EFFICIENT OPTIMIZATION FOR DATA VISUALIZATION AS AN INFORMATION RETRIEVAL TASK

Jaakko Peltonen and Konstantinos Georgatzis

Aalto University, Department of Information and Computer Science,
P.O. Box 15400, FI-00076 Aalto, Finland
{jaakko.peltonen, konstantinos.georgatzis}@aalto.fi

ABSTRACT

Visualization of multivariate data sets is often done by mapping data onto a low-dimensional display with nonlinear dimensionality reduction (NLDR) methods. Many NLDR methods are designed for tasks like manifold learning rather than low-dimensional visualization, and can perform poorly in visualization. We have introduced a formalism where NLDR for visualization is treated as an information retrieval task, and a novel NLDR method called the Neighbor Retrieval Visualizer (NeRV) which outperforms previous methods. The remaining concern is that NeRV has quadratic computational complexity with respect to the number of data. We introduce an efficient learning algorithm for NeRV where relationships between data are approximated through mixture modeling, yielding efficient computation with near-linear computational complexity with respect to the number of data. The method inherits the information retrieval interpretation from the original NeRV, it is much faster to optimize as the number of data grows, and it maintains good visualization performance.

***Index Terms*—** Visualization, dimensionality reduction, neighbor retrieval, efficient computation, mixture modeling

1. INTRODUCTION

Visualization is crucial in analysis of multivariate data sets, especially in the first stages where strong hypotheses about the data are not available. Large data arise for instance in bioinformatics studies where expression levels can be measured for thousands of genes over each patient. To reduce the data onto a two- or three-dimensional scatterplot display, dimensionality reduction is usually applied, from linear projections to mappings found by *nonlinear dimensionality reduction* (NLDR) methods. Visualization is a traditional application of NLDR, however, many NLDR methods are not designed for visualization but tasks like *manifold learning*. Manifold learning methods are often not designed to reduce dimensionality below the dimensionality of the manifold, and can work poorly in low-dimensional visualization [1].

A low-dimensional display cannot represent all properties of the high-dimensional data; it is then crucial to quantify errors that necessarily happen in the visualization, and the goodness of the visualization. There is no generally agreed on goodness measure for visualization. Often mappings are just inspected by eye, or by classification accuracy on the display or preservation of pairwise distances which may not match the tasks an analyst would like to do using the visualization. A good criterion should correspond to real needs of analysts.

We have introduced a novel NLDR approach in low-dimensional visualization, where visualizations are *optimized for a specific visualization task, information retrieval based on the visualization*. Based on the approach, we have introduced a well-performing NLDR method, the Neighbor Retrieval Visualizer (NeRV) [2, 3], which optimizes visualizations for retrieval of neighborhood relationships. NeRV has outperformed state of the art visualization methods [3]. A remaining concern is that its *computational complexity* is quadratic with respect to the number of data samples N. Quadratic complexity often occurs in pairwise distance based NLDR methods: it takes much time as N grows, hindering use of the methods for large data sets. A typical speedup is to visualize a data subset, but this can be unsatisfactory in exploratory analysis as the subset may contain just a small portion of the interesting data relationships. E.g. neighborhood preserving hashing or approximate queries [4] can speed up neighbor search, but using them well in NLDR is nontrivial as coordinates undergo complex changes in optimization.

We introduce *a fast version of the Neighbor Retrieval Visualizer* having *near-linear complexity* with respect to the number of samples. The idea is: use all neighbor relationships in optimization, a few exactly, the rest by *expectation over a mixture model*. For each point, a few of the neighbors are treated with exact coordinates, the rest by mean over a Gaussian mixture model of the coordinates. We recap the NeRV approach, then present our efficient method and experiments.

The authors belong to Helsinki Institute for Information Technology HIIT. This work was financially supported by the Academy of Finland (Finnish Centre of Excellence in Computational Inference Research COIN, grant no 251170; JP also by decision number 252845. The work was also supported in part by the PASCAL2 EU NoE, ICT 216886. This publication reflects the authors' views only.

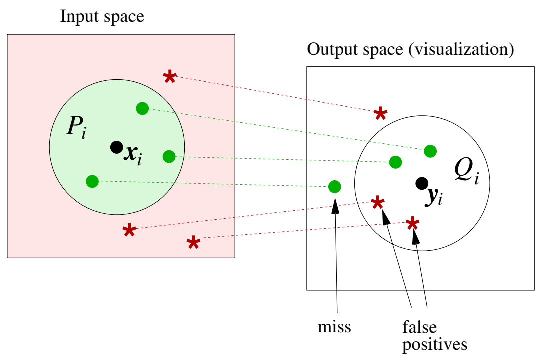

Fig. 1. Illustration of errors in visual information retrieval of neighbors for query point i. Misses are points in the input neighborhood P_i that are not retrieved in the output neighborhood Q_i, false positives are points in Q_i that are not in P_i.

