Robustness Can Be Cheap: A Highly Efficient Approach to Discover Outliers under High Outlier Ratios

Siqi Wang,[1,2] **En Zhu,**[1] **Xiping Hu,**[2] **Xinwang Liu,**[1] **Qiang Liu,**[1] **Jianping Yin,**[3] **Fei Wang**[4]
[1]College of Computer, National University of Defense Technology, Changsha, China
[2]Shenzhen Institute of Advanced Technology (SIAT), Chinese Academy of Science, Shenzhen, China
[3]Dongguan University of Technology, Dongguan, 523808, Guangdong, China
[4]Healthcare Policy and Research, Weill Cornell Medical School, Cornell University, New York, NY 10065, USA

Abstract

Efficient detection of outliers from massive data with a *high outlier ratio* is challenging but not explicitly discussed yet. In such a case, existing methods either suffer from *poor robustness* or require *expensive computations*. This paper proposes a Low-rank based Efficient Outlier Detection (LEOD) framework to achieve favorable robustness against high outlier ratios with much cheaper computations. Specifically, it is worth highlighting the following aspects of LEOD: (1) Our framework exploits the low-rank structure embedded in the similarity matrix and considers inliers/outliers equally based on this low-rank structure, which facilitates us to encourage satisfying robustness with low computational cost later; (2) A novel re-weighting algorithm is derived as a new general solution to the constrained eigenvalue problem, which is a major bottleneck for the optimization process. Instead of the high space and time complexity ($O((2n)^2)$/$O((2n)^3)$) required by the classic solution, our algorithm enjoys $O(n)$ space complexity and a faster optimization speed in the experiments; (3) A new alternative formulation is proposed for further acceleration of the solution process, where a cheap closed-form solution can be obtained. Experiments show that LEOD achieves strong robustness under an outlier ratio from 20% to 60%, while it is at most 100 times more memory efficient and 1000 times faster than its previous counterpart that attains comparable performance. The codes of LEOD are publicly available at `https://github.com/demonzyj56/LEOD`.

Introduction

Outlier detection aims to identify the unusual patterns that do not conform to frequently-seen behaviors in a data set. As outliers cannot be defined by any specific class, classic methods are usually based on the "few and different" assumption (Chandola and Kumar 2007): (1) "Few": The ratio of outliers is much lower than inliers. (2) "Different": Outliers evidently differ from inliers. However, this commonly-used assumption will fail when the proportion of outliers is large, i.e. a *high outlier ratio*. In fact, a high outlier ratio is frequently seen in many real-life scenarios since *outliers may originate from a variety of different underlying causes and account for a high overall outlier ratio*. For example, online reviews of Yelp contain up to 30% outlying faked reviews from various sources (Luca and Zervas 2016); Images

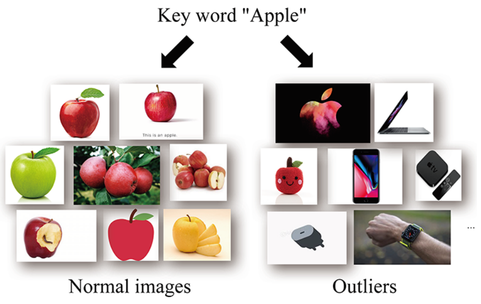
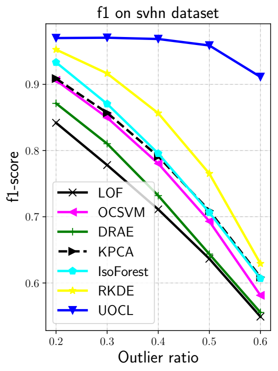


Figure 1: Top images returned by Google on "apple".

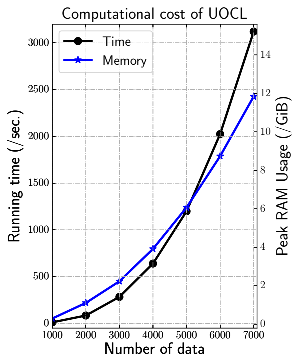
crawled from Internet are often mixed with massive outliers from a host of unknown classes, which may occupy nearly 50% of the top search results (see Fig. 1). As handling a high outlier ratio is vital for practical applications but has not been specifically discussed by prior literature, this paper will focus on this problem and formally formulate it later.

Under a high outlier ratio, existing methods are faced with some important challenges: As traditional outlier detection methods often explicitly or implicitly assume outliers to be minority, they suffer from *poor robustness*, which refers to a very high vulnerability to the high outlier ratio, so their performance will be severely degraded when the outlier ratio gets higher (see Fig. 2a). In the meantime, despite that our experiments show that the recent method Unsupervised One-class Learning (UOCL) (Liu, Hua, and Smith 2014) obtains satisfying robustness under high outlier ratios since it no longer assumes "few" outliers in the first place (blue line in Fig. 2a), it requires intolerably *expensive computations* (see Fig. 2b): It costs 50 minutes and 12 GiB memory to process merely $n = 7000$ 128-dimension data on a benchmark dataset. What is worse, the growth rate of its memory/time cost is quadratic/cubic. Such fatal flaw in efficiency makes it only applicable to small-scale problems. Thus, a solution that can preserve good outlier detection performance under a high outlier ratio ("*robust*") but require low memory and time cost ("*cheap*") remains open to be explored.

To address the aforementioned challenges, this paper proposes Low-rank based Efficient Outlier Detection (LEOD). Specifically, our approach (1) introduces a novel framework that utilizes the low-rank structure embedded in the simi-



(a)



(b)

Figure 2: Existing methods suffer from poor robustness or require expensive computations under high outlier ratios.

larity matrix and evaluates outliers/inliers equally based on such low-rank structure, which lays the foundation to preserve good robustness under high outlier ratios with much cheaper computations; (2) develops a novel re-weighting algorithm to solve the constrained eigenvalue problem during optimization. Its classic solution needs $O((2n)^2)$ space and $O((2n)^3)$ time complexity, while our algorithm enjoys $O(n)$ space complexity and tens of times speedup in experiments; (3) proposes a even faster alternative, which yields a highly efficient closed-form solution and a significantly faster optimization speed. Experiments on benchmarks demonstrate that LEOD possesses strong robustness against high outlier ratios (20%-60%), while at most 100 times more memory efficient and at most 1000 times faster than the previous method that reaches the same level of robustness.

Related Work

A closely related problem to outlier detection is *novelty* or *anomaly detection* (Japkowicz, Myers, and Gluck 1995; Gupta and Ghosh 2005; Désir et al. 2013), which requires pure training data from a single target class to fit a profile to detect deviated data. By contrast, *outlier detection* handles completely unlabeled data and detects outliers by their intrinsic properties, e.g. local data density (Breunig et al. 2000), cluster membership (Aggarwal, Zhao, and Yu 2011), pair-wise distance (Ramaswamy, Rastogi, and Shim 2000), probabilistic density (Parzen 1962; Kim and Scott 2012), reconstruction error distribution (Schölkopf, Smola, and Müller 1997; Xia et al. 2015) or path length of random binary trees (Liu, Ting, and Zhou 2008; 2010). Besides, outliers can be automatically detected by optimizing a soft decision boundary of hyper-plane/sphere (Schölkopf et al. 2001; Tax and Duin 2004). However, as we discussed above, they usually suffer from significant performance loss under a high outlier ratio. Notably, although it does not specifically formulate and study the case of high outlier ratios, the recent work (Liu, Hua, and Smith 2014) provides a promising formulation using margin maximization and manifold regularization, which no longer assumes outliers to be minority and is found to be robust under high outlier ratios. Unfortunately, it requires quadratic space and cubic time complexity for optimization. In addition, another related topic is *low-rank analysis*, which is rarely considered in outlier detection. The most relevant works to our knowledge are (Li, Shao, and Fu 2014a; 2014b), which embed a low-rank constraint into the classic one-class support vector machine or support vector data description to improve their performance. Besides, (Li, Shao, and Fu 2018) conduct cross-view low-rank analysis to perform multi-view outlier detection.

