Scaling Locally Linear Embedding

Yasuhiro Fujiwara[†‡*], Naoki Marumo[†], Mathieu Blondel[†], Koh Takeuchi[†], Hideaki Kim[†], Tomoharu Iwata[†], Naonori Ueda[†]

[†]NTT Communication Science Laboratories,
[‡]NTT Software Innovation Center,
[‡*]Osaka University

Introduction

- Locally Linear Embedding (LLE) is one of the popular approaches of nonlinear dimensionality reduction
- However, it requires high computational cost

- The goal in this study is to enhance the processing speed of LLE
- We propose a novel approach, *Ripple*, as a solution

LLE: overview

- LLE is an approach that can effectively reduce the dimensionality of multi-dimensional data[*1]
  - Effectively represents manifold (clustering structure) of a dataset by embedding

- 3 steps:
  1. k-NN graph
  2. Edge weight by regression
  3. Eigen decomposition based on adjacency matrix
     - Eigen vectors correspond to embedding

*1 Roweis et al., “Nonlinear Dimensionality Reduction by Locally Linear Embedding”, Science, 2000

LLE: step1 k-NN graph

- Use naive approach
  - Pick up data points (nodes) one by one
  - Compute top K based on Euclidean distance
  - Computation cost: $O(N^2 M)$
    - N: # data points, M: # dimensions

LLE: step2 edge weight

- Compute edge weights by regression (sum of edge weights is normalized to 1)
- Lagrange multiplier method
  - Regression error of node x_p

$$\varepsilon = \left\| \underbrace{x_p - \sum_{x_i \in \mathbb{N}[x_p]} \mathrm{w}[p,i] x_i}_{\text{regression}} \right\|^2 = \sum_{x_\mathrm{i}, x_\mathrm{j} \in \mathbb{N}[x_p]} G[\mathrm{i},\mathrm{j}]$$

$\mathbb{N}[x_p]$: top K data points of x_p

$\mathrm{G}[\mathrm{i},\mathrm{j}]$: $(x_p\text{-}x_\mathrm{i})(x_p\text{-}x_j)$

  - Edge weights are obtained from inverse of matrix G

$$\mathrm{w}[\mathrm{p},\mathrm{i}] = \sum_{j=1}^{K} G^{-1}[i,j] \Big/ \sum_{i=1}^{K} \sum_{j=1}^{K} G^{-1}[i,j]$$

$\mathrm{w}[\mathrm{p},\mathrm{i}]$: edge weight from x_i to x_p

  - Computation cost: $O(N(MK^2 + K^3))$
    - $O(MK^2)$ for computing matrix G
    - $O(K^3)$ for computing matrix G^{-1}

LLE: step 3 Eigen decomposition

- Minimize error of dimensionality reduction
  - Compute Eigen decomposition of matrix K obtained from adjacency matrix W of step 2

$$\mathrm{K} = (I - W)^T (I - W)$$

  - "Bottom (the smallest) " Eigen vectors of matrix K are the embedding to low dimensionalities
    - Not top (the highest) Eigen vectors
  - Computation cost: $O(N^3)$
    - Size of matrix K is $N \times N$

LLE: problem

- Impractical for large size of dataset
  - It needs high computation cost
  - Computation cost: $O(N^2M + N(MK^2 + K^3) + N^3)$
    - Especially Eigen decomposition needs high computation cost since it requires $O(N^3)$ time