2. VISUALIZATION AS AN INFORMATION RETRIEVAL TASK

Consider mapping a D_{in} dimensional data set $\{\mathbf{x}_i\}_{i=1}^N$ onto a low-dimensional display for visualizing similarity relationships. Let each sample i have an *input neighborhood* P_i of samples close to i. P_i might contain all samples (other than i) within some radius from i, or a fixed number of samples closest to i. The goal is to create output coordinates $\{\mathbf{y}_i\}_{i=1}^N$ for the data, for use in *visual information retrieval*. The output dimensionality D_{out} is typically 2 or 3. In visual information retrieval, given a sample i as a query, a set Q_i called the *output neighborhood* is retrieved containing samples close to $\mathbf{y}_i$ on the display, typically all samples within some radius from $\mathbf{y}_i$ or a fixed number of samples closest to $\mathbf{y}_i$. The number of points k_i in Q_i may differ from the number of points r_i in P_i.

Evaluating visualization performance. Generally a low-dimensional display cannot represent all neighborhood relationships of high-dimensional data exactly; some output neighborhoods Q_i will not correspond to the original input neighborhoods P_i. Two kinds of errors will then happen: neighbor points from P_i that are not in Q_i are *misses*, whereas points that are in Q_i but not in P_i are *false neighbors* (false positives); see Fig. 1. Assume the user has assigned a cost C_F for a false positive and C_M for a miss, and denote the number of false positives for query i by $N_{F,i}$ and the number of misses by $N_{M,i}$. The total cost for query i is $E_i = N_{F,i}C_F + N_{M,i}C_M$. Setting $C_M = C'_M/r_i$, it can be shown the total cost becomes a sum of *precision* and *recall*: $E_i/k_i = C_F(1 - \text{precision}(i)) + (C'_M/k_i)(1 - \text{recall}(i))$, where $\text{precision}(i) = 1 - N_{F,i}/k_i$ and $\text{recall}(i) = 1 - N_{M,i}/r_i$ are usual information retrieval definitions. To evaluate a whole visualization, the cost is averaged over samples i yielding *mean precision* and *mean recall* of the visualization.

Precision-recall tradeoff for continuous neighborhoods. To evaluate grades of relevance in visualizations, we extend the neighborhood relationships to continuous probabilistic neighborhoods. We define the neighborhood on the display as a probability distribution over neighbor points; for a query point $\mathbf{y}_i$, if the user selects a neighbor for inspection, $q_{j|i}$ is the probability that point $\mathbf{y}_j$ is chosen. The $q_{j|i}$ can be defined at simplest as a normalized Gaussian falloff so that

$$q_{j|i} = \frac{\exp(-||\mathbf{y}_i - \mathbf{y}_j||^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-||\mathbf{y}_i - \mathbf{y}_k||^2/2\sigma_i^2)} \tag{1}$$

where σ_i^2 adjusts the size of the neighborhood around i. In the input space, we can analogously extend the neighborhood definition to probabilistic neighborhood distributions $p_{j|i}$ as

$$p_{j|i} = \frac{\exp(-d^2(\mathbf{x}_i, \mathbf{x}_j)/2\sigma_i^2)}{\sum_{k\neq i}\exp(-d^2(\mathbf{x}_i, \mathbf{x}_k)/2\sigma_i^2)} \tag{2}$$

where $d(\cdot,\cdot)$ is a suitable distance measure; we simply use the Euclidean distance metric between original feature vectors $\mathbf{x}_i$. The parameter σ_i^2 is set to make the entropy of the $p_{\cdot|i}$ distribution equal to $\log k$ where k is set by the user as a rough upper limit for the number of relevant neighbors.

It remains to measure how well retrieval from the display corresponds to input space neighborhoods. Nonlinear mappings can yield false neighbors ($q_{j|i} > p_{j|i}$) and misses ($q_{j|i} < p_{j|i}$). A natural measure of the difference between neighborhood distributions is the Kullback-Leibler divergence $D(p_i, q_i) = \sum_{j\neq i} p_{j|i}\log(p_{j|i}/q_{j|i})$ where p_i and q_i are neighborhood distributions around point i in the input space and on the display. It can be shown $D(p_i, q_i)$ is a generalization of recall and $D(q_i, p_i)$ is a generalization of precision [3]; we call the generalizations *smoothed recall* and *smoothed precision*. To evaluate a whole visualization we take the mean over points i, yielding *mean smoothed recall* $\mathbb{E}_i[D(p_i, q_i)]$ and *mean smoothed precision* $\mathbb{E}_i[D(q_i, p_i)]$; they can be used to evaluate performance in visual information retrieval. We argue the retrieval measures correspond better to a human task than e.g. feature reconstruction error in autoencoders where reconstruction functions can be complex.