The Proposed Algorithm

Problem formulation: A high outlier ratio

We begin with a formal discussion on data with a high outlier ratio: Given an unlabeled data collection $\mathcal{X} = \{\mathbf{x}_i \in \mathbb{R}^d\}_{i=1}^n$, its data originate from q underlying classes (or data distributions) $\{\mathcal{Y}_1, \cdots \mathcal{Y}_q\}$ $(2 \leq q < n)$. In particular, the underlying classes $\{\mathcal{Y}_1, \cdots \mathcal{Y}_q\}$ are "compact", which means data have very large similarity within a class and very small similarity between classes. Denoting the ratio of data from class $\mathcal{Y}_i$ by $|\mathcal{Y}_i|$, $\mathbf{x} \in \mathcal{Y}_i$ is discriminated as an outlier if $|\mathcal{Y}_i|$ is small: When $|\mathcal{Y}_i|$ is small, $\mathbf{x}$ rarely appears in $\mathcal{X}$ and it evidently differs from data from other classes ($\mathcal{Y}_i$ is compact). In this way, we simply assume $|\mathcal{Y}_1| \geq |\mathcal{Y}_2| \cdots \geq |\mathcal{Y}_q|$ and data from the last q' classes $\{\mathcal{Y}_{q-q'+1}, \cdots \mathcal{Y}_q\}$ are outliers, $1 \leq q' < q$. When q' is small (e.g. $q' = 1$), the overall outlier ratio $\rho = \sum_{i=q-q'+1}^{q} |\mathcal{Y}_i|$ is low, which is the case of the "few and different" assumption. However, ρ can be very high when q' is large, e.g. suppose $|\mathcal{Y}_1| = 50\%$, $q' = 50$ and $|\mathcal{Y}_2| = |\mathcal{Y}_3| \cdots = |\mathcal{Y}_{51}| = 1\%$, data from $\mathcal{Y}_2, \cdots \mathcal{Y}_{51}$ should be viewed as outliers since their ratio is significantly lower than $\mathcal{Y}_1$, but the outlier ratio is very high ($\rho = 50\%$).

Low-rank based outlier detection framework

Motivation Under a high outlier ratio, outliers cannot be assumed to be "few", so we can only rely on "different", i.e. the pair-wise similarity of data to discriminate outliers. Thus, it is often necessary to compute a similarity matrix $\mathbf{K}$ with its element $\mathbf{K}_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$, where K is a function that returns a similarity measure for data $\mathbf{x}_i$ and $\mathbf{x}_j$. Since storing and computing with $\mathbf{K}$ usually induce $O(n^2)$ space and $O(n^3)$ time complexity, which are very expensive with large n, it naturally motivates us to observe the structure of $\mathbf{K}$ for outlier detection: We still assume $|\mathcal{Y}_1| \geq |\mathcal{Y}_2| \cdots \geq |\mathcal{Y}_q|$ and the last q' classes are outliers. As $\{\mathcal{Y}_1, \cdots \mathcal{Y}_q\}$ are compact, in the ideal case we have $K(\mathbf{x}_i, \mathbf{x}_j) \approx 1$ if $\mathbf{x}_i$ and $\mathbf{x}_j$ are from the same class, otherwise $K(\mathbf{x}_i, \mathbf{x}_j) \approx 0$. Thus, $\mathbf{K}$ can be rewritten into a block-diagonal matrix, which can be divided into the inlier part $\mathbf{K}_{in}$ and outlier part $\mathbf{K}_{out}$:

$$\mathbf{K} = \begin{bmatrix} \mathbf{1}_{n_1 \times n_1} & & & \\ & \mathbf{1}_{n_2 \times n_2} & & \\ & & \ddots & \\ & & & \mathbf{1}_{n_q \times n_q} \end{bmatrix} \triangleq \begin{bmatrix} \mathbf{K}_{in} \\ \mathbf{K}_{out} \end{bmatrix} \quad (1)$$

where $n_i = |\mathcal{Y}_i| \times n$ is the number of data from $\mathcal{Y}_i$ and $\mathbf{1}_{a \times b}$ denotes an $a \times b$ all-1 matrix with its rank $r(\mathbf{1}_{a \times b}) = 1$.

As inliers are from class $\mathcal{Y}_i$ with $n_i \gg 1$ $(1 \leq i \leq q - q')$, $\mathbf{K}_{in}$'s rank $r(\mathbf{K}_{in}) = \sum_{i=1}^{q-q'} r(\mathbf{1}_{n_i \times n_i}) \ll \sum_{i=1}^{q-q'} n_i$, which indicates that $\mathbf{K}_{in}$ has a clear low-rank structure. Compared with inliers, outliers under high outlier ratios often originates from a much larger number of classes ($q' \gg q-q'$) and n_i of each outlier class are much smaller (in many cases $n_i = 1$ for outliers), so its rank $r(\mathbf{K}_{out}) \gg r(\mathbf{K}_{in})$. Meanwhile, the eigenvalues of $\mathbf{K}$ are $n_1, n_2, \cdots n_q$, which exactly correspond to each all-1 block $\mathbf{1}_{n_i \times n_i}$. Since for outlier detection we actually do not care about the pair-wise similarity of outliers, which accounts for the high-rank structure in $\mathbf{K}$, we are naturally inspired to exploit the low-rank structure in $\mathbf{K}$ to obtain a more efficient representation of inliers' pair-wise similarity. This representation can be obtained by low-rank matrix approximation (Williams and Seeger 2001), which computes a rank-l ($l \ll n$) approximation that preserves only the low-rank structure that corresponds to large eigenvalues of $\mathbf{K}$. Specifically, as similarity matrix $\mathbf{K}$ is usually positive semi-definite (PSD), we compute the rank-l approximation of $\mathbf{K}$ efficiently by Randomized Nyström approximation (Li, Kwok, and Lü 2010) with $O(nml + l^3)$ time complexity (m is the number of columns that need to be randomly sampled from $\mathbf{K}$ and $l \leq m \ll n$):

$$\mathbf{K}_{n\times n} \approx (\mathbf{U}_K)_{n\times l}(\mathbf{D}_K)_{l\times l}(\mathbf{U}_K^\top)_{l\times n} \tag{2}$$

where $\mathbf{D}_K$ is a low-rank diagonal matrix. Since computing (2) does not require building the complete $\mathbf{K}$, the space complexity is reduced from $O(n^2)$ to $O(nl + l^2)$, which facilitates us to discover outliers in an effective and efficient manner later. Other recent low-rank approximation methods are applicable to this framework as well.