Proposed method

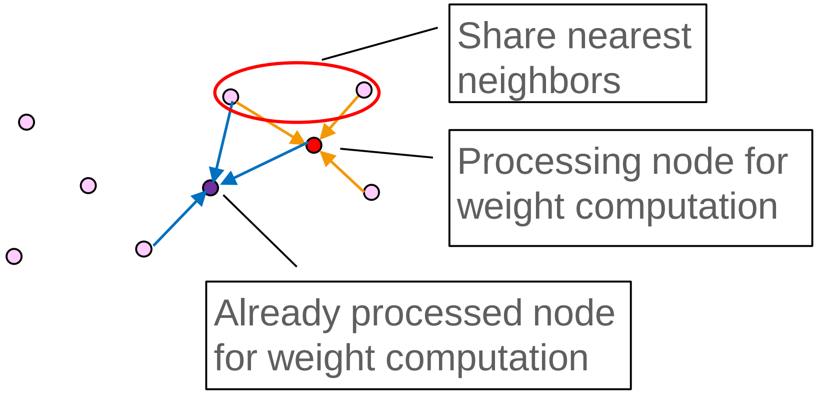
- We exploit 3 approaches
  - Incremental weight computation
    - Efficiently compute weights by using inner product of data points and the Woodbury formula
  - Improve lower bound of Euclidean distance
    - Enhance effectiveness of pruning by exploiting graph structure
    - Efficiently obtain top-k nodes
  - LU decomposition based Eigen decomposition
    - Compute eigenvectors “ascending” order of Eigen values
      - Power method compute eigenvectors in “descending” order
    - We avoid computing inverse matrix by LU decomposition
    - Significantly reduce computation cost of Eigen decomposition

Incremental weight computation

- Basic idea
  - Compute k-NN graph by picking up nodes one by one
  - Since such nodes share nearest neighbors, we uses the Wood bury formula to compute inverse matrix
  - But, each node has totally different matrix G_p

$$G_p[\mathrm{i},\mathrm{j}]: (x_p\text{-}x_\mathrm{i})(x_p\text{-}x_j)$$

  - We use matrix C_p instead of G_p that can share element in applying Lagrange multiplier method


Share nearest neighbors
Processing node for weight computation
Already processed node for weight computation

Incremental weight computation

- Let y $= \sum_{x_i \in \mathbb{N}[x_p]} W[p,i]x_i$, regression error ε

$$\varepsilon = \left\| x_p - \sum_{x_i \in \mathbb{N}[x_p]} W[p,i]x_i \right\|^2 = x_p(x_p)^T - 2xy^T + y(y)^T$$

- Let $L = \varepsilon + \gamma W$ be function of Lagrange multiplier method

$$\frac{\partial L}{\partial W[p,i]} = -2x_p(x_i)^T + 2\sum_{x_j \in \mathbb{N}[x_p]} x_i(x_j)^T W[p,i] + \gamma \qquad \frac{\partial L}{\partial \gamma} = \sum_{x_i \in \mathbb{N}[x_p]} W[p,i]\text{-}1$$

- If $C_p[i,j] = x_i(x_j)^T$ and $p[i] = x_p(x_i)^T$, we have $C_pW = p$ from Lagrange multiplier method, thus $W = p(C_p)^{-1}$

- $(C_p)^{-1}$ is independent from x_p, so incrementally updated

Incremental weight computation

- Here we assume that the first nearest neighbor node is different from node x_p and x_q
  - We also assume that norms of x_p and x_q are 1

different

$$C_q = \begin{pmatrix} 1 & x_1(x_2)^T & x_1(x_3)^T \\ x_1(x_2)^T & 1 & x_2(x_3)^T \\ x_1(x_3)^T & x_2(x_3)^T & 1 \end{pmatrix} \Rightarrow C_p = \begin{pmatrix} 1 & \boxed{x_1(x_2)^T \quad x_1(x_3)^T} \\ \boxed{\begin{matrix} x_1(x_2)^T \\ x_1(x_3)^T \end{matrix}} & \begin{matrix} 1 & x_2(x_3)^T \\ x_2(x_3)^T & 1 \end{matrix} \end{pmatrix}$$

