**Noname manuscript No.**
(will be inserted by the editor)

# Fast and Accurate Pseudoinverse with Sparse Matrix Reordering and Incremental Approach

**Jinhong Jung · Lee Sael**



Received: date / Accepted: date

**Abstract** How can we compute the pseudoinverse of a sparse feature matrix efficiently and accurately for solving optimization problems? A pseudoinverse is a generalization of a matrix inverse, which has been extensively utilized as a fundamental building block for solving linear systems in machine learning. However, an approximate computation, let alone an exact computation, of pseudoinverse is very time-consuming due to its demanding time complexity, which limits it from being applied to large data. In this paper, we propose FastPI (Fast PseudoInverse), a novel incremental singular value decomposition (SVD) based pseudoinverse method for sparse matrices. Based on the observation that many real-world feature matrices are sparse and highly skewed, FastPI reorders and divides the feature matrix and incrementally computes low-rank SVD from the divided components. To show the efficacy of proposed FastPI, we apply them in real-world multi-label linear regression problems. Through extensive experiments, we demonstrate that FastPI computes the pseudoinverse faster than other approximate methods without loss of accuracy. Results imply that our method efficiently computes the low-rank pseudoinverse of a large and sparse matrix that other existing methods cannot handle with limited time and space.

**Keywords** Pseudoinverse · Sparse Matrix Reordering · Incremental SVD · Multi-label Linear Regression

Jinhong Jung
Jeonbuk National University, 567 Baekje-daero, Deokjin-gu, Jeonju 54907, Republic of Korea
E-mail: jinhongjung@jbnu.ac.kr

Lee Sael (Corresponding author.)
Ajou University, 206 Worldcup-ro, Yeongton-ju, Suwon 16499, Republic of Korea
E-mail: sael@ajou.ac.kr

## 1 Introduction

A pseudoinverse is a generalized inverse method for all types of matrices (Ben-Israel and Greville, 2003) that play a crucial role in obtaining best-fit solutions to the linear systems even when unique solutions do not exist (Strang, 2006). Pseudoinverses have been studied by many researchers in various domains, including mathematics and machine learning, from the viewpoint of theory (Ben-Israel and Greville, 2003), computational engineering (Golub and Van Loan, 2012), and applications (Guo et al., 2019; Xu and Guo, 2018; He et al., 2016; Spyromitros-Xioufis et al., 2016; Chen and Lin, 2012; Horata et al., 2013).

Although pseudoinverses have been widely applied, applications were limited to small data due to their high computational complexity. More specifically, the most widely applied pseudoinverse is the Moore-Penrose inverse; and the most elegant and precise solution for obtaining the Moore-Penrose inverse is by utilizing a singular value decomposition (SVD). However, calculating SVDs are impractical for large matrices, i.e., the time complexity of a full-rank SVD for an $m \times n$ matrix is $min(O(n^2m), O(nm^2))$ (Trefethen and Bau III, 1997). Low-rank approximation techniques (Halko et al., 2011; Feng et al., 2018) have been proposed to reduce the time complexity problem. However, costs can still be improved, especially for handling large matrices using larger rank approximations for higher accuracies; e.g., $O(mn\log(r)+(m+n)r^2)$ when a randomized algorithm is used, where $r$ is the low-rank, is still large.

In this paper, we propose FastPI (Fast PseudoInverse), a novel approximation algorithm for computing pseudoinverse efficiently and accurately based on sparse matrix reordering and incremental low-rank SVD. FastPI was motivated by the observation that many real-world feature matrices are highly sparse and skewed. Based on this observation, FastPI reorders a feature matrix such that its non-zero elements are concentrated at the bottom right corner leaving a large sparse area at the top left of the feature matrix (Figure 3(e)). The reordered matrix is split into four submatrices, where one of the submatrices is the large and sparse rectangular block diagonal matrix, whose SVD is easy-to-compute. FastPI efficiently obtains the approximate pseudoinverse of the feature matrix by performing incremental low-rank SVD starting from the SVD of this block diagonal submatrix. Experiments show that FastPI successfully approximates the pseudoinverse faster than compared methods without loss of accuracy in the multi-label linear regression problem. Our contributions are the followings:

- **Observation.** We observed that a sparse feature matrix can be transformed to a bipartite network (Definition 1) characteristic with a highly skewed node degree distribution (Figure 1).
- **Method.** We propose FastPI, a novel method for efficiently and accurately obtaining the approximate pseudoinverse with sparse matrix reordering and incremental SVD (Algorithm 1).

Table 1: Table of symbols.

| Symbol | Definition |
|---|---|
| $m$ | number of training instances |
| $n$ | number of features |
| $L$ | number of labels |
| $\mathbf{A} \in \mathbb{R}^{m \times n}$ | input feature matrix |
| $\mathbf{A}^\dagger \in \mathbb{R}^{n \times m}$ | pseudoinverse of the feature matrix |
| $\mathbf{U}_{m \times r}$ | $m \times r$ matrix of left singular vectors |
| $\mathbf{\Sigma}_{r \times r}$ | $r \times r$ diagonal matrix of singular values |
| $\mathbf{V}^\top_{r \times n}$ | $r \times n$ matrix of right singular vectors |
| $\alpha$ | target rank ratio in Algorithm 1 where $0 < \alpha \leq 1$ |
| $r$ | target rank, i.e., $r = \lceil \alpha n \rceil$ for $m \times n$ matrix when $m > n$ |
| $\mathbf{A}_{ij} \in \mathbb{R}^{m_i \times n_j}$ | $(i, j)$-th submatrix of reordered $\mathbf{A}$ |
| $k$ | hub selection ratio in Algorithm 2 where $0 < k < 1$ |
| $m_1$ & $n_1$ | number of spoke instance and feature nodes, respectively |
| $m_2$ & $n_2$ | number of hub instance and feature nodes, respectively |
| $B$ | number of rectangular blocks in $\mathbf{A}_{11}$ |
| $m_{1i}$ & $n_{1i}$ | height and width of $i$-th block in $\mathbf{A}_{11}$, respectively |
| $\lvert\mathbf{A}\rvert$ | number of non-zero entries in $\mathbf{A}$ |

– **Experiment.** We show FastPI computes an approximate pseudoinverse faster than its competitors for most datasets without loss of accuracy in the multi-label linear regression experiments (Figures 5 and 6).

## 2 Preliminaries

We describe the preliminaries on pseudoinverse and singular value decomposition (SVD), and provide the formal definition of the problem and target application handled in this paper. Symbols used in the paper are summarized in Table 1.

### 2.1 Pseudoinverse and SVD

In many machine learning models, a training data is represented as a feature matrix denoted by $\mathbf{A} \in \mathbb{R}^{m \times n}$, where $m$ is the number of training instances and $n$ is the number of features. Learning optimal model parameters often involves pseudoinverse $\mathbf{A}^\dagger \in \mathbb{R}^{n \times m}$. The Moore-Penrose inverse is the most accurate and widely used generalized matrix inverse that can be solved using SVD as follows.

