descent method to converge, one should keep the spectral radius of the iteration matrix M less than 1. For GraDes, though by sparsification using hard thresholding function (Eq. 6), the iteration can still diverge.

In practice, it is unrealistic to verify that the spectral radius of M is less than 1. This is restricted by the space and time complexity (M is a dense matrix in size of $O(d^2)$). For instance, in one of the authors' machines with 2GB memory, MATLAB ran out of memory when computing the largest eigenvalue of a 20000×20000 matrix. It costs even more if we have to compute the spectral radius of

Table 3: Distribution and statistics of exponents

p in Eq. (8)	$p = 2\ln(d)$		$p = 0.5\ln(d)$	
range	within	mean±std	within	mean±std
comp_vs_rec	0.095	-80.3±23.6	1.000	-12.7±7.2
comp_vs_sci	0.258	-69.8±32.4	0.999	-15.4±7.1
comp_vs_talk	0.282	-78.6±43.1	0.996	-17.2±9.1
sci_vs_rec	0.381	-63.6±29.4	1.000	-14.3±6.1
sci_vs_talk	0.273	-69.3±26.9	1.000	-16.0±5.9
rec_vs_talk	0.347	-68.4±33.7	0.999	-15.8±7.5
rcv1	0.012	-136.8±32.6	0.998	-26.3±6.5
webspam	NA	NA	0.976	-71.4±22.1

the thresholded iteration matrix $D_{J^{(t)}}M$ to decide how to truncate $\tilde{\mathbf{w}}^{(t)}$ to $\mathbf{w}^{(t)}$. For the above 20newsgroup tasks, we reduce the number of features to around 5000 during text preprocessing while keeping the number of examples the same. Then we compute the top eigenvalues of the iteration matrices $D_{J^{(t)}}M$ at each iteration. $|J^{(t)}|$ is set to 2000 and the elements of $J^{(t)}$ are determined by the GraDes algorithm, $D_{J^{(0)}}$ is the identity matrix. According to [], γ is set to 3 and 100, respectively, Note that the learning rate is $1/\gamma$, thus we have two different learning rate settings. The spectral radii are plotted as a function of the number of iterations ,under two learning rates, as shown in Fig. 1 and Fig. 2.

One can easily observe that the spectral radius is far more than 1, indicating a fast divergence speed of the solution. For example, in Fig. 1, before the first iteration, the iteration matrix M has a spectral radius at least 7.5×10^3 (task "comp_vs_rec", shown in the red line with plus "+"). Although in the first iteration, the spectral radii are reduced by the hard thresholding function, they are still in the scale of 10^3. Moreover, the spectral radii increase as the algorithm proceeds. The radius of the iteration matrix for task "comp_vs_rec" increases from about 6×10^3 at the first iteration to about 7×10^3 at the 10-th iteration. All the other spectral radii increase to some extent and remain above 10^3. Fig. 2 shows similar situations, the spectral radii are over 160. The large spectral radii of the iteration matrices indicate that the solution should go beyond the optimal solution and therefore increase the potential function. To show this, for each iteration of GraDes, we compute the the value of $\Psi(\mathbf{w})$, which is expected to be reduced by the algorithm. However, the potential function values also go up quickly. In Fig. 3 and Fig. 4, for the 6 20newsgroups tasks, we plot $\Psi(\mathbf{w})$ in log_{10} scale at each GraDes iteration under two settings of γ. The scales climb up linearly, indicating an exponential increase of $\Psi(\mathbf{w})$.

4.3 Numerical Problem of Mirror Descent Algorithm

We show the evidence of the numerical problem leading to the low accuracy of MDA. Typical *IEEE 745* double precision floating numbers have at most 52 bits in the mantissa. If p in Eq. (8) is set too large, then small difference in exponents of θ_j would be greatly amplified in w_j. Adding two binary numerals with difference in exponent larger than 52 would cause the smaller one to be truncated. Usually, elements in one example of the data are normalized to be in the same scale, such as in the range of $[0, 1]$, the great disparity between exponents in the weights would cancel out parts of the data.