- If $\Delta C = C_p - C_\mathrm{q}$, $\Delta C = V^T DV$ where V and D are rank-2
  - Since ΔC has particular form, it need $O(K)$ to compute $V^T DV$
- We can compute $(C_p)^{-1}$ at $O(K^2)$ by the Woodbury formula
  - We apply each different nearest neighbor node to compute $(C_p)^{-1}$

$$(C_p)^{-1} = (C_q)^{-1} - (C_q)^{-1} V((F)^{-1} + (V)^T (C_q)^{-1} V)^{-1} (V)^T (C_q)^{-1}$$

Improve lower bound

- Prune distance computation in computing k-NN graph by approximating Euclidean distance $\mathrm{E}[x_p, x_q]$
  - SVD is a popular approach for approximation

- Improve lower bound by SVD $\mathrm{E}[\widetilde{x_p}, \widetilde{x_q}]$ as $\underline{E}[x_p, x_q]$ in the following form:

$$\underline{E}[x_p, x_q] = \sqrt{\underbrace{(\mathrm{E}[\widetilde{x_p}, \widetilde{x_q}])^2}_{\text{Distance by SVD}} + \underbrace{(u_r[x_p] - u_r[x_q])^2}_{\text{Lower bound by dimensions not used in SVD}}}$$

Distance by SVD

Lower bound by dimensions not used in SVD

where $u_r[x_p] = \sqrt{\underbrace{(\mathrm{E}[x_p, x_r])^2 - (\mathrm{E}[\widetilde{x_p}, \widetilde{x_q}])^2}_{\text{Norm of dimensions not used in SVD}}}$

Norm of dimensions not used in SVD

We use triangular inequality in this approach

LU decomposition based Eigen decomposition

- Power method is the most popular approach in computing Eigen vector
  - But, it computes the largest not smallest Eigen values
  - Embedding is the smallest Eigen vector of $\mathrm{K} = (I - W)^T(I - W)$
- Inverse power method computes the smallest Eigen value
  - It apply power method for the inverse matrix
  - Its computation cost is $O(N^3)$
  - Impractical for large-size of dataset
- We avoid the inverse matrix by LU decomposition
  - We have sparse matrices after LU decomposition
  - We can apply this approach of large graphs

LU decomposition based Eigen decomposition

- Compute LU decomposition for $I - W$ $(\mathrm{LU} = I - W)$
  - Thus we have $\mathrm{K} = (I - W)^T(I - W) = U^T L^T LU$

- The smallest Eigen value λ_N and its Eigen vector Z_N can be computed as follows similar to power method:

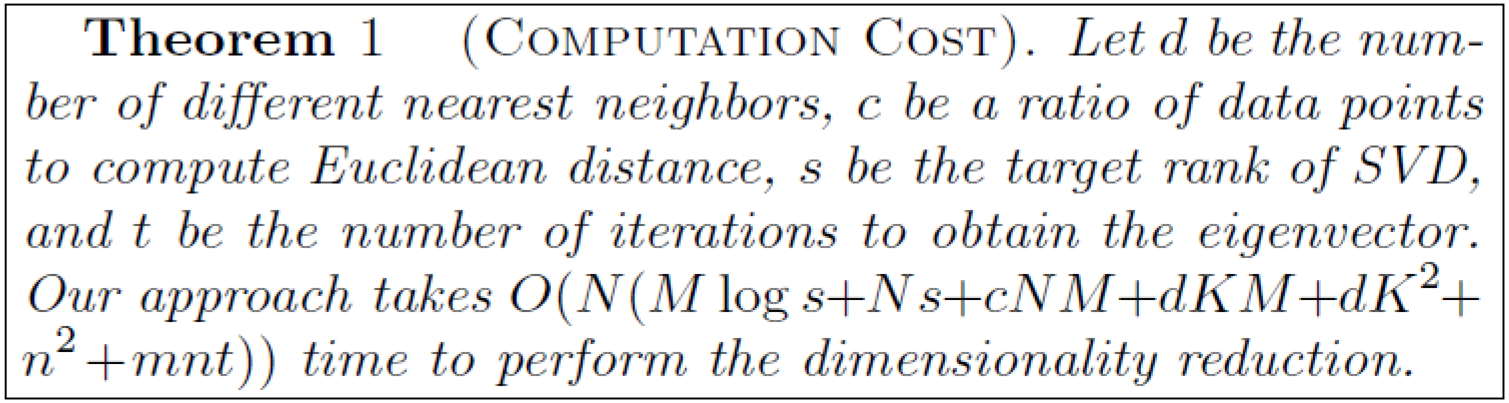
$$\lambda_\mathrm{N} = \{(a_\tau)^T a_\tau\}/\{(a_\tau)^T a_{\tau-1}\} \qquad z_\mathrm{N} = \|a_\tau\|/a_\tau$$