**Problem 1 (Solving Moore-Penrose Inverse via low-rank SVD (Golub and Van Loan, 2012))** For feature matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, let $\mathbf{A}$ be decomposed into $\mathbf{U}_{m \times r} \mathbf{\Sigma}_{r \times r} \mathbf{V}^\top_{r \times n}$, where $\mathbf{U}_{m \times r}$ and $\mathbf{V}_{n \times r}$ are orthogonal matrices and $\mathbf{\Sigma}_{r \times r}$ is diagonal with $r$ singular values. If $r$ is the rank of $\mathbf{A}$, the pseudoinverse $\mathbf{A}^\dagger$ of $\mathbf{A}$ is given by $\mathbf{A}^\dagger = \mathbf{V}_{n \times r} \mathbf{\Sigma}^\dagger_{r \times r} \mathbf{U}^\top_{r \times m}$. Otherwise, for a given target rank $r$, it results in a best approximate pseudoinverse $\mathbf{A}^\dagger \approx \mathbf{V}_{n \times r} \mathbf{\Sigma}^\dagger_{r \times r} \mathbf{U}^\top_{r \times m}$.

The state-of-the-art low-rank SVD is randomized-SVD with the computational complexity of $O(mn\log(r) + (m+n)r^2)$. Randomized-SVD utilizes randomized algorithm with oversampling technique (see the details in Section 4.1) for efficient computation (Halko et al., 2011). In cases of sparse matrices, Krylov subspace-based methods have also been shown to be efficient (Baglama and Reichel, 2005). However, both methods target problems that require very small ranks, while as accurate approximations of pseudoinverses require relatively large rank approximations of SVDs. Thus, the costs of existing low-rank SVDs are still too heavy for practical applications of pseudoinverses on large feature matrices.

### 2.2 Target Application of Pseudoinverse

We describe our target application, multi-label linear regression based on pseudoinverse as follows:

**Application 1 (Multi-label Linear Regression (Yu et al., 2014))** *Given feature matrix* $\mathbf{A} \in \mathbb{R}^{m \times n}$ *and label matrix* $\mathbf{Y} \in \mathbb{R}^{m \times L}$ *where* $m > n$*,* $L$ *is the number of labels, and each row of* $\mathbf{Y}$ *is a binary label vector of size* $L$*, the goal is to learn parameter* $\mathbf{Z} \in \mathbb{R}^{n \times L}$ *satisfying* $\mathbf{AZ} \simeq \mathbf{Y}$ *to estimate the score vector* $\hat{\mathbf{y}} = \mathbf{Z}^\top \mathbf{a}$ *for a new feature vector* $\mathbf{a} \in \mathbb{R}^n$*.*

The linear system for unknown $\mathbf{Z}$ is over-determined when $m > n$; thus, the solution for $\mathbf{Z}$ is obtained by minimizing the least square error $\|\mathbf{AZ} - \mathbf{Y}\|_F^2$, which results in the closed form solution $\mathbf{Z} = \mathbf{A}^\dagger\mathbf{Y}$ (Chen and Lin, 2012). As described in Problem 1, $\mathbf{A}^\dagger \simeq \mathbf{V}_{n\times r}\mathbf{\Sigma}^\dagger_{r\times r}\mathbf{U}^\top_{r\times m}$, where the equality holds when $r$ is the rank of $\mathbf{A}$. Hence, the SVD results can be used to compute pseudoinverse exactly or approximately in a multi-label linear regression.

## 3 Proposed Method

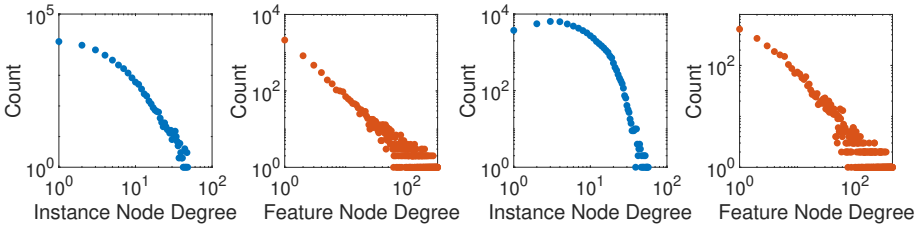
We propose FastPI (Fast PseudoInverse), a novel method for efficiently and accurately computing the approximate pseudoinverse for sparse matrices. We describe the overall procedure of FastPI in Algorithm 1. Our main ideas for accelerating the pseudoinverse computation are as follows:

- **Idea 1 (line 1).** Many feature matrices collected from real-world domains are highly sparse and skewed as shown in Figure 1 (**Section 3.1**); and we show that these feature matrices can be reordered such that their non-zeros are concentrated as shown in Figure 3(e) (**Section 3.2**).
- **Idea 2 (line 2).** The reordered matrix involves a large and sparse block diagonal submatrix whose SVD is easy-to-compute (**Section 3.3**).
- **Idea 3 (lines 3 and 4).** The final SVD result of the feature matrix is efficiently obtained by incrementally updating the SVD result of the sparse submatrix (**Section 3.3**).

**Algorithm 1:** FastPI

**Input:** input matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$ and target rank ratio $0 < \alpha \leq 1$
**Output:** approximate pseudoinverse $\mathbf{A}^\dagger$

1: reorder $\mathbf{A}$ using Algorithm 2, and divide $\mathbf{A}$ into $\begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$
2: $\mathbf{U}_{m_1 \times s} \mathbf{\Sigma}_{s \times s} \mathbf{V}^\top_{s \times n_1} \leftarrow$ compute the SVD result for $\mathbf{A}_{11}$ with target rank $s = \lceil \alpha n_1 \rceil$ according to Equation (1)
3: $\mathbf{U}_{m \times s} \mathbf{\Sigma}_{s \times s} \mathbf{V}^\top_{s \times n_1} \leftarrow$ incrementally update the SVD result for $\mathbf{A}_{21}$ with target rank $s = \lceil \alpha n_1 \rceil$ according to Equation (2)
4: $\mathbf{U}_{m \times r} \mathbf{\Sigma}_{r \times r} \mathbf{V}^\top_{r \times n} \leftarrow$ incrementally update the SVD result for $\begin{bmatrix} \mathbf{A}_{12} \\ \mathbf{A}_{22} \end{bmatrix}$ with target rank $r = \lceil \alpha n \rceil$ according to Equation (3)
5: $\mathbf{A}^\dagger \leftarrow$ solve pseudoinverse (Problem 1)
6: **return** $\mathbf{A}^\dagger$




(a) The Amazon dataset (b) The RCV dataset

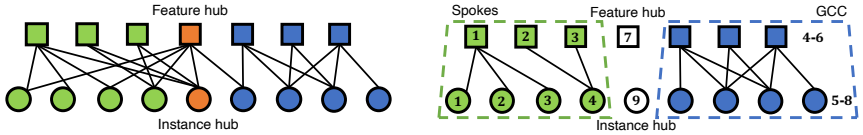
Fig. 1: Degree distributions of instance and feature nodes in a bipartite network derived from real-world feature matrices. Note that there are few high degree nodes while the majority of nodes have low degrees, implying skewness on the degree distributions.