General formulation As we solely rely on pair-wise similarity under a high outlier ratio, we discover outliers by a classifier $f(\mathbf{x}) = \sum_{j=1}^{n} \alpha_j K(\mathbf{x}, \mathbf{x}_j)$ based on the representer theorem (Scholkopf, Herbrich, and Smola 2001). With the low-rank based similarity representation in (2), the scores of data in $\mathcal{X}$ can be computed in an efficient form:

$$\mathbf{f} = [f(\mathbf{x}_1), f(\mathbf{x}_2), \cdots f(\mathbf{x}_n)]^\top = \mathbf{U}_K\mathbf{D}_K(\mathbf{U}_K^\top\boldsymbol{\alpha}) \tag{3}$$

where $\boldsymbol{\alpha} = [\alpha_1, \cdots \alpha_n]^\top$. Note that the brackets in (3) is the proper matrix multiplication order for cheaper computations: Computing $\mathbf{f}$ by $\mathbf{U}_K\mathbf{D}_K(\mathbf{U}_K^\top\boldsymbol{\alpha})$ only involves $n \times l$, $l \times l$ and $n \times 1$ matrix, but $(\mathbf{U}_K\mathbf{D}_K\mathbf{U}_K^\top)\boldsymbol{\alpha}$ will generate $n \times n$ intermediate matrix. In this way, we can lower the cost of all intermediate computations associated with $\mathbf{U}_K\mathbf{D}_K\mathbf{U}_K^\top$. To handle high outlier ratios without priorly assuming outliers to be few, we are inspired by (Liu, Hua, and Smith 2014) and consider a basic formulation that jointly learns a similarity based classifier f and a soft label assignment $\mathbf{y} = [y_1, y_2, \cdots y_n]^\top$, which evaluates each datum fairly by its pair-wise similarity to other data in a self-guided manner:

$$\begin{aligned} \min_{\boldsymbol{\alpha}\in\mathcal{D},\mathbf{y}} \quad & \sum_{i=1}^{n}(f(\mathbf{x}_i) - y_i)^2 + \gamma_1\|f\|_{\mathcal{M}}^2 - \frac{2\gamma_2}{n^+}\sum_{i,y_i>0} f(\mathbf{x}_i) \\ s.t. \quad & y_i \in \{c^+, c^-\},\ 0 < n^+ < n \end{aligned} \tag{4}$$

In Eq. (4), $\gamma_1, \gamma_2 > 0$ are two trade-off hyperparameters. n^+ denotes the number of assigned positive labels in $\mathbf{y}$. $\|f\|_{\mathcal{M}}^2 = \mathbf{f}^\top\mathbf{L}\mathbf{f}$ is a manifold regularizer, where $\mathbf{L}$ is the Laplacian matrix. Soft labels (c^+, c^-) are set to be $(\sqrt{(n-n^+)/n^+}, -\sqrt{n^+/(n-n^+)})$ so as to guarantee $\|\mathbf{y}\|_2^2$ is constant and avoid a trivial solution. The weights $\boldsymbol{\alpha}$ of classifier f is constrained into the domain $\mathcal{D}$ to avoid overly large $\boldsymbol{\alpha}$. Each term of Eq. (4) serves a specific purpose: The first term $\sum(f(\mathbf{x}_i) - y_i)^2$ enforces the classifier $f(\mathbf{x}_\mathbf{i})$ to output a close label to the assigned label y_i; The manifold regularizer $\|f\|_{\mathcal{M}}^2$ encourages neighboring data to share a consistent label assignment. The last term maximizes the margin of those data assigned with positive label $y_i = c^+$. The objective in (4) can be simplified as follows:

$$\begin{aligned} \min_{\boldsymbol{\alpha}\in\mathcal{D},\tilde{\mathbf{y}}} \quad & \mathbf{f}^\top(\mathbf{I}_n + \gamma_1\mathbf{L})\mathbf{f} - 2\tilde{\mathbf{y}}^\top\mathbf{f} \\ s.t. \quad & \tilde{y}_i \in \{c^+ + \frac{\gamma_2}{n^+}, c^-\},\ 0 < n^+ < n \end{aligned} \tag{5}$$

where $\tilde{\mathbf{y}} = [\tilde{y}_1, \cdots \tilde{y}_n]^\top$ and $\mathbf{I}_n$ is a $n \times n$ identity matrix. However, despite that we have already provided an efficient way to compute output scores $\mathbf{f}$ by Eq. (3), formulation (5) is still inefficient to be adopted because it involves the $n \times n$ matrix $(\mathbf{I}_n + \gamma_1\mathbf{L})$. Since $(\mathbf{I}_n + \gamma_1\mathbf{L})$ is also a PSD matrix, a straightforward solution is to compute the low-rank approximation of $(\mathbf{I}_n + \gamma_1\mathbf{L})$, but it is not a good solution since our later experiments show that it will severely undermine the performance (see Fig. 5a-5b). Instead of this naive solution, the low-rank structure in Eq. (3) enables us to kill two birds with one stone and yield a new low-rank based general formulation: As Laplacian matrix $\mathbf{L} = diag(\mathbf{K}\mathbf{1}_{n\times 1}) - \mathbf{K}$, we notice that the similarity matrix $\mathbf{K}$ can be substituted by its sparsified form $\mathbf{W}$, i.e. if the set of $\mathbf{x}$'s k-nearest neighbors is $knn(\mathbf{x})$, we set $\mathbf{W}_{ij} = \mathbf{K}_{ij}$ if $\mathbf{x}_i \in knn(\mathbf{x}_j)$ or $\mathbf{x}_j \in knn(\mathbf{x}_i)$, otherwise $\mathbf{W}_{ij} = 0$. After sparsification, $\mathbf{W}$ has at most $2nk$ non-zero elements (k is small and $k < l$), which are cheap for storage and computation. Therefore, we can represent $\mathbf{W}$ by a more efficient sparse form:

$$\mathbf{W} = [\{\mathbf{v}_1, \mathcal{I}_1\}, \{\mathbf{v}_2, \mathcal{I}_2\}, \cdots, \{\mathbf{v}_n, \mathcal{I}_n\}] \tag{6}$$

where $\mathbf{v}_i$ is a column vector formed by those non-zero elements of $\mathbf{W}$'s i_{th} column, and $\mathcal{I}_i$ is the corresponding indexes. Since $\mathbf{L} = diag(\mathbf{W}\mathbf{1}_{n\times 1}) - \mathbf{W}$ and $\mathbf{I}$ is a sparse diagonal matrix, $(\mathbf{I} + \gamma_1\mathbf{L})$ has a similar sparse form to $\mathbf{W}$:

$$\mathbf{I} + \gamma_1\mathbf{L} = [\{\mathbf{v}'_1, \mathcal{I}'_1\}, \{\mathbf{v}'_2, \mathcal{I}'_2\}, \cdots, \{\mathbf{v}'_n, \mathcal{I}'_n\}] \tag{7}$$

As Eq. (3) gives $\mathbf{f} = \mathbf{U}_K\mathbf{D}_K(\mathbf{U}_K^\top\boldsymbol{\alpha})$, we can compute a $l \times l$ matrix $\mathbf{M} \triangleq \mathbf{U}_K^\top(\mathbf{I} + \gamma_1\mathbf{L})\mathbf{U}_K$ efficiently by:

$$\mathbf{M} = ([\mathbf{U}_{\mathcal{I}'_1}\mathbf{v}'_1, \mathbf{U}_{\mathcal{I}'_2}\mathbf{v}'_2, \cdots, \mathbf{U}_{\mathcal{I}'_n}\mathbf{v}'_n])_{l\times n} \cdot (\mathbf{U}_K)_{n\times l} \tag{8}$$

where $\mathbf{U}_{\mathcal{I}'_i}$ is the matrix built by concatenating those selected columns of $\mathbf{U}_K^\top$ using indexes $\mathcal{I}'_i$. Thus, we have:

$$\mathbf{f}^\top(\mathbf{I}_n + \gamma_1\mathbf{L})\mathbf{f} = \boldsymbol{\alpha}^\top(\mathbf{U}_K\mathbf{D}_K)\mathbf{M}(\mathbf{U}_K\mathbf{D}_K)^\top\boldsymbol{\alpha} \tag{9}$$