where $a_{\tau-1} = U^T b, b = U^T b', b' = Lb'', b'' = Ua_\tau$

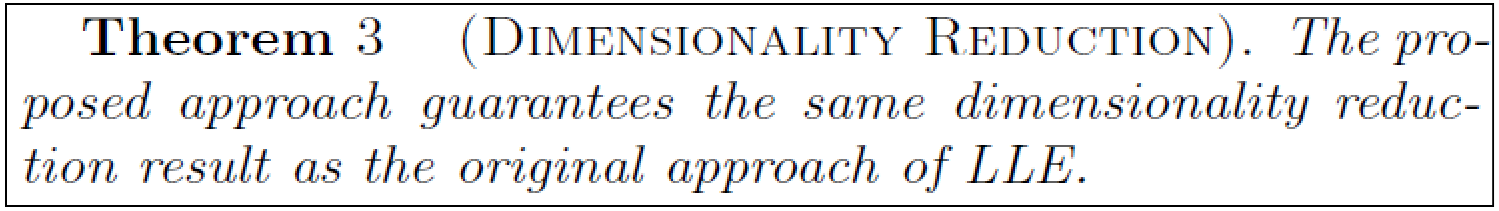
- Since vector a_τ is updated as $a_{\tau-1} = U^T L^T LU a_\tau$, we can compute the smallest Eigen value
  - Note we have $a_\tau = K^{-1} a_{\tau-1}$ since $\mathrm{K} = U^T L^T LU$

Theoretical analysis

- Ripple can efficiently obtain the same embedding results as the original approaches



Theorem 1 (Computation Cost). *Let d be the number of different nearest neighbors, c be a ratio of data points to compute Euclidean distance, s be the target rank of SVD, and t be the number of iterations to obtain the eigenvector. Our approach takes $O(N(M \log s + Ns + cNM + dKM + dK^2 + n^2 + mnt))$ time to perform the dimensionality reduction.*



Theorem 3 (Dimensionality Reduction). *The proposed approach guarantees the same dimensionality reduction result as the original approach of LLE.*

Experiment: preliminaries

- We used the following five datasets
  - USPS; 7291 items and 256 features
  - SensIT; 78,823 items and 100 features
  - ALOI; 108,000 items and 128 features
  - MSD; 515,345 items and 90 features
  - INRIA; 1,000,000 items and 128 features
- Comparison methods
  - CLLE: k-means based approach[*3]
  - LLL: Nystrom method based approach[*4]
  - VN: Nystrom method based approach[*5]