### 3.1 Observation from Real-world Feature Matrix

We first explain the skewness of feature matrices in the real-world datasets, which plays a key role in motivating the matrix reordering of FastPI. A notable characteristic of feature matrices collected from many real-world problems is that they are extremely sparse as shown in Table 3 (the details of the datasets are described in Section 4).

This sparsity naturally leads us to interpret $\mathbf{A}$ as a sparse network. Moreover, rows of a feature matrix map training instances, columns map the features, and non-zero values map the relations between instance-to-feature pairs. Thus a feature matrix $\mathbf{A}$ naturally represents a bipartite network as follows:

**Definition 1 (Bipartite Network from Feature Matrix)** Given $\mathbf{A} \in \mathbb{R}^{m \times n}$, a bipartite network $G = (V_T, V_F, E)$ is derived from $\mathbf{A}$, where $V_T$ is the set of instance nodes, $V_F$ is the set of feature nodes, and $E$ is the set of edges between instance and feature nodes. For each non-zero entry $a_{ij}$, an edge $(i, j)$ is formed in $G$, where $i \in V_T$ and $j \in V_F$.




(a) Before removing hub nodes (b) After removing hub nodes

Fig. 2: A bipartite network after one iteration of Algorithm 2, where a square indicates a feature node and a circle indicates an instance node. FastPI assigns the highest id to the feature node (id 7) and instance node (id 9), respectively. The nodes in spokes get the lowest ids and the GCC receives the remaining ids. We remove multiple hub nodes in each iteration of the algorithm to sufficiently shatter the graph.

Figure 1 depicts the degree distributions of instance and feature nodes in each bipartite network derived from the Amazon and RCV feature matrices, respectively. Note that the degree distributions of both bipartite networks are skewed, i.e., there are only a few high degree nodes. In network analysis, skewness of the degree distributions have been exploited to reorder association matrices for efficient analysis. In the network terminology, the high degree nodes are called *hub* nodes, or simply hubs, and the neighbor nodes of a hub node are called the *spoke* nodes, or simply spokes. There is no consistent threshold degree of a node for it to be considered a hub and often a relative proportion of high-degree nodes rather than an explicit threshold degree is used to select the hubs.

Previous works on real-world networks have shown that real-world networks can be shattered by removing sets of highest degree nodes (Kang and Faloutsos, 2011; Lim et al., 2014; Jung et al., 2016, 2017). That is, when a set of hubs is removed from a connected component, a non-trivial portion of the nodes, i.e., the *spokes*, form small disconnected components, while the majority of the nodes remain in a giant connected component. In figure 2, we show that this shattering property of real-world networks also applies to bipartite graphs formed from feature matrices. We apply the shattering property to our feature matrix reordering (Algorithm 2 of FastPI).

### 3.2 Matrix Reordering of FastPI

Given a bipartite network $G$ derived from feature matrix $\mathbf{A}$ (Definition 1), FastPI obtains permutation arrays $\pi_T : V_T \rightarrow \{1, \cdots, m\}$ for instance nodes and $\pi_F : V_F \rightarrow \{1, \cdots, n\}$ for feature nodes, such that the non-zero entries of the feature matrix are concentrated as seen in Figure 3(e).

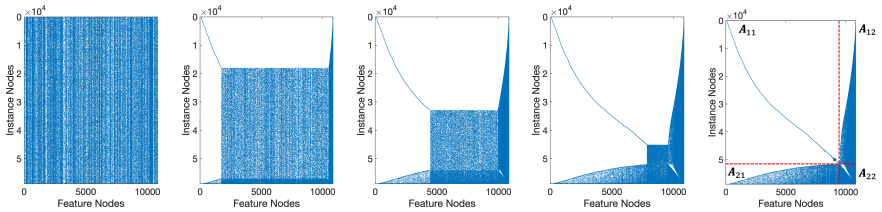
The high-level mechanism of the matrix reordering procedure is summarized in Algorithm 2. The algorithm first selects hub instance and feature nodes at line 2 in the order of node degree; given a hub selection ratio $0 < k < 1$, it chooses $m_{\text{hub}} \leftarrow \lceil k \times |V_T| \rceil$ hub instance nodes and $n_{\text{hub}} \leftarrow \lceil k \times |V_F| \rceil$ hub feature nodes, respectively. Then, it removes the selected hubs at line 3, such that the given network is split into three parts: 1) hubs, 2) giant connected

**Algorithm 2:** Matrix Reordering of FastPI

**Input:** bipartite network $G = (V_T, V_F, E)$ derived from feature matrix $\mathbf{A}$ and hub selection ratio $k$

**Output:** permutation arrays $\pi_T : V_T \to \{1, \cdots, m\}$ and $\pi_F : V_F \to \{1, \cdots, n\}$

1: **repeat**
2: select $m_{\text{hub}} \leftarrow \lceil k \times |V_T| \rceil$ hubs in $V_T$ and $n_{\text{hub}} \leftarrow \lceil k \times |V_F| \rceil$ hubs in $V_F$, respectively
3: place the selected hubs in $V_T$ and $V_F$ to the end of $\pi_T$ and $\pi_F$, respectively; and remove them from $G$ to generate new graph $G'$
4: find the connected components in $G'$ using breadth first search; and place nodes belonging to each non-giant connected component at the beginning of $\pi_T$ and $\pi_F$, respectively
5: set $G = (V_T, V_F, E)$ to be the giant connected component (GCC) of $G'$
6: **until** the number of nodes in $V_T$ or $V_F$ of the GCC is smaller than current $m_{\text{hub}}$ or $n_{\text{hub}}$, respectively
7: **return** permutation arrays $\pi_T$ and $\pi_F$




(a) The original matrix (b) After thirty iterations (c) After eighty iterations (d) After 115 iterations (e) After the final (119) iteration

Fig. 3: The matrix reordering process of FastPI on the feature matrix of the Amazon dataset. (a) depicts the original matrix, (b-d) are the reordered matrix after several iterations in Algorithm 2, and (e) is the matrix after the final iteration. As shown in (e), the non-zero entries of the feature matrix are concentrated by the matrix reordering such that it is divided into four submatrices where $\mathbf{A}_{11}$ is a large and sparse rectangular block diagonal matrix.

component (GCC) (colored blue), and 3) spokes (colored green) to the hubs as shown in Figure 2.

After removing the hubs, we assign new nodes ids to each $\pi_T$ and $\pi_F$ according to their node types. In Figure 2, the initial number of instance and features nodes are $|V_T| = 9$ and $|V_F| = 7$. After line 3, the hub instance node gets the highest instance id 9 ($= |V_T|$), and the hub feature node gets the highest feature id 7 ($= |V_F|$). Note that those two hubs should be treated differently; the instance node id corresponds to a row index, and the feature node id corresponds to a column index in the feature matrix. At line 4, the nodes in spokes take the lowest ids as in Figure 2(b). The remaining ids are assigned to the GCC. The same procedure is recursively repeated on the new GCC at line 5.