By $\mathbf{U}_K\mathbf{D}_K \triangleq \mathbf{U}_T$, we obtain a new general formulation:

$$\begin{aligned} \min_{\boldsymbol{\alpha}\in\mathcal{D},\tilde{\mathbf{y}}} \quad & \boldsymbol{\alpha}^\top\mathbf{U}_T\mathbf{M}\mathbf{U}_T^\top\boldsymbol{\alpha} - 2\tilde{\mathbf{y}}^\top\mathbf{U}_K\mathbf{D}_K\mathbf{U}_K^\top\boldsymbol{\alpha} \\ s.t. \quad & \tilde{y}_i \in \{c^+ + \frac{\gamma_2}{n^+}, c^-\},\ 0 < n^+ < n \end{aligned} \tag{10}$$

Algorithm 1 Re-weighting Algorithm

```
1: Input: $\mathbf{U}_T$, $\mathbf{M}$, $\tilde{\mathbf{y}}$.
2: Output: $\boldsymbol{\alpha}$.
3: Randomly initialize $\boldsymbol{\alpha}^{(0)}$ that satisfies $\|\boldsymbol{\alpha}^{(0)}\|_2 = 1$ and $t = 1$.
4: Calculate $\beta = MaxEigenValue(\mathbf{U}_K\mathbf{D}_K\mathbf{U}_K^\top)$.
5: $\mathbf{b} = \mathbf{U}_K\mathbf{D}_K(\mathbf{U}_K^\top\tilde{\mathbf{y}})$.
6: repeat
7:     Calculate $\mathbf{u} = \beta\boldsymbol{\alpha}^{(t-1)} - \mathbf{U}_T\mathbf{M}(\mathbf{U}_T^\top\boldsymbol{\alpha}^{(t-1)}) + \mathbf{b}$.
8:     Update $\boldsymbol{\alpha}^{(t)} = \frac{\mathbf{u}}{\|\mathbf{u}\|_2}$.
9:     $t = t + 1$.
10: until convergence.
```

The general formulation (10) of the proposed low-rank based framework is favorable as it involves absolutely no $n \times n$ matrix and only builds a much smaller $n \times l$ and a $l \times l$ matrix. By (10), we develop two alternative formulations, LEOD-*basic* and LEOD-*fast*, as well as corresponding algorithms to optimize them with fairly cheap computations.

The basic solution: LEOD-*basic*

If no additional hyperparameter is introduced, a direct way to avoid overly large $\boldsymbol{\alpha}$ for classifier $\mathbf{f}$ is to impose the constraint $\|\boldsymbol{\alpha}\|_2 = 1$, which is the objective of LEOD-*basic*:

$$\begin{aligned} \min_{\|\boldsymbol{\alpha}\|_2=1,\tilde{\mathbf{y}}} \quad & \boldsymbol{\alpha}^\top\mathbf{U}_T\mathbf{M}\mathbf{U}_T^\top\boldsymbol{\alpha} - 2\tilde{\mathbf{y}}^\top\mathbf{U}_K\mathbf{D}_K\mathbf{U}_K^\top\boldsymbol{\alpha} \\ s.t.\ \ & \tilde{y}_i \in \{c^+ + \frac{\gamma_2}{n^+}, c^-\},\ 0 < n^+ < n \end{aligned} \tag{11}$$

We can optimize (11) by alternate optimization that optimizes the $\boldsymbol{\alpha}$-problem and $\tilde{\mathbf{y}}$-problem alternatively with $\tilde{\mathbf{y}}$ and $\boldsymbol{\alpha}$ fixed respectively. The details are shown below.

Re-weighting algorithm for $\boldsymbol{\alpha}$-problem The major bottleneck to solve (11) is the $\boldsymbol{\alpha}$-problem, which actually requires solving the constrained eigenvalue problem below:

$$\min_{\|\boldsymbol{\alpha}\|_2=1} \ \boldsymbol{\alpha}^\top\mathbf{U}_T\mathbf{M}\mathbf{U}_T^\top\boldsymbol{\alpha} - 2\mathbf{b}^\top\boldsymbol{\alpha} \tag{12}$$

where $\mathbf{b} = \mathbf{U}_K\mathbf{D}_K(\mathbf{U}_K^\top\tilde{\mathbf{y}})$. However, the classic solution to (12) (Gander, Golub, and Matt 1991) requires building and decomposing a large $2n \times 2n$ asymmetric matrix to yield its smallest real-valued eigenvalue. This results in a $O((2n)^2)$ space and $O((2n)^3)$ time complexity, which can be computationally prohibitive for even medium n and are not straightforward to reduce. To make the optimization cheap, we propose a brand new algorithm to solve the constrained eigenvalue problem: Recall the constraint $\boldsymbol{\alpha}^\top\boldsymbol{\alpha} = 1$, so we can cleverly transform (12) into the equivalent form below by reversing the sign of (12) and adding a positive constant $\beta\boldsymbol{\alpha}^\top\mathbf{I}\boldsymbol{\alpha} = \beta$:

$$\max_{\|\boldsymbol{\alpha}\|_2=1} \ \boldsymbol{\alpha}^\top(\beta\mathbf{I} - \mathbf{U}_T\mathbf{M}\mathbf{U}_T^\top)\boldsymbol{\alpha} + 2\mathbf{b}^\top\boldsymbol{\alpha} \tag{13}$$

Note that β should be sufficiently large to make $(\beta\mathbf{I} - \mathbf{U}_T\mathbf{M}\mathbf{U}_T^\top) \triangleq \mathbf{Q}$ a PSD matrix, so we choose β to be the largest eigenvalue of $(\mathbf{U}_T\mathbf{M}\mathbf{U}_T^\top)$, which can be obtained efficiently by power iterations or randomized SVD (Halko, Martinsson, and Tropp 2009). In this way, we obtain a convex function $h(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^\top\mathbf{Q}\boldsymbol{\alpha} + 2\mathbf{b}^\top\boldsymbol{\alpha}$. With $g(\boldsymbol{\alpha}) = \boldsymbol{\alpha}$ and $p(\boldsymbol{\alpha}) = 0$, the objective (13) actually has the form below:

$$\max_{\mathbf{x}\in\mathcal{C}} \ h(g(\mathbf{x})) + p(\mathbf{x}) \tag{14}$$

Objective functions with form (14) can be solved by iteratively re-weighting $\mathbf{x}$, which is originally used to seek the optimal mean for Robust PCA (Nie, Yuan, and Huang 2014):

$$\mathbf{x}^{(t+1)} = \arg\max_{\mathbf{x}\in\mathcal{C}} \mathrm{Tr}(D^\top(\mathbf{x}^{(t)})g(\mathbf{x})) + p(\mathbf{x}) \tag{15}$$

where $D(\mathbf{x}_t)$ denotes any subgradient function of h at the point $g(\mathbf{x}_t)$. In our problem, it is easy to know $D(\boldsymbol{\alpha}) = \mathbf{Q}\boldsymbol{\alpha} + \mathbf{b}$. Using Cauchy-Schwarz inequality, we can find the optimal $\boldsymbol{\alpha}^*$ by re-weighting $\boldsymbol{\alpha}$ as follows:

$$\boldsymbol{\alpha}^{(t+1)} = \arg\max_{\|\boldsymbol{\alpha}\|_2=1} (\mathbf{Q}\boldsymbol{\alpha}^{(t)} + \mathbf{b})^\top\boldsymbol{\alpha} = \frac{\mathbf{Q}\boldsymbol{\alpha}^{(t)} + \mathbf{b}}{\|\mathbf{Q}\boldsymbol{\alpha}^{(t)} + \mathbf{b}\|_2} \tag{16}$$

