Fast Pseudoinverse method for sparse feature matrices

Grigory Yaremenko, Iskander Akmanov, Kundyz Onlabek, Nina Aleskerova

December 2020

Problem statement

A **pseudoinverse** is a generalized inverse method for all types of matrices:

- Plays a crucial role in obtaining best-fit solutions to linear systems;
- The most widely applied pseudoinverse is the Moore-Penrose inverse;
- Applications were limited to small data due to their high computational complexity.

Pseudoinverse and SVD

In machine learning models, a training data is represented as a feature matrix $A \in \mathbb{R}^{m \times n}$.

The **Moore-Penrose inverse** can be solved using SVD. If r is the rank of A:

$$A^{\dagger} = V_{n \times r} \Sigma_{r \times r} U^{\top}_{r \times m}.$$

For a given target rank r a best approximate pseudoinverse:

$$A^{\dagger} \approx V_{n \times r} \Sigma_{r \times r} U^{\top}_{r \times m}.$$

- SVD calculations are impractical for large matrices, the time complexity of a full-rank SVD is $min(O(n^2 m), O(nm^2))$.
- For low-rank approximation - $O(m \times n \times log(r) + (m + n)r^2)$.

The proposed solution

- **FastPI (Fast PseudoInverse) method** of Jinhong Jung and Lee Sael (2020).
- Algorithm for obtaining pseudoinverse efficiently and accurately based on sparse matrix reordering and incremental low-rank SVD.
- The main motivation comes from the fact that in the real world many matrices are highly sparse and skewed.

Target application of Pseudoinverse

Multi-label Linear Regression:
Given feature matrix $A \in \mathbb{R}^{m \times n}$ and label matrix $Y \in \mathbb{R}^{m \times L}$, where $m > n$, L is the number of labels, and each row of Y is a binary label vector of size L.

- The goal is to learn the parameter $Z \in \mathbb{R}^{n \times L}$ satisfying $AZ \simeq Y$ to estimate the score vector $\widehat{y} = Z^{\top} a$ for a new feature vector $a \in \mathbb{R}^{n}$.
- The solution is to minimize the least square error $\|AZ - Y\|_F^2$.
- The closed form solution $Z = A^{\dagger} Y$.

Fast PseudoInverse method

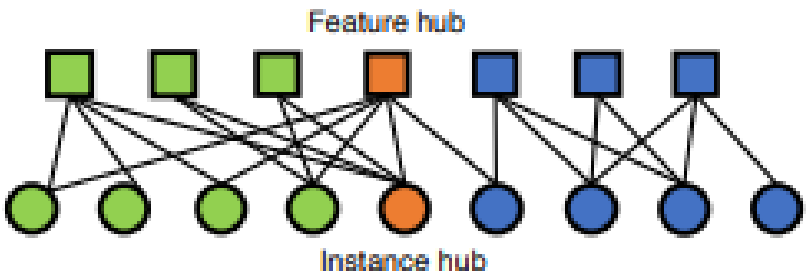
Main ideas of FastPI method for sparse matrices:

- Reorder $A \rightarrow \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$, where A_{11} - a large and sparse block diagonal submatrix whose SVD is easy-to-compute
- Compute the SVD for $A_{11} \approx U_{m_1 \times s} \Sigma_{s \times s} V^{\top}_{s \times n_1}$ with target rank $s = [\alpha n_1]$
- Incrementally update the SVD for A_{21}, $\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} \approx U_{m \times s} \Sigma_{s \times s} V^{\top}_{s \times n}$, with target rank $s = [\alpha n_1]$
- Incrementally update the SVD for $\begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix}$ with target rank $r = [\alpha n]$.
- Solve pseudoinverse $\rightarrow A^{\dagger}$