Figure 3 depicts the matrix reordering in Algorithm 2 for the feature matrix of the Amazon dataset. The spy plot of the feature matrix before reordering is in Figure 3(a). Figures 3(b)∼3(d) shows the intermediate matrices of the reordering process. As iterations proceed, the non-zero entries of the feature matrix are concentrated at the bottom right corner of the feature matrix as

shown in Figure 3(e). The final reordered matrix can be divided into four submatrices, or blocks, where the top left submatrix is a large and sparse block diagonal matrix. More specifically, the top and left submatrix contains small rectangular blocks at the diagonal area, where those blocks are formed by the spokes nodes; e.g., in Figure 2(b), the feature node with id 1 and instance nodes with ids 1-3 are grouped to form a tiny rectangular block on the diagonal area of the submatrix.

## 3.3 Incremental SVD Computation of FastPI

The reordered matrix $\mathbf{A}$ is divided into $\mathbf{A} = \begin{bmatrix} \mathbf{A}_{11} & \mathbf{A}_{12} \\ \mathbf{A}_{21} & \mathbf{A}_{22} \end{bmatrix}$, where $\mathbf{A}_{11} \in \mathbb{R}^{m_1 \times n_1}$, $\mathbf{A}_{12} \in \mathbb{R}^{m_1 \times n_2}$, $\mathbf{A}_{21} \in \mathbb{R}^{m_2 \times n_1}$, and $\mathbf{A}_{22} \in \mathbb{R}^{m_2 \times n_2}$. Note that $m_1$ and $n_1$ are the number of spoke instance and feature nodes, respectively. $m_2$ and $n_2$ are the number of hub instance and feature nodes, respectively.

### *3.3.1 SVD Computation for* $\mathbf{A}_{11}$

This step computes the low-rank SVD result of $\mathbf{A}_{11}$. Note that $\mathbf{A}_{11}$ is large and sparse, where many but small rectangular blocks are located at the diagonal area of $\mathbf{A}_{11}$ as shown in Figure 3(e). In this case, SVD result of $\mathbf{A}_{11}$ is efficiently obtained by computing SVD of each small block in $\mathbf{A}_{11}$ instead of performing SVD on the whole submatrix. For $i$th-block $\mathbf{A}_{11}^{(i)} \in \mathbb{R}^{m_{1i} \times n_{1i}}$, suppose $\mathbf{U}^{(i)}\mathbf{\Sigma}^{(i)}\mathbf{V}^{(i)\top}$ is the low-rank approximated SVD with the target rank $s_i = \lceil \alpha n_{1i} \rceil$ (let $m_{1i} > n_{1i}$ without loss of generality). Then, the SVD result of $\mathbf{A}_{11}$ is as follows:

$$\begin{aligned} \mathbf{U}_{m_1 \times s}\mathbf{\Sigma}_{s \times s}\mathbf{V}^{\top}_{s \times n_1} =& \text{bdiag}(\mathbf{U}^{(1)}, \cdots, \mathbf{U}^{(B)}) \times \\ & \text{bdiag}(\mathbf{\Sigma}^{(1)}, \cdots, \mathbf{\Sigma}^{(B)}) \times \text{bdiag}(\mathbf{V}^{(1)\top}, \cdots, \mathbf{V}^{(B)\top}) \end{aligned} \tag{1}$$

where $B$ is the number of blocks and bdiag$(\cdot)$ is the function returning a rectangular block diagonal matrix with a valid block sequence. The obtained rank is $s = \sum_{i=1}^{B} \alpha \lceil n_{1i} \rceil \approx \lceil \alpha n_1 \rceil$. Note that this is also a valid SVD result since $\mathbf{U}_{m_1 \times s}$ and $\mathbf{V}^{\top}_{s \times n_1}$ are orthogonal matrices, and $\mathbf{\Sigma}_{s \times s}$ is diagonal, which follows the definition of SVD (Strang, 2006).

### *3.3.2 Incremental Update of the SVD result*

The next step is to obtain the SVD result for $[\mathbf{A}_{11}; \mathbf{A}_{21}]$, where ';' indicates a vertical concatenation. The SVD of $[\mathbf{A}_{11}; \mathbf{A}_{21}]$ is calculated by incremental SVD (Brand, 2003; Ross et al., 2008) given the SVD of $\mathbf{A}_{11}$. The derivation for this incremental computation with the given target rank $s = \lceil \alpha n_1 \rceil$ is the followings:

$$
\begin{aligned}
\begin{bmatrix}\mathbf{A}_{11}\\ \mathbf{A}_{21}\end{bmatrix} &\simeq \begin{bmatrix}\mathbf{U}_{m_1\times s}\mathbf{\Sigma}_{s\times s}\mathbf{V}_{s\times n_1}^{\top}\\ \mathbf{A}_{21}\end{bmatrix}\\
&= \begin{bmatrix}\mathbf{U}_{m_1\times s}\mathbf{O}_{m_1\times m_2}\\ \mathbf{O}_{m_2\times s}\mathbf{I}_{m_2\times m_2}\end{bmatrix}\begin{bmatrix}\mathbf{\Sigma}_{s\times s}\mathbf{V}_{s\times n_1}^{\top}\\ \mathbf{A}_{21}\end{bmatrix}\\
&\simeq \begin{bmatrix}\mathbf{U}_{m_1\times s}\mathbf{O}_{m_1\times m_2}\\ \mathbf{O}_{m_2\times s}\mathbf{I}_{m_2\times m_2}\end{bmatrix}\underbrace{\tilde{\mathbf{U}}_{(s+m_2)\times s}\tilde{\mathbf{\Sigma}}_{s\times s}\tilde{\mathbf{V}}_{s\times n_1}^{\top}}_{\text{Low-rank approximation with } s}\\
&= \mathbf{U}_{m\times s}\mathbf{\Sigma}_{s\times s}\mathbf{V}_{s\times n_1}^{\top}
\end{aligned}
\tag{2}
$$

where $\mathbf{\Sigma}_{s\times s} = \tilde{\mathbf{\Sigma}}_{s\times s}$, $\mathbf{V}_{s\times n_1}^{\top} = \tilde{\mathbf{V}}_{s\times n_1}^{\top}$, $\mathbf{O}$ is a zero matrix, and $\mathbf{I}$ is an identity matrix. Note that $\mathbf{U}_{m\times s} = \begin{bmatrix}\mathbf{U}_{m_1\times s} & \mathbf{O}_{m_1\times m_2}\\ \mathbf{O}_{m_2\times s} & \mathbf{I}_{m_2\times m_2}\end{bmatrix}\tilde{\mathbf{U}}_{(s+m_2)\times s}$ is orthogonal since the product of two orthogonal matrices is also orthogonal (Strang, 2006). Note also that any low-rank SVD algorithm can be used for this purpose; we use frPCA (Feng et al., 2018) for a given low target rank ($r < \lceil 0.3n \rceil$ used), and the standard SVD otherwise since frPCA is optimized for very low ranks, and thus it is too slow for handling high ranks.