3. THE NEIGHBOR RETRIEVAL VISUALIZER

Mean smoothed recall and precision are not only useful measures for evaluating visualizations, they are continuous functions of the outputs $\mathbf{y}_i$ and can be used as *optimization criteria* for a visualization method. The method Neighborhood Retrieval Visualizer (NeRV; [2, 3]) uses the measures to optimize visualizations for information retrieval. Once the user has assigned a cost for misses and false neighbors, the total cost of visualization errors corresponds to a tradeoff between precision and recall. NeRV then minimizes the total cost

$$E_{\text{NeRV}} = \lambda\mathbb{E}_i[D(p_i, q_i)] + (1 - \lambda)\mathbb{E}_i[D(q_i, p_i)] \tag{3}$$

where the relative cost of false neighbors to misses is represented by a parameter λ: $\lambda = 0$ optimizes precision only,

$\lambda = 1$ optimizes recall only, and λ between 0 and 1 optimize intermediate tradeoffs between precision and recall. Setting $\lambda = 1$ in NeRV yields the earlier method Stochastic Neighbor Embedding (SNE; [5]), hence SNE can be seen as a method that optimizes mean smoothed recall. More generally NeRV optimizes flexible tradeoffs between precision and recall.

The cost (3) is a function of the output coordinates: they can then be directly treated as variables and optimized. NeRV minimizes (3) with respect to output coordinates $\mathbf{y}_i$ of all points by conjugate gradient descent. The gradient is

$$\frac{\partial E_{\text{NeRV}}}{\partial \mathbf{y}_i} = \Big(\sum_{j\neq i} D_{i,j} q_{j|i}\Big)\left(\sum_{j\neq i} q_{j|i}\frac{\mathbf{y}_i - \mathbf{y}_j}{\sigma_i^2}\right) + \sum_{j\neq i}(w_j - D_{j,i})q_{i|j}\frac{\mathbf{y}_i - \mathbf{y}_j}{\sigma_j^2} - \sum_{j\neq i} D_{i,j} q_{j|i}\frac{\mathbf{y}_i - \mathbf{y}_j}{\sigma_i^2} \quad (4)$$

where $D_{i,j} = (1-\lambda)(\log q_{j|i} - \log p_{j|i} + 1) - \lambda p_{j|i}/q_{j|i}$ and $w_i = \sum_{j\neq i} D_{i,j} q_{j|i}$.

Computing the cost and gradient takes $O(N^2 D_{\text{out}})$ time for N data points and D_{out} output dimensions, similarly to many pairwise distance based embedding methods like SNE. Quadratic complexity with respect to N makes it hard to use information retrieval based visualization for large data; we now present an efficient method based on mixture modeling.

4. EFFICIENT LEARNING FOR NERV BY MIXTURE MODELING BASED APPROXIMATION

We introduce an efficient learning method for NeRV, having linear computational complexity with respect to the number of data points: this speeds up visualization and allows fast visualization of large data. The method is based on iterative optimization (conjugate gradient descent) used successfully for the original NeRV; we get speedup by computing the cost function and gradient approximately in near-linear time. The approximation does not drop any terms from the cost or gradient: we process all terms but some of them through *mean over a mixture model*. In our new method, we compute some expectations over neighbor relationships analytically. We use the Gaussian falloff as in (1) and (2), and use the Euclidean metric in (2). This setting is useful for a variety of data; fast learning with other settings will be treated in future work.

Fast computation of the cost function. Inserting the definitions (1) and (2) with the Euclidean metric into the NeRV cost function (3) yields

$$E_{\text{NeRV}} = \sum_{i,j\neq i}\frac{\lambda e^x_{ij}}{R^x_i}\log\left(\frac{e^x_{ij}/R^x_i}{e^y_{ij}/R^y_i}\right) + \sum_{i,j\neq i}\frac{(1-\lambda)e^y_{ij}}{R^y_i}\log\left(\frac{e^y_{ij}/R^y_i}{e^x_{ij}/R^x_i}\right)$$
$$= \sum_i\left[\frac{\lambda}{\mathrm{R}^x_i}(\mathrm{S}^x_i - \mathrm{T}^{xy}_i) + \frac{1-\lambda}{\mathrm{R}^y_i}(\mathrm{S}^y_i - \mathrm{T}^{yx}_i) + (1-2\lambda)\log\frac{\mathrm{R}^x_i}{\mathrm{R}^y_i}\right] \quad (5)$$