For each task, we trained a model with 40% of density (i.e. the percentage of non-zeros in the model, with 100% density we include all features in the model) using SMIDAS. p is set to $2\ln(d)$ and $0.5\ln(d)$, respectively. For the last task, the density is set to 0.1% due to the large number of features (over 16 millions), and $p \approx 33$ when $p = 2\ln(d)$, this will obviously cause data truncation when computing inner products. Thus we only report results when $p = 0.5\ln(d)$ for this task. We calculated the exponents of w_j in 2-based numeric. Denote the largest exponent by e_m. We showed in Table 3 the ratios of exponents falling within $[e_m - 51, e_m]$ (the column "within"). The columns "mean±std" show the means and standard deviations of exponents. As we can see, for $p = 2\ln(d)$, most of the w_j have their exponents out of $[e_m - 51, e_m]$. During prediction, the corresponding features of these elements in the data are totally truncated, therefore the models SMIDAS produce lack of sufficient discriminability and lead to poor performance (black lines with empty squares in Fig. (5)). The standard deviations are large, indicating a wide dynamic range of exponents. For $p = 0.5\ln(d)$, SMIDAS approximates the normal gradient descent [6] and produces more reasonable models. Most of the exponents are in the range of $[e_m - 51, e_m]$ and the dynamic ranges become much smaller. The performances go up dramatically, but still inferior to the proposed method (blue lines with filled squares in Fig. (5)).

4.4 Sparsity and Performance Comparison

We focus on how the performance varies as a function of sparsity, and how numerical problem affects the MDA. The proposed method is compared with three scalable algorithms, SMIDAS, TG and SCD, all use logistic loss function. We randomly split all labeled data of each task into training, test and validation set (see Table 2). The training sets are further randomly shuffled (for task 1-7) or split (task 8) to create 10 copies of training data. For the proposed algorithm and TG, for each parameter setting, we trained 10 models using these 10 copies and calculated the average performance of the models on validation set. For SMIDAS and SCD, since they randomly shuffle the examples and features respectively, only one copy of training data of each task is needed. We chose the best parameters according to this average performance. Finally we reported the best models' averaged accuracy on test sets. All algorithms require a regularization parameter λ to control the intensity of sparsification. With a larger λ, we expect more sparsity of the model. We tuned λ using 0.0001, 0.0005, 0.001, 0.01 and 0.1. For learning rate η needed in SMIDAS, TG and the proposed method, we varied it from 0.1 to 0.5 with step size 0.1. The proposed method requires one more parameter τ determining the minimum margin above which to skip one instance. τ was searched using 0.001, 0.01 and 0.1.

To compare the performance with sparsity restriction, we require the percentage of the non-zeros in the model (model density) to be at most 40% for task 1-7 and 0.1% for task 8. We recorded the error rates when the density reach pre-specified percentages. For all algorithms, we stopped running the program when they reach density upper bounds, or the maximal scans of training set (10 scans for all tasks). These error rates of each algorithms in 8 tasks are depicted in Fig. 5. In general, the proposed algorithm achieves lower error rate using fewer features compared with the other three. Particularly, in task 2, 7 and 8, though the density of the proposed algorithm went up to 40%, the errors are consistently lower than other methods. In the rest 5 tasks, the proposed method stop updating the model before the model is too dense, even when the maximum number of scans is reached. These not only demonstrated the convergence and generalizability of the proposed algorithm (see Section 3.2 and 3.3), but also the sparsity of the models it produces. Specifically, in task 1, the proposed algorithm converges before the density reaches 30%, with approximately 5% of error rate, which outperformed the remaining algorithms, even they used 40% of features. In task 3, the proposed method achieved 5% of error rate at density 20% which can only be obtained when SCD reached 40% of density, with other two have their error rates higher than that of SCD. Note that in task 1-7 SMIDAS with $p = 2\ln(d)$ was beat by all other 3 methods and itself with $p = 0.5\ln(d)$. This showed that SMIDAS is not applicable for approximating the EG algorithm, especially when the dimensionality is high.