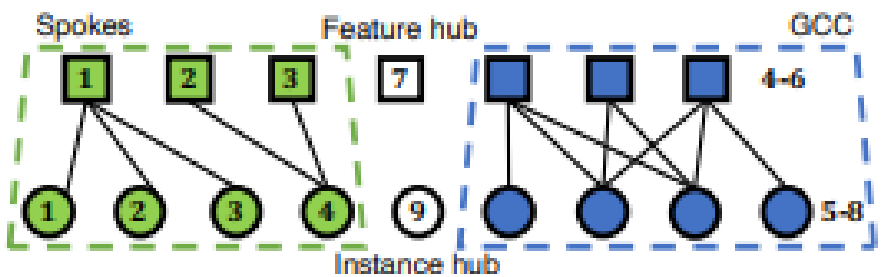
Matrix reordering

FastPI reordering algorithm: bipartite network $G = (V_I, V_F, E)$ derived from feature matrix A and hub selection ratio $0 < k < 1$.

- Select $m_{hub} = [k \times V_I]$ hubs in V_I and $n_{hub} = [k \times V_F]$ hubs in V_F
- Place the selected hubs in V_I and V_F to the end, and remove them from G to generate new graph G'
- Find the connected components in G', and place nodes belonging to each non-giant connected component at the beginning



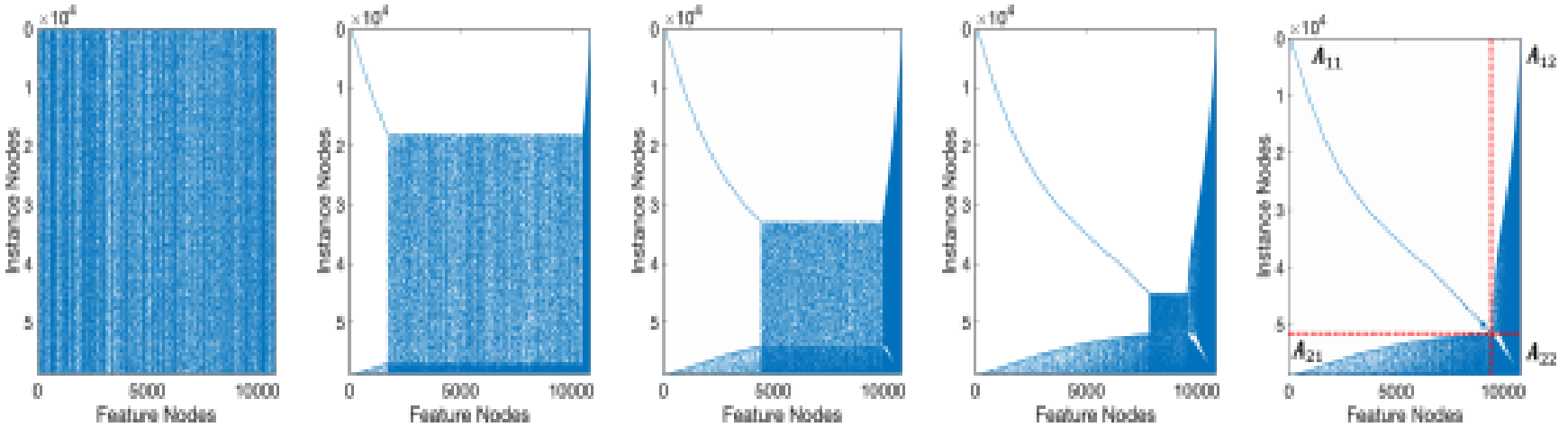
(a) Before removing hub nodes



(b) After removing hub nodes

Matrix reordering

- Set $G = (V_I, V_F, E)$ to be the giant connected component (GCC) of G'
- Repeat until the number of nodes in V_I or V_F of the GCC is smaller than m_{hub} or n_{hub}



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

Incremental SVD computation of FastPI

The reordered matrix $A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$, where
$A_{11} \in \mathbb{R}^{m_1 \times n_1}, A_{12} \in \mathbb{R}^{m_1 \times n_2}, A_{21} \in \mathbb{R}^{m_2 \times n_1}, A_{22} \in \mathbb{R}^{m_2 \times n_2}$.
SVD computation for A_{11}:

$$U_{m_1 \times s} \Sigma_{s \times s} V^{\top}_{s \times n_1} = bdiag(U^{(1)}, ..., U^{(B)}) \times bdiag(\Sigma^{(1)}, ..., \Sigma^{(B)}) \times \\ \times bdiag(V^{(1)\top}, ..., V^{(B)\top})$$