where $e^x_{ij} = \exp(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma_i^2})$, $e^y_{ij} = \exp(-\frac{||\mathbf{y}_i - \mathbf{y}_j||^2}{2\sigma_i^2})$ and $\mathrm{R}^x_i = \sum_{j\neq i} e^x_{ij}$, $\mathrm{S}^x_i = -\sum_{j\neq i} e^x_{ij}\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma_i^2}$, the cross-term is $\mathrm{T}^{xy}_i = -\sum_{j\neq i} e^x_{ij}\frac{||\mathbf{y}_i - \mathbf{y}_j||^2}{2\sigma_i^2}$, and R^y_i, S^y_i and T^{yx}_i are the same with roles of $\mathbf{x}$ and $\mathbf{y}$ reversed. To simplify the setup we use equal neighborhood scales $\sigma_i^2 = \sigma^2$.

Each point i has $N-1$ neighbors j, and evaluating all the sums exactly over all the points i would yield $O(N^2)$ complexity. Instead, for each point i we will compute each sum R^x_i, S^x_i, T^{xy}_i, R^y_i, S^y_i, and T^{yx}_i in two parts as follows.

In each sum over neighbors such as R^x_i, we evaluate the terms exactly for a subset E_i of neighbors denoted the *exact neighbors*. The E_i will be chosen to contain a small number m_{exact} of neighbors close to i as detailed later.

We call the remaining $N-1-m_{\text{exact}}$ neighbors of point i the *non-exact neighbors*: we treat the input coordinates $\mathbf{x}_j$ and current output coordinates $\mathbf{y}_j$ of these neighbors as latent variables, and take the *expectation* of each sum R^x_i, S^x_i, T^{xy}_i, R^y_i, S^y_i, and T^{yx}_i over the latent coordinates. Assume the $\mathbf{x}_j$ follow a mixture of Gaussians distribution with C components $N(\mu^x_c, \mathbf{\Sigma}^x_c)$ and the $\mathbf{y}_j$ follow a corresponding Gaussian mixture with components $N(\mu^y_c, \mathbf{\Sigma}^y_c)$. For each non-exact neighbor we know the posterior weights $w^j_c = p(c|\mathbf{x}_j)$ over the mixture components. The expectation over a sum term $f(\mathbf{x}_j)$ then becomes $\sum_c w^j_c \int_{\mathbf{x}} N(\mathbf{x};\mu^x_c,\mathbf{\Sigma}^x_c)f(\mathbf{x})d\mathbf{x}$ where $N(\mathbf{x};\mu^x_c,\mathbf{\Sigma}^x_c)$ is the density of mixture component c evaluated at $\mathbf{x}$. The sums R^x_i, S^x_i, T^{xy}_i, R^y_i, S^y_i, and T^{yx}_i in the cost function (5) involve exponents, or exponents multiplied by squared distances, and the expectation over such terms can be computed analytically, using Gaussian identities. For all non-exact neighbors, the expectations do not depend on the exact value of the latent coordinate $\mathbf{x}_j$ and instead depend on the mixture distribution; the same happens for terms involving $\mathbf{y}_j$. The contribution of all non-exact neighbors to the cost function terms then reduces to a sum over the mixture components, yielding efficient linear-time computational complexity: for a fixed σ^2, input-space terms are first computed in linear time $O(Nm_{\text{exact}}D_{\text{in}} + NCD^2_{\text{in}} + CD^3_{\text{in}})$ which only needs to be done once for a fixed σ^2, and then each evaluation of the cost function takes linear time $O(N(m_{\text{exact}} + C)D_{\text{out}})$. The resulting expectation equations follow from Gaussian identities; we will provide the full equations in a technical report.