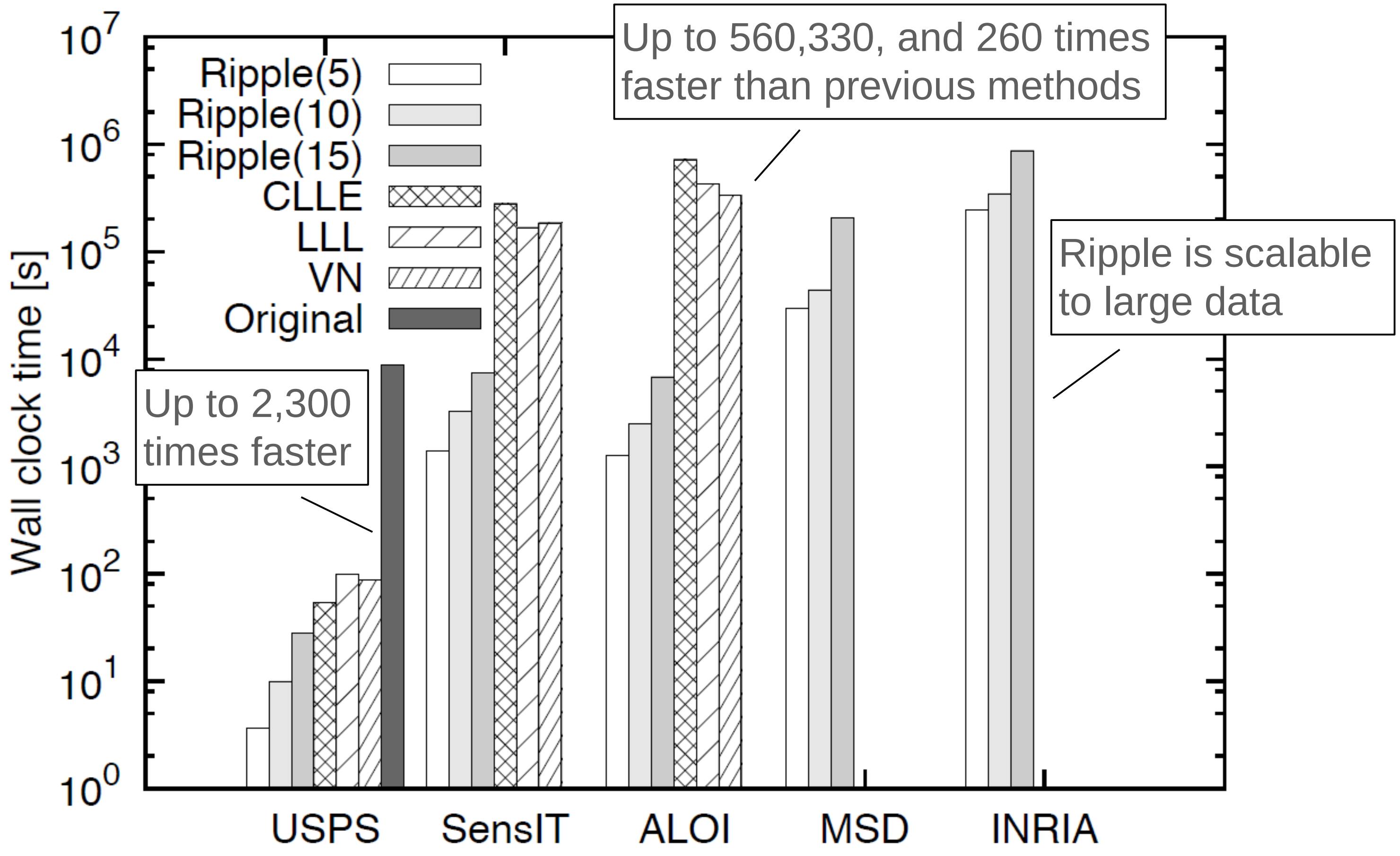
*3 Hui et al., “Clustering-based Locally Linear Embedding”, ICPR, 2008

*4 Vladymyrov et al., “Locally Linear Landmarks for Large-scale Manifold Learning”, ECML/PKDD, 2013

*5 Vladymyrov et al., “The Variational Nystr¨om Method for Large-Scale Spectral Problems”, ICML, 2016