B - the number of blocks, the obtained rank $s = \sum_{i=1}^{B} \alpha n_{1i} \approx \alpha n_1$

Incremental SVD computation of FastPI

SVD for $\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix}$:

$$\begin{bmatrix} A_{11} \\ A_{21} \end{bmatrix} \approx \begin{bmatrix} U_{m_1 \times s} \Sigma_{s \times s} V^{\top}_{s \times n_1} \\ A_{21} \end{bmatrix} = \begin{bmatrix} U_{m_1 \times s} & O_{m_1 \times m_2} \\ O_{m_2 \times s} & I_{m_2 \times m_2} \end{bmatrix} \begin{pmatrix} \Sigma_{s \times s} V^{\top}_{s \times n_1} \\ A_{21} \end{pmatrix} =$$

$$= \begin{bmatrix} U_{m_1 \times s} & O_{m_1 \times m_2} \\ O_{m_2 \times s} & I_{m_2 \times m_2} \end{bmatrix} \tilde{U}_{(s+m_2) \times s} \tilde{\Sigma}_{s \times s} \tilde{V}^{\top}_{s \times n_1} = U_{m \times s} \Sigma_{s \times s} V^{\top}_{s \times n_1}$$

Finally, update the SVD for $T = \begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix}$:

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = U_{m \times s} \Sigma_{s \times s} V^{\top}_{s \times n_1} T = U_{m \times s} \Sigma_{s \times s} T \begin{bmatrix} V^{\top}_{s \times n_1} & O_{s \times n_2} \\ O_{n_2 \times n_1} & I_{n_2 \times n_2} \end{bmatrix} =$$

$$= \tilde{U}_{m \times r} \tilde{\Sigma}_{r \times r} \tilde{V}^{\top}_{r \times (s+n_2)} \begin{bmatrix} V^{\top}_{s \times n_1} & O_{s \times n_2} \\ O_{n_2 \times n_1} & I_{n_2 \times n_2} \end{bmatrix} = U_{m \times r} \Sigma_{r \times r} V^{\top}_{r \times n}$$

Computational complexity of FastPI

	Task	Computational complexity
1	Reorder A	$O(T(m \times log(m) + \|A\|))$
2	Compute SVD of A_{11}	$O(\sum_{i=1}^{B} m_{1i} n_{1i} s_i)$
3	Update the SVD result for A_{21}	$O(m_1 r^2 + n_1 r^2 + m_2 n_1 r)$
4	Update the SVD result for $\begin{bmatrix} A_{12} \\ A_{22} \end{bmatrix}$	$O(n_1 r^2 + m r^2 + m n_2 r)$

Total: $O(mr^2 + m_1 r^2 + n_1 r^2 + m n_2 r + m_2 n_1 r + \sum_{i=1}^{B} m_{1i} n_{1i} s_i + T(m \times log(m) + |A|))$

where $|A|$ - the number of non-zeros elements of A, T - the number of iterations in matrix reordering algorithm.