Based on (16), a novel re-weighting based algorithm is summarized in Algorithm 1, which is easy to implement and only takes $O(n)$ additional space complexity. The convergence of the proposed re-weighting algorithm is guaranteed by the proved convergence of re-weighting (Nie, Yuan, and Huang 2014), which is examplified by Fig. 3a: Given a PSD matrix $(\mathbf{U}_T\mathbf{M}\mathbf{U}_T^\top)$ and a vector $\mathbf{b}$ for problem (12), the objective value of the re-weighting algorithm can quickly converge to the optimal value yielded by the classic solution (Gander, Golub, and Matt 1991). Based on the proposed novel re-weighting algorithm, optimizing LEOD-*basic* becomes significantly cheaper, especially in terms of space complexity ($O(n)$ vs. $O((2n)^2)$). More importantly, our re-weighting algorithm provides as a general solution to solve the constrained eigenvalue problem efficiently.

Optimization of $\tilde{\mathbf{y}}$-problem When we fix $\boldsymbol{\alpha}$ and remove the constant term in (11), the second step of alternate optimization is to solve the $\tilde{\mathbf{y}}$-problem below:

$$\begin{aligned} \min_{\tilde{\mathbf{y}}} \quad & -\mathbf{b}'^\top\tilde{\mathbf{y}} \\ s.t.\ \ & \tilde{y}_i \in \{c^+ + \frac{\gamma_2}{n^+}, c^-\},\ 0 < n^+ < n \end{aligned} \tag{17}$$

where $\mathbf{b}' = \mathbf{U}_K\mathbf{D}_K(\mathbf{U}_K^\top\boldsymbol{\alpha})$. We simply follow (Liu, Hua, and Smith 2014) to solve problem (17): Consider the m-subproblem that assumes there exist m positive labels in $\tilde{\mathbf{y}}$, i.e. $n^+ = m$ $(0 < m < n)$. The optimal solution to this m-subproblem $\tilde{\mathbf{y}}_m^*$ is obtained by assigning $\tilde{y}_i = c^+ + \frac{\gamma_2}{m}$ when b_i' is among the m largest element of $\mathbf{b}'$, and $\tilde{y}_i = c^-$ otherwise. Therefore, we simply scan m from 1 to $n-1$ and find the global optimal solution $\tilde{\mathbf{y}}^*$ in (17) efficiently by:

$$\tilde{\mathbf{y}}^* = \arg\max_{0<m<n} -(\mathbf{U}_K\mathbf{D}_K(\mathbf{U}_K^\top\boldsymbol{\alpha}))^\top\tilde{\mathbf{y}}_m^* \tag{18}$$

To be even faster: LEOD-*fast*

Despite that LEOD-*basic* is very cheap in terms of space complexity, it involves an iterative re-weighting algorithm,

Algorithm 2 LEOD Algorithm

```
1: Input: $\mathbf{U}_T$, $\mathbf{M}$, $\mathbf{U}_K$, $\mathbf{D}_K$, hyperparameter $\lambda$ (if needed).
2: Output: Optimal $\boldsymbol{\alpha}^*$, soft label assignment $\tilde{\mathbf{y}}^*$.
3: Initialize $\boldsymbol{\alpha}_0$ by $\frac{1}{\sqrt{n}}\mathbf{1}_n$, $i = 0$.
4: Initialize $\tilde{\mathbf{y}}_0$ by Eq. (18) with $\boldsymbol{\alpha}_0, \mathbf{U}_K, \mathbf{D}_K$.
5: repeat
6:     Update $\boldsymbol{\alpha}_{i+1}$ by either of the two methods:
       (1) LEOD-basic: Algorithm 1 with $\tilde{\mathbf{y}}_i, \mathbf{U}_T, \mathbf{M}$
       (2) LEOD-fast: Eq. (22) with $\tilde{\mathbf{y}}_i, \mathbf{U}_T, \mathbf{M}$ and $\lambda$ .
7:     Update $\tilde{\mathbf{y}}_{i+1}$ by Eq. (18) with $\boldsymbol{\alpha}_{i+1}, \mathbf{U}_K, \mathbf{D}_K$.
8:     $i = i + 1$.
9: until convergence.
```

which may not be fast enough to handle the growing demand for high processing speed. Thus, we propose a faster alternative formulation named LEOD-*fast* by using a regularization term rather than imposing any constraint on $\boldsymbol{\alpha}$:

$$\min_{\boldsymbol{\alpha},\tilde{\mathbf{y}}} \ \boldsymbol{\alpha}^\top \mathbf{U}_T \mathbf{M} \mathbf{U}_T^\top \boldsymbol{\alpha} - 2\mathbf{b}^\top \boldsymbol{\alpha} + \lambda \boldsymbol{\alpha}^\top \boldsymbol{\alpha}$$
$$s.t. \ \ \tilde{y}_i \in \{c^+ + \frac{\gamma_2}{n^+}, c^-\}, \ 0 < n^+ < n \tag{19}$$

where $\lambda > 0$ is a tunable hyperpaprameter and $\mathbf{b}$ is identically defined as (12). As the $\tilde{\mathbf{y}}$-problem of LEOD-*fast* is the same as LEOD-*basic*, we will only discuss the optimization of the $\boldsymbol{\alpha}$-problem, which is the key for acceleration.

Closed-form solution for $\boldsymbol{\alpha}$-problem With $\tilde{\mathbf{y}}$ fixed, we can yield the following unconstrained optimization problem:

$$\min_{\boldsymbol{\alpha}} \ \boldsymbol{\alpha}^\top (\mathbf{U}_T \mathbf{M} \mathbf{U}_T^\top + \lambda \mathbf{I}) \boldsymbol{\alpha} - 2\mathbf{b}^\top \boldsymbol{\alpha} \tag{20}$$

We note that problem (20) is a kernel ridge regression problem, which can be solved by the closed-form solution:

$$\boldsymbol{\alpha}^* = (\mathbf{U}_T \mathbf{M} \mathbf{U}_T^\top + \lambda \mathbf{I}_n)^{-1} \mathbf{b} \tag{21}$$

Solution in (21) still requires inverting a $n \times n$ matrix by $O(n^2)$ space complexity and $O(n^3)$ time complexity. Considering that the formulation is based on the low-rank representation $\mathbf{U}_T \mathbf{M} \mathbf{U}_T^\top$, we can apply the Woodbury-Sherman-Morrison formula and covert (21) into a more efficient form:

$$\boldsymbol{\alpha}^* = \frac{1}{\lambda}(\mathbf{b} - \mathbf{U}_T(\mathbf{M}\mathbf{U}_T^\top \mathbf{U}_T + \lambda \mathbf{I}_l)^{-1} \mathbf{M}(\mathbf{U}_T^\top \mathbf{b})) \tag{22}$$

By Eq. (22), we can compute $\boldsymbol{\alpha}^*$ analytically by simply inverting a small $l \times l$ matrix, which takes merely $O(l^2)$ space complexity and $O(l^3)$ time complexity. Therefore, it enjoys an overall $O(l^2 + nl)$ space complexity. Finally, we summarize the entire algorithm of LEOD in Algorithm 2. The alternate optimization of LEOD is convergent, the proof of which is provided in supplementary material, and experiments show that LEOD usually converges in less than 5 iterations (see Fig. 3a).