Fast gradient computation. The gradient can be handled much like the cost function, by inserting (1) and (2) into the gradient (4) and arranging it as a function of several sums. The procedure goes as before; for brevity we just note the result gradient equation uses the same sums R^x_i, T^{xy}_i, R^y_i, S^y_i, and T^{yx}_i as (5), and also sums $\mathrm{G1}^y_i = \sum_{j\neq i} e^y_{ij}(\mathbf{y}_i - \mathbf{y}_j)$, $\mathrm{G2}^{yx}_i = \sum_{j\neq i}\frac{w_j}{\mathrm{R}^y_j}e^y_{ji}(\mathbf{y}_i - \mathbf{y}_j)$ where $w_i = \frac{1-\lambda}{2\sigma_i^2}(\mathrm{S}^y_i - \log\mathrm{R}^y_i - \mathrm{T}^{yx}_i + \log\mathrm{R}^x_i) + \frac{1-2\lambda}{2\sigma_i^2}$, $\mathrm{G3}^y_i = \sum_{j\neq i} e^y_{ij}(\frac{-||\mathbf{y}_i - \mathbf{y}_j||^2}{2\sigma_i^2} - \log\mathrm{R}^y_i)\mathbf{y}_j$, $\mathrm{G4}^y_i = \sum_{j\neq i} e^y_{ij}(\frac{-||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma_i^2} - \log\mathrm{R}^x_i)\mathbf{y}_j$, $\mathrm{G5}^y_i = \sum_{j\neq i} e^y_{ij}\frac{\mathbf{y}_j}{\mathrm{R}^y_i}$, $\mathrm{G6}^y_i = \sum_{j\neq i} e^x_{ij}\frac{\mathbf{y}_j}{\mathrm{R}^x_i}$, $\mathrm{H}^{yx}_i = \sum_{j\neq i} e^x_{ji}\frac{\mathbf{y}_i - \mathbf{y}_j}{2\sigma_j^2\mathrm{R}^x_j}$,

$\mathrm{G7}_i^y = \sum_{j\neq i} e_{ji}^y (\frac{-||\mathbf{y}_i-\mathbf{y}_j||^2-\log \mathrm{R}_j^y}{\mathrm{R}_j^y}) \frac{\mathbf{y}_i-\mathbf{y}_j}{2\sigma_j^2}$, $\mathrm{G8}_i^{yx} = \sum_{j\neq i} e_{ji}^y (\frac{-||\mathbf{x}_i-\mathbf{x}_j||^2-\log \mathrm{R}_j^x}{\mathrm{R}_j^y}) \frac{\mathbf{y}_i-\mathbf{y}_j}{2\sigma_j^2}$, and $\mathrm{G9}_i^y = \sum_{j\neq i} e_{ji}^y \frac{\mathbf{y}_i-\mathbf{y}_j}{2\sigma_j^2 \mathrm{R}_j^y}$. For the non-exact neighbors we again take expectation over the Gaussian mixture and the resulting integrals can again be computed analytically. To avoid double integrals over the mixture, for the non-exact neighbors in sums $\mathrm{G2}_i^{yx}$, $\mathrm{G7}_i^y$, $\mathrm{G8}_i^{yx}$, $\mathrm{G9}_i^y$ and H_i^{yx} we approximate w_j/R_j^y, R_j^y and R_j^x within each mixture component by its mean value over data from the component, which can be computed in linear time. As a result, for a fixed σ^2, after input-space terms have been computed in linear time $O(Nm_{\text{exact}}D_{\text{in}} + NCD_{\text{in}}^2 + CD_{\text{in}}^3)$, evaluating the gradient for a low output dimensionality D_{out} takes linear time $O(Nm_{\text{exact}}D_{\text{out}} + NCD_{\text{out}}^2 + CD_{\text{out}}^3)$.

4.1. Full Algorithm

The full algorithm is presented as pseudocode in Figure 2. We now tell how the remaining operations are done in linear time.

Clustering. The clustering of the original data into C clusters and computation of the cluster means and covariances can be done in $O(NCD_{\text{in}} + (N + C)D_{\text{in}}^2)$ time by several methods such as k-means which we use here (k-means has linear complexity per iteration; we use a fixed 150 iterations). After k-means we run one E-step of expectation-maximization (EM) based Gaussian mixture modeling, to yield probabilistic membership weights of the data points in the clusters; we run only one E-step instead of several EM iterations since each E-step involves inversion of the mixture component covariances which takes $O(CD_{\text{in}}^3 + NCD_{\text{in}}^2)$ time. The cluster statistics and data point memberships are then used throughout the rest of the algorithm.

Choosing the exact neighbors. Choosing the m_{exact} exact neighbors randomly for each point i is done by a permutation approach: briefly, maintain a permutation of data and swap the point i to the front; choose the lth neighbor from the $N - l - 1$ positions at the tail of the permutation and swap it to position $l + 1$. To ensure exact neighbors are fairly close to i, we take exact neighbors first from the same cluster as i and if the cluster does not contain enough points we take the remaining neighbors from all data; this can be done by a modification where permutations are maintained also within each cluster. The total time complexity is linear $O(NC + Nm_{exact})$.