Computational performance

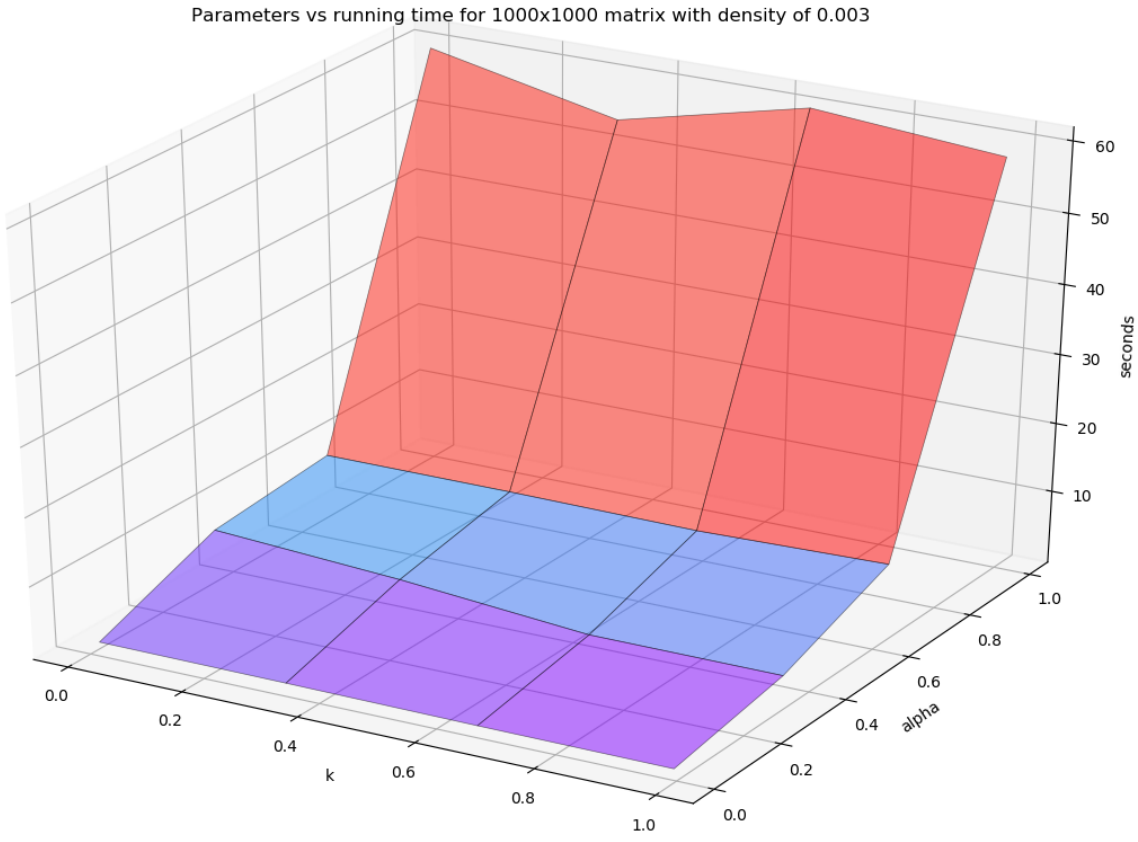


Figure: Plot of performance vs. rank ratio α and the number of rows/columns permuted at the same time k.