The final step is to incrementally update the SVD result in equation (2) for $\mathbf{T} = [\mathbf{A}_{12}; \mathbf{A}_{22}]$ with $r = \lceil \alpha n \rceil$ as follows:

$$
\begin{aligned}
\begin{bmatrix}\mathbf{A}_{11}\mathbf{A}_{12}\\ \mathbf{A}_{21}\mathbf{A}_{22}\end{bmatrix} &\simeq \begin{bmatrix}\mathbf{U}_{m\times s}\mathbf{\Sigma}_{s\times s}\mathbf{V}_{s\times n_1}^{\top}\mathbf{T}\end{bmatrix}\\
&= \begin{bmatrix}\mathbf{U}_{m\times s}\mathbf{\Sigma}_{s\times s} & \mathbf{T}\end{bmatrix}\begin{bmatrix}\mathbf{V}_{s\times n_1}^{\top}\mathbf{O}_{s\times n_2}\\ \mathbf{O}_{n_2\times n_1}\mathbf{I}_{n_2\times n_2}\end{bmatrix}\\
&= \underbrace{\tilde{\mathbf{U}}_{m\times r}\tilde{\mathbf{\Sigma}}_{r\times r}\tilde{\mathbf{V}}_{r\times(s+n_2)}^{\top}}_{\text{Low-rank approximation with } r}\begin{bmatrix}\mathbf{V}_{s\times n_1}^{\top}\mathbf{O}_{s\times n_2}\\ \mathbf{O}_{n_2\times n_1}\mathbf{I}_{n_2\times n_2}\end{bmatrix}\\
&= \mathbf{U}_{m\times r}\mathbf{\Sigma}_{r\times r}\mathbf{V}_{r\times n}^{\top}
\end{aligned}
\tag{3}
$$

where $\mathbf{\Sigma}_{r\times r} = \tilde{\mathbf{\Sigma}}_{r\times r}$, $\mathbf{U}_{m\times r} = \tilde{\mathbf{U}}_{m\times r}$, and $\mathbf{V}_{r\times n}^{\top} = \tilde{\mathbf{V}}_{r\times(s+n_2)}^{\top}\begin{bmatrix}\mathbf{V}_{s\times n_1}^{\top}\mathbf{O}_{s\times n_2}\\ \mathbf{O}_{n_2\times n_1}\mathbf{I}_{n_2\times n_2}\end{bmatrix}$ are also orthogonal.

### 3.4 Complexity Analysis

We analyze the computational complexity of Algorithm 1 in the following lemma:

**Lemma 1 (Computational Complexity of FastPI)** *Given a feature matrix $\mathbf{A} \in \mathbb{R}^{m\times n}$ and a target rank $r = \lceil \alpha n \rceil$, where $\alpha$ is the target rank ratio, the computational complexity of* FastPI *is $O(mr^2 + n_1r^2 + mn_2r + m_2n_1r + (\sum_{i=1}^{B} m_{1i}n_{1i}s_i) + T(m\log(m) + |\mathbf{A}|))$ prior to the final pseudoinverse construction (line 5 in Algorithm 1), where $m_1$ and $n_1$ are the number of spoke*

*instance and feature nodes, respectively, $m_2$ and $n_2$ are the number of hub instance and feature nodes, respectively, $B$ is the number of rectangular blocks, $m_{1i}$ and $n_{1i}$ are the height and the width of each rectangular block in $\mathbf{A}_{11}$, respectively, $s_i = \lceil \alpha n_{1i} \rceil$ is the target rank of $i$-th block, $|\mathbf{A}|$ is the number of non-zeros of $\mathbf{A}$, and $T$ is the number of iterations in Algorithm 2.*

*Proof We summarize the complexity of each step of Algorithm 1 in Table 2. For this proof, we use the traditional complexity of the low-rank approximation as describe in (Gu and Eisenstat, 1996; Halko et al., 2011); for matrix $\mathbf{A} \in \mathbb{R}^{p \times q}$, the low-rank approximation takes $O(pqk)$ time with target rank $k$. For a detailed comparison, we omit the cost of the final pseudoinverse construction (line 5 in Algorithm 1) because all SVD based methods should perform the construction as a common step. The complexity of each step of the algorithm is proved as follows:*

- *Line 1: for each iteration,* FastPI *sorts the degrees of instance and feature nodes; thus, it requires up to $O(m \log(m))$ since $m > n$. Then, it searches connected components in $G'$ using the breadth first search (BFS) algorithm in $O(|\mathbf{A}|)$ indicating the number of edges in the network. Hence, each iteration demands $O(m \log(m) + |\mathbf{A}|)$ time.*
- *Line 2:* FastPI *computes the low-rank approximated SVD of each rectangular block in $O(m_{1i} n_{1i} s_i)$ with target rank $s_i = \lceil \alpha n_{1i} \rceil$; thus, it is $O(\sum_{i=1}^{B} m_{1i} n_{1i} s_i)$.*
- *Line 3: in equation (2), the low-rank approximation takes $O((m_2+s)n_1 s) = O(m_2 n_1 s + n_1 s^2)$ time. The matrix multiplication for $\mathbf{U}_{m \times s}$ takes $O(m_1 s^2)$ as follows:*

$$\mathbf{U}_{m \times s} = \begin{bmatrix} \mathbf{U}_{m_1 \times s} & \mathbf{O}_{m_1 \times m_2} \\ \mathbf{O}_{m_2 \times s} & \mathbf{I}_{m_2 \times m_2} \end{bmatrix} \tilde{\mathbf{U}}_{(s+m_2) \times s} = \begin{bmatrix} \mathbf{U}_{m_1 \times s} \tilde{\mathbf{U}}_{s \times s} \\ \tilde{\mathbf{U}}_{m_2 \times s} \end{bmatrix}$$

*where $\tilde{\mathbf{U}}_{(s+m_2) \times s} = \begin{bmatrix} \tilde{\mathbf{U}}_{s \times s} \\ \tilde{\mathbf{U}}_{m_2 \times s} \end{bmatrix}$. Since $s \leq r$, it is bounded by $O(m_1 r^2 + n_1 r^2 + m_2 n_1 r)$ where $s = \lceil \alpha n_1 \rceil$ and $r = \lceil \alpha n \rceil$ and $n_1 \leq n$.*