In task 1-6, when d and m differ by at most one order, TG converges slower than SCD. SCD adds features according to global information provided by all training examples, while TG works based on stochastic gradient. This is inaccurate compared with SCD. However, in task 8, when d are several order larger than m and sparser model are required, SCD fails to converge before reaching 40% of density. TG converges since it exploits more features than SCD. The proposed method takes the best of both. For any training example, it is always possible to reduce the loss following the gradient direction. As long as the current example does not increase the regret of online learning, we simply hold it out of the model. In this way, the proposed method achieves better generalization ability with a higher degree of sparsity.

5 Conclusions

We addressed several problems in existing sparse learning methods. Though RIP provides theoretical guarantee to recover the sparse solutions, it is mostly satisfied by designed matrices in compressive sensing, rather than data collected in the real world. Failing to meet RIP can cause matrix iteration-based gradient descent to diverge [5], while ℓ_0 constraint doesn't solve the problem. Second, though MDA using Bregman divergence enjoys fast convergence and approximates the EG algorithm, its update rules produce inaccurate models when dimensionality is high. Lastly, ℓ_1 regularization is insufficient for finding an sparser solution, in several existing methods, density of model grows faster than necessary.

We have proposed to combine the perceptron algorithm with soft-thresholding. The algorithm converges with numerical error taken into account. We have also provided generalization error bound of the algorithm. Finally, the algorithm is highly scalable, data access can be distributed on a per-feature basis, making it more suitable for real-world applications. Experiments have shown that the proposed method outperformed 4 existing sparse learning algorithms in numerical stability, accuracy and model sparsity control. In one task with approximately 3.5GB of training data (16 million features, 50k examples), the proposed approach achieved 5.9% error rate using model with 0.8% density. But the best competing approach (TG for this dataset) can only get 8.2% error rate, using model with 0.7% density.

References

1. E. J. Candes and M. B. Wakin. An introduction to compressive sampling. *IEEE Signal Processing Magazine*, 25(2):21–30, March 2008.
2. David Donoho and Iain M. Johnstone. Adapting to unknown smoothness via wavelet shrinkage. *Journal of the American Statistical Association*, 90:1200–1224, 1995.
3. John Duchi and Yoram Singer. Boosting with structural sparsity. In *ICML*, page 38, 2009.
4. Yoav Freund and Robert E. Schapire. Large margin classification using the perceptron algorithm. In *Machine Learning*, pages 277–296, 1998.
5. Rahul Garg and Rohit Khandekar. Gradient descent with sparsification: an iterative algorithm for sparse recovery with restricted isometry property. In *ICML*, pages 337–344, 2009.
6. Claudio Gentile and Nick Littlestone. The robustness of the p-norm algorithms. In *Proceeding of 12th Annual Conference on Computer Learning Theory*, pages 1–11. ACM Press, New York, NY, 1999.
7. Thore Graepel and Ralf Herbrich. From margin to sparsity. In *In Advances in Neural Information Processing Systems 13*, pages 210–216. MIT Press, 2001.
8. John Langford, Lihong Li, and Tong Zhang. Sparse online learning via truncated gradient. *Journal of Machine Learning Research*, 10:777–801, 2009.
9. N. Littlestone and M. Warmuth. Relating data compression and learnability, 1986.
10. Albert B. Novikoff. On convergence proofs for perceptrons. In *Proceedings of the Symposium on the Mathematical Theory of Automata*, volume 12, pages 615–622, 1963.
11. Shai Shalev-Shwartz, Nathan Srebro, and Tong Zhang. Trading accuracy for sparsity. Technical report, Toyota Technological Institute at Chicago, 2009.
12. Shai S. Shwartz and Ambuj Tewari. Stochastic methods for ℓ_1 regularized loss minimization. In *ICML*, pages 929–936. ACM, 2009.
13. Tong Zhang. Adaptive forward-backward greedy algorithm for sparse learning with linear models. In *NIPS*, pages 1921–1928, 2008.
14. Peng Zhao and Bin Yu. On model selection consistency of lasso. *Journal of Machine Learning Research*, 7:2541–2563, 2006.