On density and usefulness of reordering

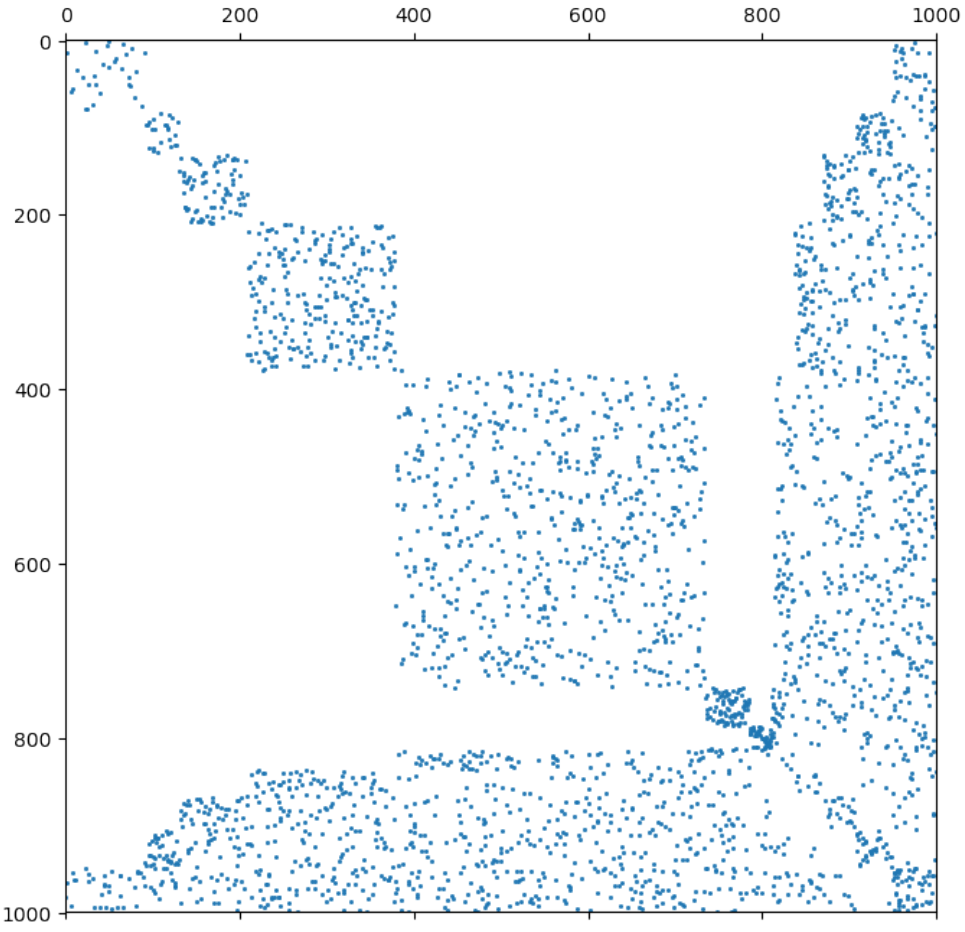


Figure: Sparsity pattern after reordering of a matrix with density of 0.003.

On density and usefulness of reordering

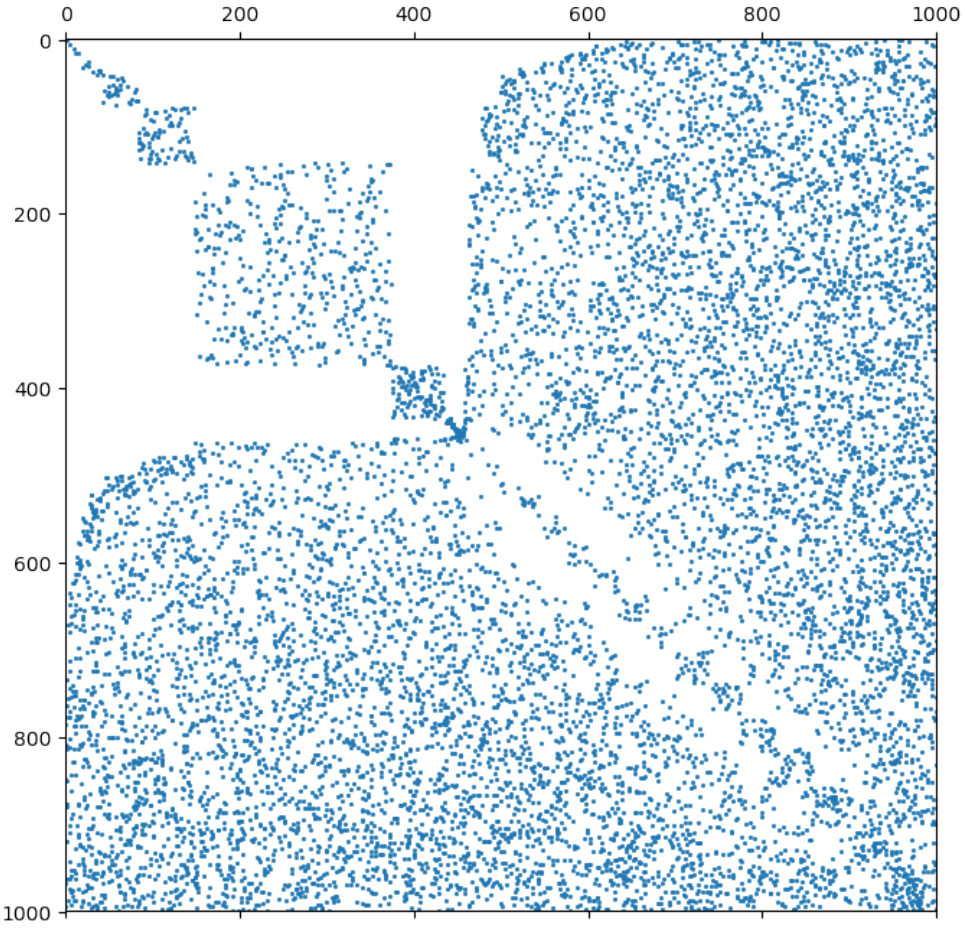


Figure: Sparsity pattern after reordering of a matrix with density of 0.01.