- *Line 4: in equation (3), the low-rank approximation takes $O(m(n_2+s)r) = O(mn_2 r + msr)$, and the matrix multiplication takes $O(n_1 s^2)$ as follows:*

$$\mathbf{V}^{\top}_{r \times n} = \tilde{\mathbf{V}}^{\top}_{r \times (s+n_2)} \begin{bmatrix} \mathbf{V}^{\top}_{s \times n_1} & \mathbf{O}_{s \times n_2} \\ \mathbf{O}_{n_2 \times n_1} & \mathbf{I}_{n_2 \times n_2} \end{bmatrix} = \begin{bmatrix} \tilde{\mathbf{V}}^{\top}_{r \times s} \mathbf{V}^{\top}_{s \times n_1} & \tilde{\mathbf{V}}^{\top}_{r \times n_2} \end{bmatrix}$$

*where $\tilde{\mathbf{V}}^{\top}_{r \times (s+n_2)} = \begin{bmatrix} \tilde{\mathbf{V}}^{\top}_{r \times s} \tilde{\mathbf{V}}^{\top}_{r \times n_2} \end{bmatrix}$. Hence, it is $O(n_1 r^2 + mr^2 + mn_2 r)$ since $s \leq r$.* □

The dominant factor is $mr^2$ in the complexity of the analysis of FastPI. As described in Section 2, $O(mr^2)$ is faster than $O(mnr)$ of traditional methods (Gu and Eisenstat, 1996). FastPI exhibits similar complexity to Randomized SVD, the state-of-the-art method with complexity of $O(mr^2 + nr^2 + mn \log(r))$ (Halko et al., 2011). However, the actual running time of the Randomized SVD is slower than that of the FastPI for a reasonably high rank (see Figure 6). The reason is that Randomized SVD is based on oversampling

Table 2: Computational complexity of each step of FastPI (Algorithm 1).

| Line | Task | Computational Complexity |
|---|---|---|
| 1 | Reorder $\mathbf{A}$ using Algorithm 2 | $O(T(m\log(m)+\|\mathbf{A}\|))$ |
| 2 | Compute SVD of $\mathbf{A}_{11}$ | $O(\sum_{i=1}^{B} m_{1i}n_{1i}s_i)$ |
| 3 | Update the SVD result for $\mathbf{A}_{21}$ | $O(m_1r^2+n_1r^2+m_2n_1r)$ |
| 4 | Update the SVD result for $\mathbf{T}=[\mathbf{A}_{12};\mathbf{A}_{22}]$ | $O(n_1r^2+mr^2+mn_2r)$ |
| **Total** | $O(mr^2+n_1r^2+mn_2r+m_2n_1r+(\sum_{i=1}^{B} m_{1i}n_{1i}s_i)+T(m\log(m)+\|\mathbf{A}\|))$ | |

Table 3: Dataset statistics. $m$ is the number of instances (rows), and $n$ is the number of features (columns) of feature matrix $\mathbf{A}$. $L$ is the number of labels of label matrix $\mathbf{Y}$. $|\mathbf{A}|$ is the number of non-zero entries of $\mathbf{A}$, and $\text{sp}(\mathbf{A})$ is the sparsity of $\mathbf{A}$ defined in Section 4.1. $k$ is the hub selection ratio of Algorithm 2. $m_2$ and $n_2$ are the number of instance and feature hub nodes, respectively.

| Dataset | $m$ | $n$ | $L$ | $\|\mathbf{A}\|$ | $\text{sp}(\mathbf{A})$ | $\text{sp}(\mathbf{Y})$ | $k$ | $m_2$ | $n_2$ |
|---|---|---|---|---|---|---|---|---|---|
| **Amazon** | 59,312 | 10,195 | 13,330 | 167,015 | 0.9997 | 0.9996 | 0.01 | 4,158 | 714 |
| **RCV** | 62,385 | 4,724 | 2,456 | 466,675 | 0.9984 | 0.9981 | 0.01 | 8,112 | 624 |
| **Eurlex** | 15,539 | 5,000 | 3,993 | 3,684,773 | 0.9525 | 0.9987 | 0.01 | 8,736 | 2,800 |
| **Bibtex** | 7,395 | 1,836 | 159 | 507,746 | 0.9626 | 0.9849 | 0.01 | 5,180 | 1,330 |

technique (see the details in Section 4.1); due to this point, Randomized SVD has a higher coefficient for the dominant factor compared to FastPI (i.e., Randomized SVD requires $4mr^2$ operations while FastPI needs $mr^2$ ones for the same $r$).

Note that for a detailed comparison, we have omitted the cost of the final pseudoinverse construction (line 5 in Algorithm 1), which is a universal step in all SVD based methods.

## 4 Experiment

In this section, we aim to answer the following questions from experiments:

- **Q1. Reconstruction error (Section 4.2).** Does FastPI correctly produce low-rank SVD results in terms of reconstruction error?
- **Q2. Accuracy (Section 4.3).** How accurate is the pseudoinverse obtained by FastPI for the multi-label linear regression task compared to other methods?
- **Q3. Efficiency (Section 4.4).** How quickly does FastPI compute the approximate pseudoinverse of sparse feature matrices compared to state-of-the-art methods?

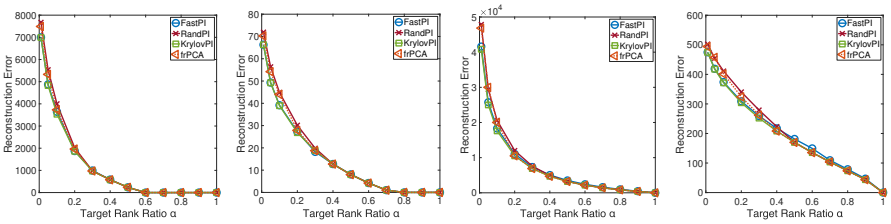
### 4.1 Experimental Setting

**Datasets.** We use four real-world multi-label datasets, and their statistics are summarized in Table 3. The Bibtex dataset is from a social bookmarking system, where each instance consists of features from a bibtex item and labels are tags in the system (Katakis et al., 2008). The Eurlex dataset is from documents about European Union law, where each instance is formed by word features from a document and labels indicate categories (Mencia and Fürnkranz, 2008). The RCV dataset is randomly sampled from an archive of newswire stories made available by Reuters, Ltd., where each instance consists of features from a document, and labels are categories (Lewis et al., 2004). The Amazon dataset is randomly sampled from a set of reviews of Amazon, where each instance is formed by word features of a review and labels are items (McAuley and Leskovec, 2013). In Table 3, sparsity($\mathbf{A}$) indicates the sparsity of feature matrix $\mathbf{A} \in \mathbb{R}^{m \times n}$, which is defined as $\text{sp}(\mathbf{A}) = 1 - |\mathbf{A}|/(mn)$, where $|\mathbf{A}|$ is the number of non-zero entries in $\mathbf{A}$.

**Machine and Implementation.** We use a single thread in a machine with an Intel Xeon E5-2630 v4 2.2GHz CPU and 512GB RAM. All tested methods, including our proposed FastPI, are implemented using MATLAB which provides a state-of-the-art linear algebra package.