Experiments

Experimental settings

We conduct experiments on five widely-used benchmark datasets: Caltech101, Cifar-10, MNIST, Fashion and SVHN.

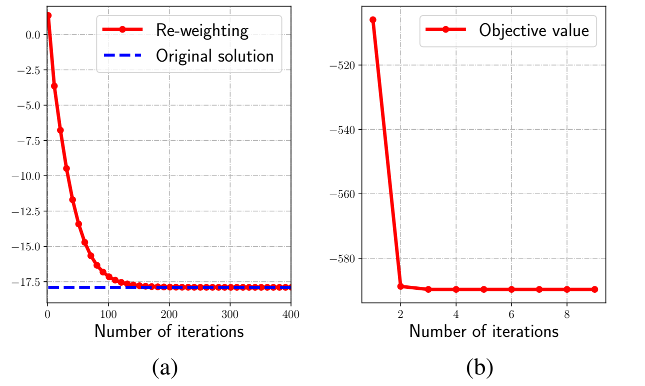


Figure 3: Algorithm convergence. Left: The re-weighting algorithm. Right: The entire LEOD algorithm.

For performance evaluation, we first follow (Liu, Hua, and Smith 2014) to obtain noisy data mixed with outliers: On each dataset, each time data from a certain class of testing set are used as inliers, while data from other classes are randomly sampled as outliers by a certain outlier ratio ρ and then mixed with the inliers. On Caltech101, the performance is evaluated by averaging the results on four big classes with more than 300 images, while for Cifar-10, MNIST, Fashion and SVHN dataset the results are averaged on all classes. Since we are interested in robustness against high outlier ratios, ρ is varied between 20%-60%. As to feature extraction, images of Caltech101 are represented by Locality-constrained Linear Coding (LLC) descriptor (Wang et al. 2010), while images from Cifar-10, MNIST, SVHN and Fashion are represented by features extracted by a pre-trained deep neural network[1]. For an extensive comparison of performance, we compare the proposed LEOD with seven prevalent methods for outlier detection: Local Outlier Factor (LOF) (Breunig et al. 2000), one-class Support Vector Machine (OCSVM) (Schölkopf et al. 2001), Discriminative Reconstruction Autoencoder (DRAE) (Xia et al. 2015), Kernel Principal Component Analysis (KPCA) (Schölkopf, Smola, and Müller 1997), Isolation Forest (Liu, Ting, and Zhou 2008), Robust Kernel Density Estimation (RKDE) (Kim and Scott 2011) and UOCL (Liu, Hua, and Smith 2014). The implementation details are given as follows: For LEOD, we set $m = ceil(n/10)$, $l = ceil(n/20)$ for low-rank approximation, and we set $\lambda = 1.0$ for LEOD-*fast*. γ_1 and γ_2 are both fixed to be 1.0, which are shared by UOCL. As for k-nn, we adopt kd-tree or ball-tree to find k-nn for and fix $k = 6$ for LEOD, LOF and UOCL. The standard Gaussian kernel is adopted for OCSVM, RKDE, KPCA, UOCL and LEOD to calculate pair-wise similarity, and the kernel width σ is estimated by $\sigma^2 = \sum_{i,j=1}^n ||\mathbf{x}_i - \mathbf{x}_j||_2^2/n^2$. For DRAE, we strictly follow (Xia et al. 2015) by using an autoencoder with 30 linear hidden neurons and an encouraging term weight

[1]https://github.com/yanssy/pytorch-playground

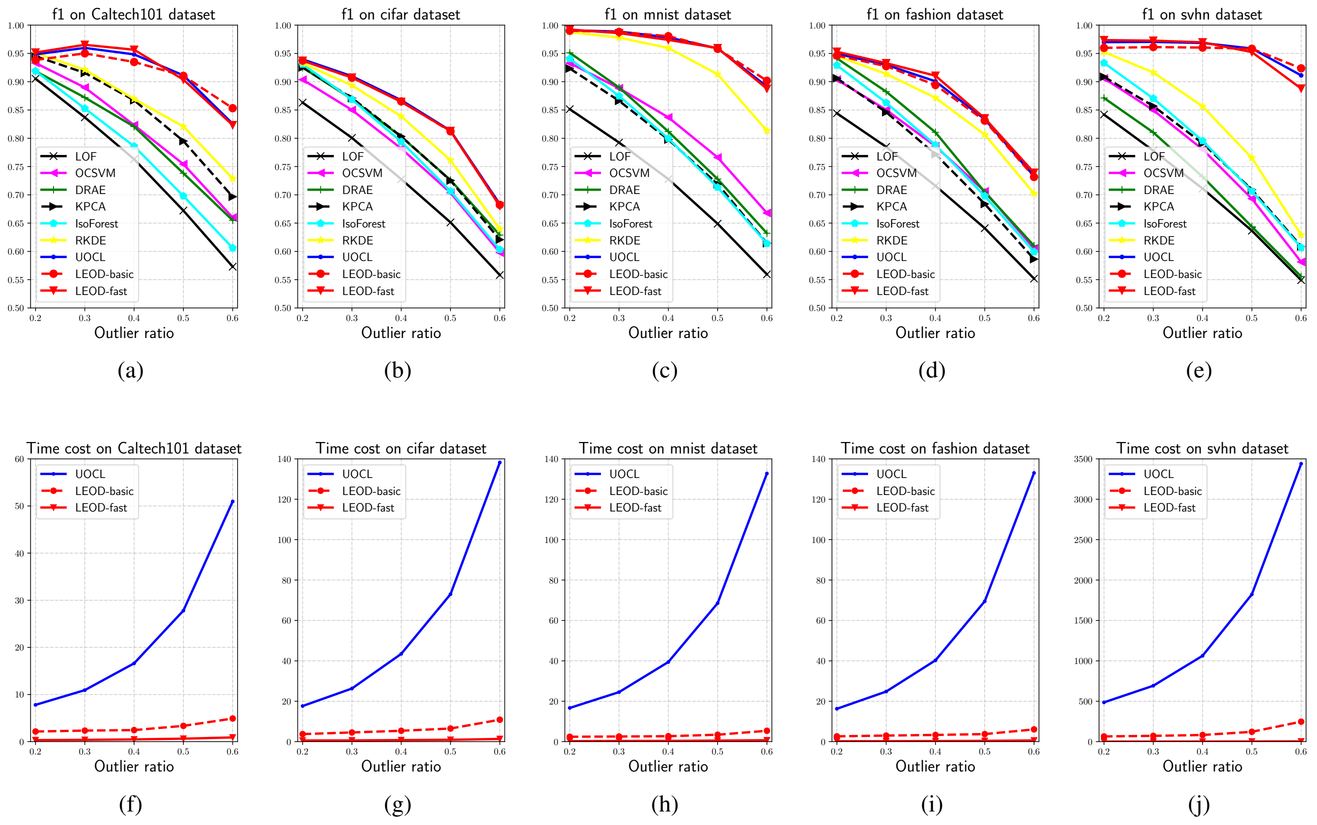


Figure 4: Average performance and time cost on benchmark datasets.