On density and usefulness of reordering

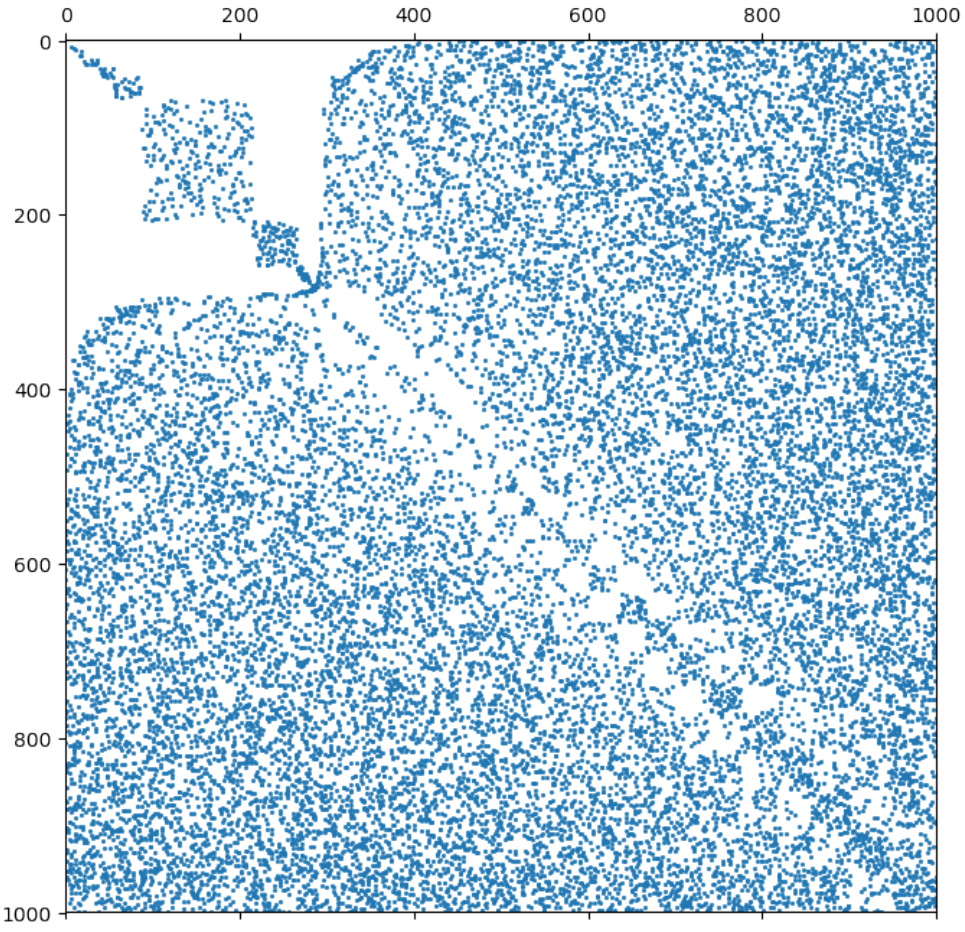


Figure: Sparsity pattern after reordering of a matrix with density of 0.02.

Reconstruction error

$$10^{-8} - 10^{-12}$$

- Highly depends on the matrix.
- In fact even more than it depends on parameters.

Conclusion

- Decent performance boost.
- Reordering has an insignificant computation time.
- Usefulness highly determined by density. An extra order of magnitude can instantaneously render the method useless.
- Stability requires further investigation.