Initialization. The optimization can be started from any initial outputs $\{\mathbf{y}_i\}_{i=1}^N$. With good initialization, the mixture component memberships found for the input space are reasonable in the output space too, hence the approximation of output-space terms of the cost and gradient will be good. We initialize the outputs either as a PCA projection in $O(ND_{\text{in}}^2 + D_{\text{in}}^3)$ time or by mapping the input-space cluster centroids onto the display with NeRV in $O(C^2(D_{\text{in}} + D_{\text{out}}))$ time and then interpolating data between centroids according to their mixture component memberships in $O(NCD_{\text{out}})$ time.

Initialization: 1. Cluster data by k-means, then run one E-step of EM Gaussian mixture modeling to compute mixture component memberships of points. **2.** Initialize output coordinates: use PCA or map input-space mixture component centroids with NeRV and interpolate data between centroids by posterior mixture weights. **3.** Assign the final neighborhood size σ^2 by line search minimizing difference between the entropy $E = \sum_i (\log R_i^x - S_i^x)/NR_i^x$ of neighborhood distributions and its desired value ($\log k$ for k effective neighbors). **4.** Assign the initial σ^2 as maximum squared input-space distance from data to exact neighbors or clusters. **Iterative optimization:** Run several iterations with the following steps each: **5.** Decrease the neighborhood width towards the final value. **6.** Run conjugate gradient descent to minimize the cost function. Each time the cost function or gradient is computed, first update the low-dimensional cluster statistics, then compute the approximate cost function or gradient based on the clustering.

Fig. 2. Pseudocode of the fast optimization algorithm.

Updating the mixture model in the output space. The mixture model in the input space and the mixture component memberships of points remain fixed throughout the algorithm. To maintain the mixture model on the display, before each evaluation of the cost function or gradient we update component means and covariances in the output space by one M-step of expectation-maximization based on the current output coordinates $\mathbf{y}_i$, which takes $O(NCD_{\text{out}}^2)$ time. We use a further speedup by tresholding very small component memberships of points to zero, and only process nonzero memberships.

4.2. Discussion

Our method is linear-time when the numbers of exact neighbors m_{exact} and clusters C are negligible compared to the number of data N. For larger m_{exact} and C the method becomes near-linear-time. Large m_{exact} and C are not needed, in experiments we get good performance even with $m_{\text{exact}} = 0$.

Since NeRV contains Stochastic Neighbor Embedding (SNE) as the case $\lambda = 1$ where only recall is optimized, our method also yields an efficient algorithm for SNE, but more generally the method optimizes any tradeoff between precision and recall. For SNE and its variants [6], there have been previous approaches for alternative and more efficient optimization methods, for example [7] proposes multiplicative updates, [8] finds the direction and step size based on a quadratic model of the SNE cost, and [9] proposes an algorithm related to SNE but with a simpler form. However, for the general case of optimizing both precision and recall we are not aware of previous efficient computation approaches.

A method for handling pairwise distances based on Krylov iteration has been proposed [10] yielding $O(N \log N)$ complexity. Such approaches are compatible with our approximation approach: if a large proportion of exact neighbors is desired, they can be handled through methods such as [10] while non-exact neighbors are handled through mixture modeling as proposed here; we did not use such combination since our method already yielded very efficient computation.

5. EXPERIMENTS

When all neighborhood relationships are treated exactly, our method essentially yields the previous Neighbor Retrieval Visualizer (NeRV) method which has already been shown to yield very good performance [3]. We now show that the new efficient method maintains the good performance with much faster computation. We use settings and comparison methods from [3], see [3] for details of comparison methods.

We compute 2-dimensional visualizations; for our method we run the fastest setting $m_{\text{exact}} = 0$ and also $m_{\text{exact}} = 0.2N, 0.4N, 0.6N, 0.8N, N$ to show how the method approaches the original NeRV. We use $C = 25$ mixture components except for data sets Phoneme where $C = 20$ and TIMIT where $C = 15$ sufficed. We compare our method to Principal Component Analysis (PCA), Metric Multidimensional Scaling and Isomap (MDS and Isomap; see [11]), Locally Linear Embedding and Hessian-based Locally Linear Embedding (LLE and HLLE; see [12]), Curvilinear Component Analysis and Curvilinear Distance Analysis (CCA and CDA; see [13]), Maximum Variance Unfolding and Landmark Maximum Variance Unfolding (MVU and LMVU; see [14]), and Local MDS (LMDS;[15]). See [3] for overviews of the methods.