0.1. For OCSVM, we set $\nu = \rho$ to enable OCSVM to exclude outliers during optimization. All methods are implemented in a Python 3.6 environment with PyTorch 0.3.0[2] and scikit-learn 0.19.1[3] packages.

Experimental results

Performance To evaluate the performance under high outlier ratios, the $f1$-score of different methods under outlier ratios from 20% to 60% on five benchmark datasets are reported in Fig. 4a-4e and Table 1. We obtain the following observations: First, LEOD-*basic*, LEOD-*fast* and UOCL have shown evidently stronger robustness by significantly outperforming other methods under high outlier ratios, and they are even able to attain at most 29.5% $f1$-score gain (see Fig. 4e) when compared to the best performer among other methods (RKDE). Such robust performance demonstrates that the proposed low-rank based framework can be readily applied to discovering outliers under high outlier ratios. Second, LEOD-*basic*, LEOD-*fast* and UOCL yield comparable performance with minor differences. Interestingly, with the proposed low-rank based similarity representation, we note that in many cases LEOD-*basic* and LEOD-*fast* are able to outperform UOCL that uses the full similarity matrix (see Table 1). Such results justify our analysis to preserve only the low-rank structure of similarity matrix for outlier detection. Consequently, results above identify LEOD as a robust solution to outlier detection under high outlier ratios.

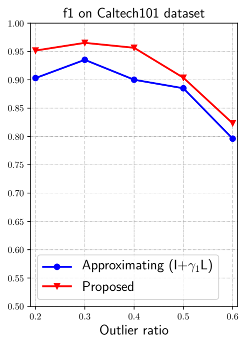
Computational cost Since LEOD and UOCL stand out with evidently stronger robustness against high outlier ratios than other methods, we specifically compare their computational cost (running time/peak memory usage) during the algorithm implementation in Fig. 4f-4j and Table 1: In terms of time cost, LEOD-*basic* and LEOD-*fast* are both significantly faster than UOCL, and the advantage tends to be larger as ρ gets larger and more data (outliers) are added. In particular, LEOD-*fast* has constantly been the fastest method, which only costs a few seconds and achieves at most 1000 times acceleration compared to UOCL that can takes minutes or even hours (see Table 1). As to the peak memory usage, LEOD-*basic* has constantly been the most memory-efficient method as it only involves an iterative $O(n)$ optimization, while LEOD-*fast* uses slightly more memory than LEOD-*basic*. By contrast, UOCL tends to consume astonishing memory when n gets large, e.g. it uses 39.06 GiB memory when $\rho = 60\%$ on SVHN dataset ($n \approx 12000$), which is about 100 times more expensive than

[2]https://pytorch.org/

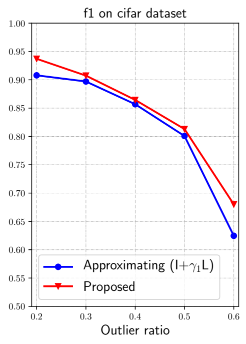
[3]http://scikit-learn.org/stable/

Table 1: Comparison of $f1$-score (%), time (sec.) and peak memory usage (GiB) on benchmark datasets. From top to bottom: Caltech101, Cifar-10, MNIST, Fashion, SVHN. UOCL, LEOD-basic and LEOD-fast are shorted as "U", "Lb" and "Lf". Best results are shown in bold.

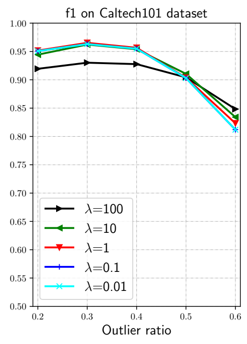
	$\rho = 0.2$			$\rho = 0.3$			$\rho = 0.4$			$\rho = 0.5$			$\rho = 0.6$		
	$f1$	Time	Mem.	$f1$	Time	Mem.	$f1$	Time	Mem.	$f1$	Time	Mem.	$f1$	Time	Mem.
U	94.8	7.8	1.11	96.0	10.9	1.26	94.8	16.6	1.26	**91.1**	27.8	1.34	82.6	51.0	1.58
Lb	93.8	2.2	**0.97**	95.0	2.3	**1.01**	93.5	2.5	**1.06**	91.0	3.3	**1.12**	**85.3**	4.9	**1.25**
Lf	**95.2**	**0.3**	0.98	**96.5**	**0.4**	**1.01**	**95.7**	**0.5**	1.08	90.4	**0.6**	**1.12**	82.3	**0.9**	**1.25**
U	**94.0**	17.6	0.50	**91.0**	26.3	0.66	**86.7**	43.5	0.89	**81.3**	73.0	1.22	**68.4**	138.2	1.64
Lb	93.7	3.7	**0.09**	90.7	4.6	**0.09**	86.5	5.4	**0.09**	81.2	6.5	**0.12**	68.2	10.9	**0.15**
Lf	93.7	**0.6**	**0.09**	90.7	**0.6**	**0.09**	86.4	**0.8**	**0.09**	**81.3**	**0.9**	**0.12**	68.0	**1.4**	0.16
U	99.1	16.7	0.58	**98.9**	24.5	0.75	97.8	39.4	1.05	**95.9**	68.5	1.40	89.2	132.7	2.03
Lb	99.0	2.4	**0.05**	98.8	2.6	**0.05**	**98.0**	2.7	**0.05**	95.8	3.4	**0.05**	**90.2**	5.4	**0.06**
Lf	**99.2**	**0.3**	**0.05**	98.6	**0.3**	**0.05**	97.4	**0.4**	**0.05**	**95.9**	**0.6**	**0.05**	88.7	**0.7**	**0.06**
U	94.9	16.3	0.48	93.0	24.8	0.64	90.1	40.3	0.83	83.4	69.5	1.15	73.3	133.0	1.61
Lb	94.6	2.6	**0.02**	92.7	3.0	**0.02**	89.4	3.3	**0.02**	83.1	3.7	**0.03**	73.2	6.1	**0.03**
Lf	**95.3**	**0.2**	**0.02**	**93.3**	**0.2**	**0.02**	**91.1**	**0.3**	**0.02**	**83.5**	**0.4**	**0.03**	**73.9**	**0.6**	0.05
U	97.0	486.2	9.84	97.0	692.0	12.81	96.8	1062.7	17.42	**95.9**	1822.9	25.20	91.1	3439.0	39.06
Lb	96.0	64.8	**0.12**	96.1	71.7	**0.17**	96.0	82.4	**0.20**	95.8	121.9	**0.23**	**92.4**	247.2	**0.36**
Lf	**97.4**	**1.1**	**0.12**	**97.3**	**1.4**	**0.17**	**97.0**	**1.6**	**0.20**	95.2	**2.3**	0.27	88.8	**3.4**	0.40



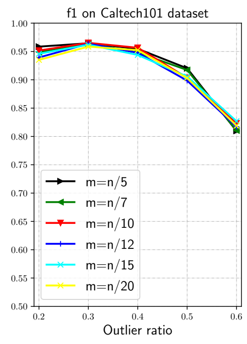
(a)



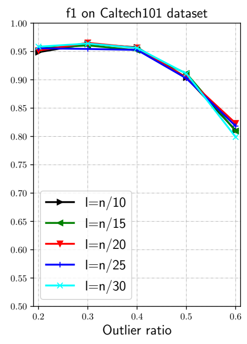
(b)



(c)



(d)



(e)

Figure 5: Discussion on approximating both $\mathbf{K}$ and $\mathbf{L}$, as well as the influence of hyperparameter λ, m and l.