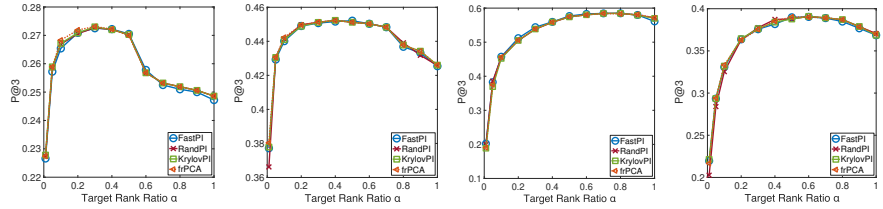
**Competing Methods.** We use the following methods as the competitors of FastPI:

- **RandPI** is based on Randomized SVD (Halko et al., 2011), the state-of-the-art low-rank SVD method for target rank $r = \lceil \alpha n \rceil$ using an oversampling technique for numerical stability. The main procedure of RandPI is as follows:
  - Step 1: It generates a Gaussian over-sampled random matrix $\mathbf{X}_{n \times 2r}$ for constructing randomized data matrix $\mathbf{B}_{m \times 2r} = \mathbf{A}\mathbf{X}_{n \times 2r}$.
  - Step 2: It finds a matrix $\mathbf{Q}_{m \times 2r}$ with orthonormal columns as a proxy of an orthogonal matrix from $\mathbf{B}_{m \times 2r}$ satisfying $\mathbf{A} \simeq \mathbf{Q}\mathbf{Q}^\top \mathbf{A}$.
  - Step 3: It constructs $\mathbf{Y}_{2r \times n} = \mathbf{Q}^\top_{2r \times m} \mathbf{A}$ and compute SVD of $\mathbf{Y}_{2r \times n} \simeq \tilde{\mathbf{U}}_{2r \times r} \mathbf{\Sigma}_{r \times r} \mathbf{V}^\top_{r \times n}$.
  - Step 4: It computes $\mathbf{U}_{m \times r} = \mathbf{Q}_{m \times 2r} \tilde{\mathbf{U}}_{2r \times r}$.
- **KrylovPI** is based on a Krylov subspace iterative method for the low-rank SVD method (Baglama and Reichel, 2005). KrylovPI is specialized for computing a few singular values and vectors on a sparse matrix (used in `svds` of MATLAB).
- **frPCA** (Feng et al., 2018) combines the randomized SVD and a power iteration method so that it controls trade-off between running time and accuracy for computing SVD of sparse data. This also exploits LU decomposition in the power iteration to improve accuracy. We set the oversampling parameter $s$ to 5 and the number of iterations to 11 as in (Feng et al., 2018).




(a) Amazon (b) RCV (c) Eurlex (d) Bibtex

Fig. 4: Reconstruction error of the SVD result of each method varying target rank ratio $\alpha$ (Section 4.2). Note that the error of our FastPI is almost the same as that of KrylovPI, indicating that our method computes the SVD result near optimally for any $\alpha$.




(a) Amazon (b) RCV (c) Eurlex (d) Bibtex

Fig. 5: Accuracy of the multi-label linear regression task (Application 1) in terms of P@3 varying target rank ratio $\alpha$ (Section 4.3). Note that accuracies of all tested methods are almost the same for each $\alpha$, implying that FastPI accurately computes the approximate pseudoinverse as other methods.

## 4.2 Reconstruction Error

We analyze the reconstruction error of the SVD result computed by each method to check if it computes the SVD result accurately. The reconstruction error of the SVD result $\mathbf{U}_{m\times r}\mathbf{\Sigma}_{r\times r}\mathbf{V}^\top_{r\times n}$ from the original matrix $\mathbf{A}$ is defined as $\|\mathbf{A} - \mathbf{U}_{m\times r}\mathbf{\Sigma}_{r\times r}\mathbf{V}^\top_{r\times n}\|_\mathrm{F}$, where $r = \lceil \alpha n \rceil$ is the target rank and $\|\cdot\|_\mathrm{F}$ is the Frobenius norm. We measure the error of each method varying the target rank ratio $\alpha$ from 0.01 to 1.0, respectively.

Figure 4 demonstrates the reconstruction error of all tested methods. The error of our FastPI is slightly better than that of RandPI, which is the state-of-the-art SVD for low-rank SVD computation, especially when $\alpha$ is low. Another point is that the reconstruction error of our method is almost the same as that of KrylovPI. These show that reconstruction-wise, our method is near-optimal given rank ratio $\alpha$ for the SVD computation.

## 4.3 Accuracy of Multi-label Linear Regression

We examine the quality of the approximate pseudoinverse by measuring the predictive performance of each method in the multi-label linear regression task as described in the Application 1. For each experiment, we randomly

split a multi-label dataset into a training set (90%) and a test set (10%); and compute the pseudoinverse on the training set varying the target rank ratio $\alpha$ from 0.01 to 1.0. Note that in the multi-label datasets, each instance has only a few positive (1) labels, i.e., the label matrix $\mathbf{Y}$ is sparse as shown in Table 3. Therefore, it is important to focus on the accurate prediction of the few positive labels. Due to this reason, many researchers (Chen and Lin, 2012; Prabhu and Varma, 2014; Yu et al., 2014) have evaluated the performance of this task using ranking based measures such as top-$k$ precision, denoted by P@$k$, based on predicted scores. We also measure P@$k$ as the accuracy of this task on the test set. Let $\mathbf{y} \in \{0,1\}^L$ be a ground truth label vector and $\hat{\mathbf{y}} \in \mathbb{R}^L$ be its predictive score vector, where $L$ is the number of labels. Then, $\text{P@}k = \frac{1}{k} \sum_{l \in \text{rank}_k(\hat{\mathbf{y}})} \mathbf{y}_l$, where $\text{rank}_k(\hat{\mathbf{y}})$ returns the $k$ largest indices of $\hat{\mathbf{y}}$ ranked in the descending order.

Figure 5 demonstrates the accuracy of each method for the task in terms of P@3. As expected, the accuracy depends on the rank ratio and there is little difference between the compared methods, which serves to verify our proposed FastPI is correctly derived. However, an interesting observation is made about the appropriate rank ratio for uses in multi-linear regression tasks: accuracy plots are curved indicating that there are underfittings when $\alpha$ are too small, and overfittings when $\alpha$ are too large. This implies that such low-rank approximation with a relatively large rank is effective in the viewpoint of the machine learning application.

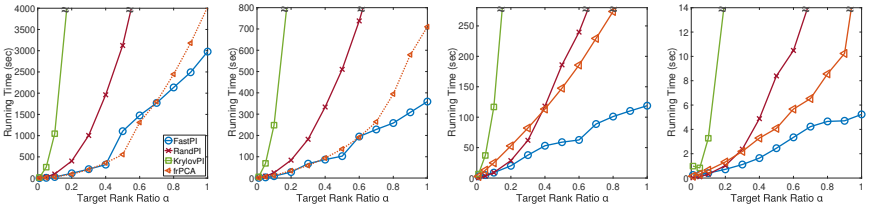
### 4.4 Computational Performance

We investigate the computational performance of each method in terms of running time. For each experiment, we measure the wall-clock time of FastPI, RandPI, and KrylovPI varying the target rank ratio $\alpha$ from 0.01 to 1.0.