Comparison of information retrieval performance. We compare our new method with the original NeRV and selected other methods in information retrieval on three data sets: S-curve (1000 data, $D_{\text{in}} = 3$), noisy S-curve (1000 data, $D_{\text{in}} = 3$), and mouse gene expression ([3]; 1600 data, $D_{\text{in}} = 45$). We use the experiment setting from [3], setting the ground-truth input neighborhood to 20 effective neighbors and evaluating methods by the information retrieval measures mean smoothed precision and mean smoothed recall. Results for other methods are from [3]. Fig. 3 shows the results: our method maintains state of the art performance even at the fastest setting. The speedup is strong: for S-curve the fastest setting $m_{\text{exact}} = 0$ is about 36 times faster than $m_{\text{exact}} = N$ corresponding to original NeRV (12.3 seconds versus 445 seconds); for noisy S-curve we get about 26 times speedup (12.2 seconds vs. 316.9), and for mouse gene expression we get 50 times speedup (26 seconds vs. 1313.5).

Comparison based on unsupervised classification accuracy. We then compare our method to the others on two labeled data sets of phoneme samples [3], Phoneme (1500 data, $D_{\text{in}} = 20$, 13 classes; distributed with LVQ-PAK software) and TIMIT (1500 data, $D_{\text{in}} = 12$, 41 classes; from the DARPA TIMIT speech database), based on a common indirect measure, unsupervised leave-one-out classification accuracy by k-nn classification ($k = 5$) based on the projec-

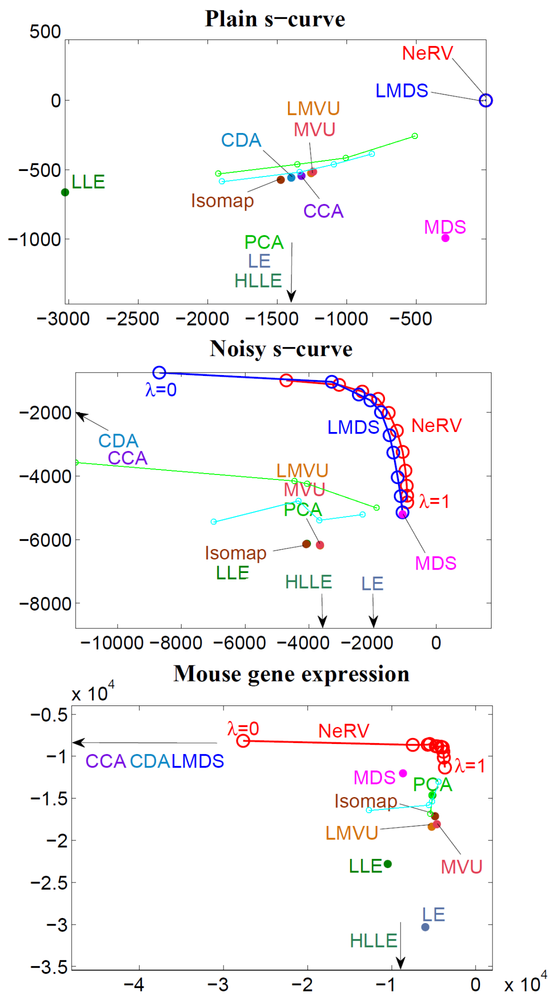

Fig. 3. Retrieval performance. Horizontal axes: $-1\cdot$ mean smoothed recall, vertical axes: $-1\cdot$ mean smoothed precision (multiplication by -1 is done so that best results appear at top right). Our method: curves over λ, green for $m_{\text{exact}} = 0$, cyan for $m_{\text{exact}} = 0.4N$.

tion. Results are shown in Table 1. The new method again maintains very good classification accuracy with substantial speedup ($m_{\text{exact}} = 0$ yields 18.3 times speedup over $m_{\text{exact}} = N$ on Phoneme and 24.1 times on TIMIT).

Efficient learning on a large data set. Lastly, we demonstrate the speed advantage of the new method compared to the original NeRV on the large Covertype data set (10 features, 581012 points) from UCI machine learning repository. We set $\lambda = 0.6$, $m_{\text{exact}} = 0$, and $C = 25$. We try subsets of 1000, 2000, 4000, and 8000 points and report running time, $(-1)\cdot$mean smoothed precision and recall. Results are in Table 2; the new method yields substantial speedup and maintains good results comparable to original NeRV.