LEOD (0.36/0.40 GiB). Therefore, LEOD also turns out to be a much cheaper solution when compared with UOCL.

Discussion

Approximating $(\mathbf{I} + \gamma_1\mathbf{L})$ To eliminate the $n \times n$ matrix $(\mathbf{I} + \gamma_1\mathbf{L})$, we compare the naive solution, which directly computes its low-rank approximation, with our solution that exploits the sparsification of $\mathbf{L}$. The performance comparison is shown in Fig. 5a and 5b), which indicate that the performance of the naive solution is constantly worse than our solution by at most 6% performance loss.

Influence of hyperparameters The proposed LEOD additionally introduces three hyperparameters: λ for LEOD-*fast*, as well as m and l for low-rank approximation. With other hyperparamters fixed, we explore their influence by varying λ from 0.01 to 100.0, m from $n/5$ to $n/20$ and l from $n/10$ to $n/30$ respectively (the $ceil(\cdot)$ operator is temporarily omitted for simplicity). The corresponding performance is shown in Fig. 5c-5e. For λ, we obtain the two observations: First, with some minor variations, the performance is insensitive to λ when it is varied between 0.01 and 10.0. Second, we notice that a large λ (e.g. $\lambda = 100$) tend to obtain better performance (at most 4% $f1$ gain) under a very high outlier ratio (e.g. $\rho = 60\%$), but it severely degrades the performance when $\rho \leq 40\%$. By contrast, a very small λ (e.g. $\lambda = 0.01$) usually leads to a bad performance under a very high outlier ratio like 60%. Empirically, we can set λ to be 1.0 or 10, but one can use a large λ when ρ is estimated to be very high ($> 50\%$). As for m and l, our results (see Fig. 5d and 5e) show that LEOD's performance remains relatively stable with different m and l in most cases (the performance variance is within 3%), so we simply set $m = ceil(n/10)$ and $l = ceil(n/20)$.

Conclusion

In this paper, we formally discuss the problem of outlier detection under high outlier ratios and proposes a highly efficient method named LEOD to achieve good robustness with much cheaper computations. Based on the proposed low-rank based framework, LEOD-*basic* enjoys very low space complexity by utilizing a novel algorithm to solve the general constrained eigenvalue problem efficiently, while LEOD-*fast* realizes a significantly faster learning speed by a new objective that takes a very cheap closed-form solution. Therefore, LEOD provides a promising solution for robust and efficient outlier detection in practical applications.

Acknowledgement

The work is supported by the National Key R&D Program of China under Grant No. 2018YFB1003203, National Natural Science Foundation of China under Grant No. 61672528, 61773392, 61702539. Xinwang Liu, En Zhu and Jianping Yin are corresponding authors of this paper (email addresses: xinwangliu@nudt.edu.cn, enzhu@nudt.edu.cn, jpyin@dgut.edu.cn).

References

Aggarwal, C. C.; Zhao, Y.; and Yu, P. S. 2011. Outlier detection in graph streams. In *IEEE International Conference on Data Engineering*, 399–409.

Breunig, M. M.; Kriegel, H.-P.; Ng, R. T.; and Sander, J. 2000. Lof: identifying density-based local outliers. In *ACM sigmod record*, volume 29, 93–104. ACM.

Chandola, V., and Kumar, V. 2007. Outlier detection : A survey. *Acm Computing Surveys* 41(3).

Désir, C.; Bernard, S.; Petitjean, C.; and Heutte, L. 2013. One class random forests. *Pattern Recognition* 46(12):3490–3506.

Gander, W.; Golub, G. H.; and Matt, U. V. 1991. *A Constrained Eigenvalue Problem*. Springer Berlin Heidelberg.

Gupta, G., and Ghosh, J. 2005. Robust one-class clustering using hybrid global and local search. In *Proceedings of the 22nd international conference on Machine learning*, 273–280. ACM.

Halko, N.; Martinsson, P.-G.; and Tropp, J. A. 2009. Finding structure with randomness: Stochastic algorithms for constructing approximate matrix decompositions.

Japkowicz, N.; Myers, C.; and Gluck, M. 1995. A novelty detection approach to classification. In *International Joint Conference on Artificial Intelligence*, 518–523.

Kim, J. S., and Scott, C. D. 2011. Robust kernel density estimation. *Journal of Machine Learning Research* 13(1):2529–2565.

Kim, J., and Scott, C. D. 2012. Robust kernel density estimation. *Journal of Machine Learning Research* 13(Sep):2529–2565.

Li, M.; Kwok, J. T.-Y.; and Lü, B. 2010. Making large-scale nyström approximation possible. In *ICML 2010-Proceedings, 27th International Conference on Machine Learning*, 631.

Li, S.; Shao, M.; and Fu, Y. 2014a. Locality linear fitting one-class svm with low-rank constraints for outlier detection. In *International Joint Conference on Neural Networks*, 676–683.

Li, S.; Shao, M.; and Fu, Y. 2014b. *Low-Rank Outlier Detection*. Springer International Publishing.

Li, S.; Shao, M.; and Fu, Y. 2018. Multi-view low-rank analysis with applications to outlier detection. *Acm Transactions on Knowledge Discovery from Data* 12(3):1–22.

Liu, W.; Hua, G.; and Smith, J. R. 2014. Unsupervised one-class learning for automatic outlier removal. In *CVPR*, 3826–3833.

Liu, F. T.; Ting, K. M.; and Zhou, Z.-H. 2008. Isolation forest. In *2008 Eighth IEEE International Conference on Data Mining*, 413–422. IEEE.

Liu, F. T.; Ting, K. M.; and Zhou, Z.-H. 2010. On detecting clustered anomalies using sciforest. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 274–290. Springer.

Luca, M., and Zervas, G. 2016. Fake it till you make it: Reputation, competition, and yelp review fraud. *Management Science* 62(12):3412–3427.

Nie, F.; Yuan, J.; and Huang, H. 2014. Optimal mean robust principal component analysis. In *International conference on machine learning*, 1062–1070.

Parzen, E. 1962. On estimation of a probability density function and mode. *Annals of Mathematical Statistics* 33(3):1065–1076.

Ramaswamy, S.; Rastogi, R.; and Shim, K. 2000. Efficient algorithms for mining outliers from large data sets. In *ACM SIGMOD International Conference on Management of Data*, 427–438.

Schölkopf, B.; Platt, J. C.; Shawe-Taylor, J.; Smola, A. J.; and Williamson, R. C. 2001. Estimating the support of a high-dimensional distribution. *Neural computation* 13(7):1443–1471.

Scholkopf, B.; Herbrich, R.; and Smola, A. J. 2001. A generalized representer theorem. *european conference on computational learning theory* 416–426.

Schölkopf, B.; Smola, A.; and Müller, K.-R. 1997. Kernel principal component analysis. In *International Conference on Artificial Neural Networks*, 583–588. Springer.

Tax, D. M., and Duin, R. P. 2004. Support vector data description. *Machine learning* 54(1):45–66.

Wang, J.; Yang, J.; Yu, K.; Lv, F.; Huang, T.; and Gong, Y. 2010. Locality-constrained linear coding for image classification. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 3360–3367.

Williams, C. K., and Seeger, M. 2001. Using the nyström method to speed up kernel machines. In *Advances in neural information processing systems*, 682–688.

Xia, Y.; Cao, X.; Wen, F.; Hua, G.; and Sun, J. 2015. Learning discriminative reconstructions for unsupervised outlier removal. In *Proceedings of the IEEE International Conference on Computer Vision*, 1511–1519.