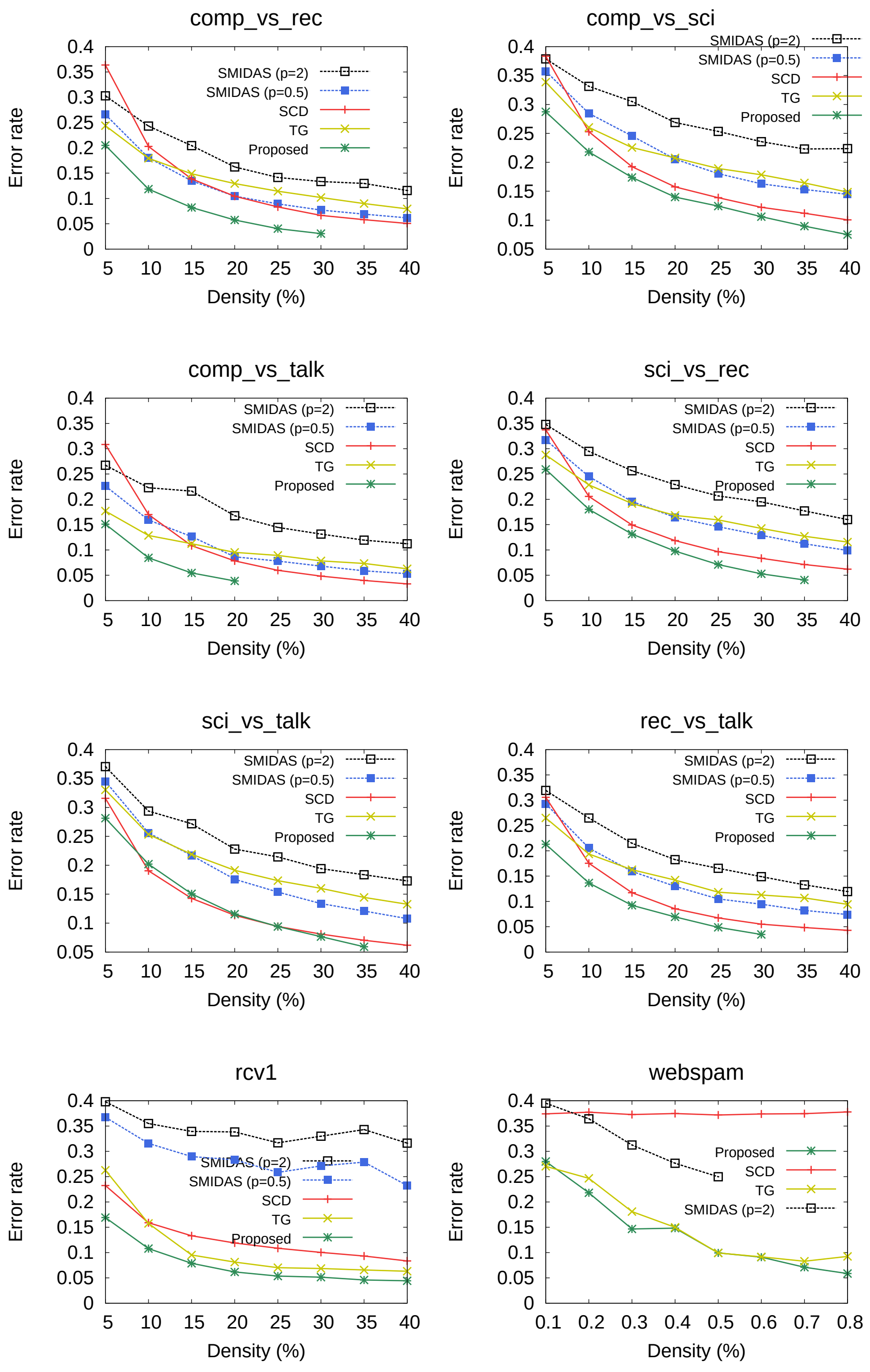

Fig. 5: Comparison of accuracy