The experiments show the new method is a viable efficient alternative for NeRV; it remains comparable with state of the

Method	Phoneme	TIMIT
MDS	0.189	0.705
CDA	0.118	0.643
CCA	0.098	0.633
NeRV	0.079	0.626
LocalMDS	0.118	0.637
Our ($m_{\text{exact}} = 0$)	0.178	0.676
Our ($m_{\text{exact}} = 0.2N$)	0.143	0.663
Our ($m_{\text{exact}} = 0.4N$)	0.126	0.667
Our ($m_{\text{exact}} = 0.6N$)	0.086	0.665
Our ($m_{\text{exact}} = 0.8N$)	0.156	0.666
Our ($m_{\text{exact}} = 1.0N$)	0.083	0.637

Table 1. Error rates of k-nearest neighbor classification accuracy based on the visualization.

Method	DataSize	Time	$-1\cdot$Recall	$-1\cdot$Precision
NeRV	1000	375.6	-3.911	-85.212
NeRV	2000	1498	-4.605	-120.295
NeRV	4000	5986	-6.008	-162.443
NeRV	8000	23358	-6.510	-205.822
Our	1000	35.2	-5.198	-84.832
Our	2000	31.9	-6.365	-119.180
Our	4000	135.7	-6.085	-162.230
Our	8000	170.1	-233.125	-170.4125

Table 2. Running times (lower values are better) and performances (higher values are better) for *covertype* data subsets.

art competitors, and achieves large speedups.

6. CONCLUSIONS

We introduced an efficient nonlinear dimensionality reduction method that optimizes visualizations for information retrieval, in linear or near-linear time with respect to the number of data. The method is a faster version of the very well-performing Neighbor Retrieval Analyzer (NeRV) where the original NeRV has quadratic time complexity. The fast learning is achieved by approximating part of the data relationships through mixture modeling. The method inherits the rigorous information retrieval interpretation from the original NeRV. In experiments, the method performed very well, achieving much faster computation than the original NeRV while maintaining good visualization performance. Code for the method is available at http://research.ics.aalto.fi/mi/software/LinearTimeNeRV.

7. REFERENCES

[1] J. Venna and S. Kaski, "Comparison of visualization methods for an atlas of gene expression data sets," *Information Visualization*, vol. 6, pp. 139–54, 2007.

[2] J. Venna and S. Kaski, "Nonlinear dimensionality reduction as information retrieval," in *Proc. AISTATS*07 (JMLR W&CP Vol. 2)*, 2007, pp. 572–579.

[3] J. Venna, J. Peltonen, K. Nybo, H. Aidos, and S. Kaski, "Information retrieval perspective to nonlinear dimensionality reduction for data visualization, *JMLR*," vol. 11, pp. 451–490, 2010.

[4] D. Dolev, Y. Harari, N. Linial, N. Nisan, and M. Parnas, "Neighborhood preserving hashing and approximate queries," in *Proc. SODA'94*, pp. 251–259. Society for Industrial and Applied Mathematics, 1994.

[5] G. Hinton and S. T. Roweis, "Stochastic neighbor embedding," in *Proc. NIPS 2002*, pp. 833–840. MIT Press, 2002.

[6] L. van der Maaten and G. Hinton, "Visualizing data using t-SNE," *JMLR*, vol. 9, pp. 2579–2605, 2008.

[7] Z. Yang, C. Wang, and E. Oja, "Multiplicative updates for t-SNE," in *Proc. MLSP 2010*, pp. 19–23. IEEE, 2010.

[8] K. Nam, H. Je, and S. Choi, "Fast stochastic neighbor embedding: a trust-region algorithm," in *Proc. IJCNN 2004*, vol. 1, pp. 123–128. IEEE, 2004.

[9] M. Á. Carreira-Perpiñán, "The elastic embedding algorithm for dimensionality reduction," in *Proc. ICML 2010*, pp. 167–174. Omnipress, 2010.

[10] N. De Freitas, Y. Wang, M. Mahdaviani, and Dustin Lang, "Fast Krylov methods for N-body learning," in *Proc. NIPS 2005*. 2006, pp. 251–258, MIT Press.

[11] J. B. Tenenbaum, V. de Silva, and J. C. Langford, "A global geometric framework for nonlinear dimensionality reduction," *Science*, vol. 290, pp. 2319–2323, 2000.

[12] D. L. Donoho and C. Grimes, "Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data," *PNAS*, vol. 100, pp. 5591–5596, 2003.

[13] J. A. Lee, A. Lendasse, and M. Verleysen, "Nonlinear projection with curvilinear distances: Isomap versus curvilinear distance analysis," *Neurocomputing*, vol. 57, pp. 49–76, 2004.

[14] K. Weinberger, B. Packer, and L. Saul, "Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization," in *Proc. AISTATS 2005*, pp. 381–388. Society for AI and Statistics, 2005.

[15] J. Venna and S. Kaski, "Local multidimensional scaling," *Neural Networks*, vol. 19, pp. 889–99, 2006.