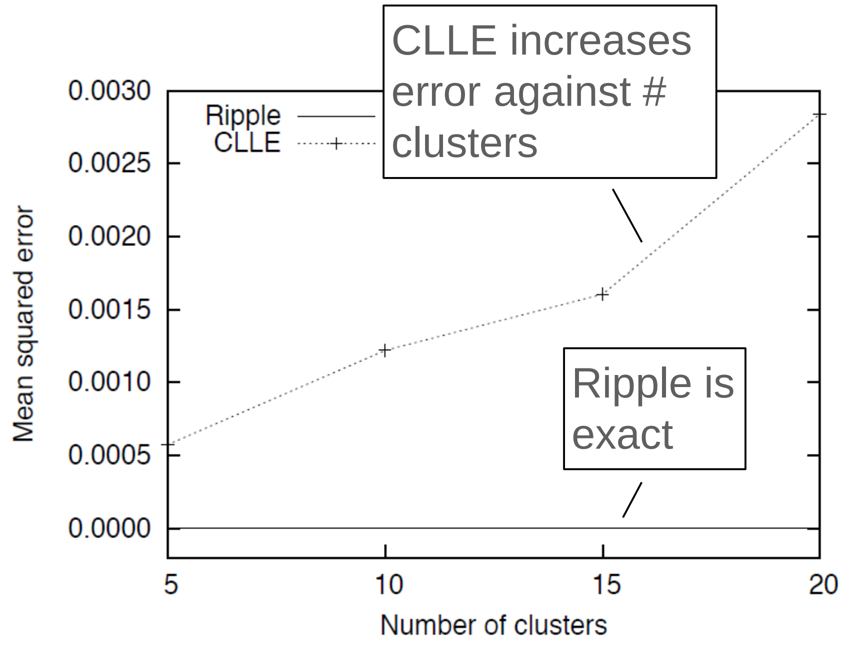
Experiment: efficiency

- Wall clock time
  - Ripple is much faster than existing methods

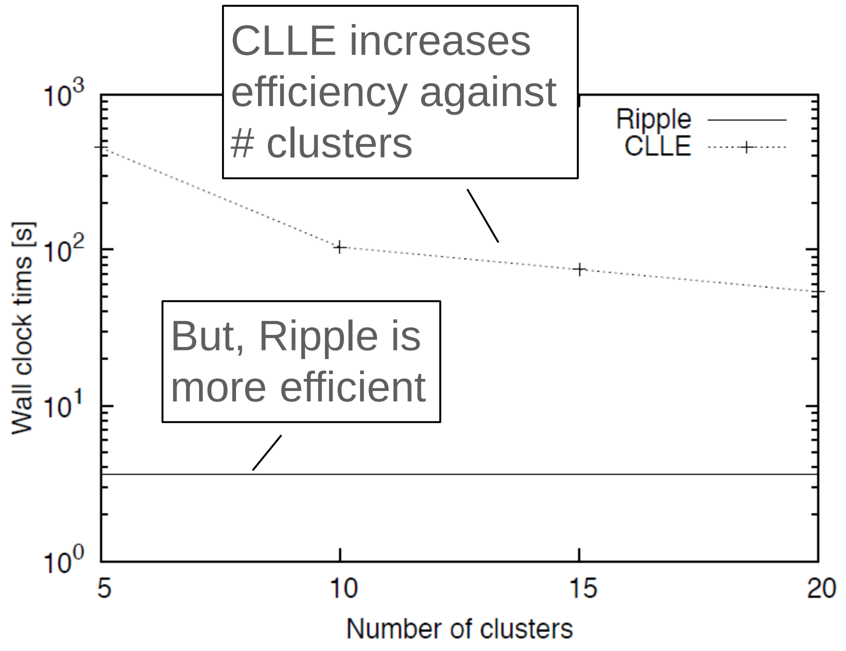


Experiment: exactness (CLLE)

- Ripple yields the same result as the original approach
- CLLE has trade-off between efficiency and accuracy against # clusters of k-means method



Accuracy



Efficiency

Conclusions

- This study proposed an efficient approach for Locally linear embedding (LLE)
- Our approach, Ripple, (1) incrementally compute edge weights, (2) improve the lower bounds in obtaining k-NN graph, and (3) exploits LU decomposition in computing Eigen vectors
- Experimental results show that our approach is faster than the previous approach

Thank you for your attention