Figure 6 demonstrates the running time of methods on each dataset. First, the running time of KrylovPI, an iterative method, skyrockets since it requires more iterations for convergence as $\alpha$ increases. KrylovPI is specialized for computing a very few largest or smallest singular values and vectors on a large and sparse matrix (e.g., $\alpha = 0.01$). Thus, it is impractical to use KrylovPI for obtaining relatively high-rank SVD results as shown in Figure 6.

We next compare FastPI to RandPI. As shown in Figure 6, FastPI is faster than RandPI over all datasets. Especially, the performance of RandPI becomes much worse as the target rank $r = \lceil \alpha n \rceil$ increases. The main reason is because of the oversampling technique of RandPI, i.e., it needs to perform several matrix operations on $m \times 2r$ matrices (see the details in Section 4.1). If $r$ is very small, their computations are efficient. However, if $r$ is large, and it is close to $n$, then RandPI needs to handle up to $m \times 2n$ matrices whose size is twice the size of original, thereby slowing down the execution speed for large $\alpha$ or high rank.

Finally, we look into the comparison between FastPI and frPCA. Overall, the performance of FastPI is better than that of frPCA, especially in the




(a) Amazon (b) RCV (c) Eurlex (d) Bibtex

Fig. 6: Computational performance in terms of running time varying target rank ratio $\alpha$ (Section 4.4).

Eurlex and Bibtex datasets. In the Amazon and RCV datasets, although the running time of FastPI is similar to or slightly slower than frPCA for $r \leq 0.6$, FastPI is faster than frPCA for $r > 0.6$. These results indicate that FastPI is competitive with frPCA for low ranks, and it is more efficient than frPCA for high ranks.

## 5 Conclusion

In this paper, we have shown how feature matrix reordering can speed up the approximate pseudoinverse calculation faster than the state-of-the-art low-rank approximation method for computing pseudoinverses. Our proposed approach FastPI (Fast PseudoInverse) is based on a crucial observation that many real-world feature matrices are considerably sparse and skewed, which have the possibility of being reordered. FastPI reorders the feature matrix such that the reordered matrix contains a large and sparse block diagonal matrix whose SVD is easily computed. FastPI then efficiently computes the approximate pseudoinverse through incremental low-rank SVD updates. We applied the FastPI on multi-label linear regression problem, a type of machine learning problem. Our experiments demonstrate that our FastPI computes SVD based pseudoinverse quickly for sufficiently large ranks compared to other methods.

## References

Baglama J, Reichel L (2005) Augmented implicitly restarted lanczos bidiagonalization methods. SIAM Journal on Scientific Computing 27(1):19–42

Ben-Israel A, Greville TN (2003) Generalized inverses: theory and applications, vol 15. Springer Science & Business Media

Brand M (2003) Fast online svd revisions for lightweight recommender systems. In: Proceedings of the 2003 SIAM international conference on data mining, SIAM, pp 37–46

Chen YN, Lin HT (2012) Feature-aware label space dimension reduction for multi-label classification. In: Advances in Neural Information Processing Systems, pp 1529–1537

Feng X, Xie Y, Song M, Yu W, Tang J (2018) Fast randomized pca for sparse data. In: Asian Conference on Machine Learning, pp 710–725

Golub GH, Van Loan CF (2012) Matrix computations, vol 3. JHU press

Gu M, Eisenstat SC (1996) Efficient algorithms for computing a strong rank-revealing qr factorization. SIAM Journal on Scientific Computing 17(4):848–869

Guo P, Zhao D, Han M, Feng S (2019) Pseudoinverse learners: New trend and applications to big data. In: INNS Big Data and Deep Learning conference, Springer, pp 158–168

Halko N, Martinsson PG, Tropp JA (2011) Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. SIAM review 53(2):217–288

He D, Kuhn D, Parida L (2016) Novel applications of multitask learning and multiple output regression to multiple genetic trait prediction. Bioinformatics 32(12):i37–i43

Horata P, Chiewchanwattana S, Sunat K (2013) Robust extreme learning machine. Neurocomputing 102:31–44

Jung J, Shin K, Sael L, Kang U (2016) Random walk with restart on large graphs using block elimination. ACM Transactions on Database Systems (TODS) 41(2):1–43

Jung J, Park N, Sael L, Kang U (2017) BePI: Fast and memory-efficient method for billion-scale random walk with restart. In: ACM International Conference on Management of Data (SIGMOD), ACM Press, Raleigh, North Carolina, USA.

Kang U, Faloutsos C (2011) Beyond 'caveman communities': Hubs and spokes for graph compression and mining. In: 11th IEEE International Conference on Data Mining, ICDM 2011, Vancouver, BC, Canada, December 11-14, 2011, pp 300–309

Katakis I, Tsoumakas G, Vlahavas I (2008) Multilabel text classification for automated tag suggestion. In: Proceedings of the ECML/PKDD, vol 18

Lewis DD, Yang Y, Rose TG, Li F (2004) Rcv1: A new benchmark collection for text categorization research. Journal of machine learning research 5(Apr):361–397

Lim Y, Kang U, Faloutsos C (2014) Slashburn: Graph compression and mining beyond caveman communities. IEEE Trans Knowl Data Eng 26(12):3077–3089, DOI 10.1109/TKDE.2014.2320716, URL `http://doi.ieeecomputersociety.org/10.1109/TKDE.2014.2320716`

McAuley J, Leskovec J (2013) Hidden factors and hidden topics: understanding rating dimensions with review text. In: Proceedings of the 7th ACM conference on Recommender systems, ACM, pp 165–172

Mencia EL, Fürnkranz J (2008) Efficient pairwise multilabel classification for large-scale problems in the legal domain. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases, Springer, pp 50–65

Prabhu Y, Varma M (2014) Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In: Proceedings of the 20th ACM SIGKDD

international conference on Knowledge discovery and data mining, ACM, pp 263–272

Ross DA, Lim J, Lin RS, Yang MH (2008) Incremental learning for robust visual tracking. International journal of computer vision 77(1-3):125–141

Spyromitros-Xioufis E, Tsoumakas G, Groves W, Vlahavas I (2016) Multi-target regression via input space expansion: treating targets as inputs. Machine Learning 104(1):55–98

Strang G (2006) Linear algebra and its applications. Thomson, Brooks/Cole

Trefethen LN, Bau III D (1997) Numerical linear algebra, vol 50. Siam

Xu B, Guo P (2018) Pseudoinverse learning algorithom for fast sparse autoencoder training. In: 2018 IEEE Congress on Evolutionary Computation (CEC), IEEE, pp 1–6

Yu HF, Jain P, Kar P, Dhillon I (2014) Large-scale multi-label learning with missing labels. In: International conference on machine learning, pp 593–601