Date of acceptance Grade

Instructor

Statistic Software for Neighbor Embedding

Zhao Jie

Helsinki May 29, 2017

UNIVERSITY OF HELSINKI
Department of Maths and Statistics

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Faculty of Science	Department of Maths and Statistics

Tekijä — Författare — Author
Zhao Jie

Työn nimi — Arbetets titel — Title

Statistic Software for Neighbor Embedding

Oppiaine — Läroämne — Subject
Statistics

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
	May 29, 2017	60

Tiivistelmä — Referat — Abstract

Dimension reduction presents expanding importance and prevalence since it lessens the challenge to data visualization and exploratory analysis that numerous science areas rely on. Recently, nonlinear dimension reduction (NLDR) methods have achieved superior performance in coping with complicated data manifolds embedded in high dimensional space. However, conventional statistic software for NLDR visualization purpose (e.g Multidimensional Scaling) often gives undesired desirable layouts. In this thesis work, to improve the performance of NLDR for data visualization, we study the recently proposed and efficient neighbor embedding (NE) framework and develop its software package in statistic software R. The neighbor embedding framework consists of a wide family of NLDR including stochastic neighbor embedding (SNE), symmetric SNE etc. Yet the original SNE optimization algorithm has several drawbacks. For example, it cannot be extended to other NE objective functions and requires quadratic computation cost. To address these drawbacks, we unify many different NE objective functions through several software layers and adopt a tree-based approach for computation acceleration. The core algorithm is implemented in C++ with an lightweight R wrapper. It thus provides an efficient and convenient package for researchers and engineers who work on statistics. We demonstrate the developed software by visualizing the two-dimensional layouts for several typical datasets in machine learning research including MNIST, COIL-20 and Phonemes etc. The results show that NE methods significantly outperform the traditional MDS visualization tool, indicating NE as a promising and useful dimension reduction tool for data visualization in statistics.

Avainsanat — Nyckelord — Keywords
Manifold Learning, Neighbor Embedding, R, Rcpp

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — övriga uppgifter — Additional information

Contents

1 Introduction

With the rapid development of data collection techniques, the amount, source and complexity of data receive explosive growth. One of the most thorny issues based on such "big data" is how to extract and to analyze the essence of data escaping from the data redundancy which commonly appears as too-high observation dimensionality. Dimension reduction techniques have thus been drawing increasing interests since they are capable to significantly reduce the difficulty for following research processes in many science domains such as data visualization, exploratory analysis and multi-perceptron training.

Relying on a simple assumption that, data points in high dimensional space exactly or approximately lie on a lower dimensional manifold embedded in the high dimensional space, a series of dimension reduction methods have been proposed. Principal Component Analysis (PCA) [Hot33] considers a linear or approximate linear global manifold and project the high dimensional data points to the axes that will maximize the variance. For its simplicity and efficiency, PCA is regarded as the most popular dimension reduction method in decades. However, restricted by its linear nature, PCA does not perform well in coping with the complicated high dimensional manifold [vdM09]. To overcome this drawback, novel algorithms were proposed from a non-linear view, not the same as PCA with linear matrix transformation and projection.

Multidimensional Scaling (MDS) [Tor52], which attempts to preserve the distances in high dimensional space, allocates the low dimensional coordinates to minimize the difference between the pairwise distances in both high and low dimensional space. Sammon mapping [Sam69] adds higher weights on closer pairwise data points based on the MDS cost function. Isomap [TDSL00] makes further improvement to retain the manifold by replacing the direct Euclidean distance in MDS with the geodesic distance after importing the k nearest neighbor graph to represent the manifold. Local Linear Embedding (LLE) [RS00] develops a coordinate-free route to seek the best matching of high and low configurations, based on the weighted "neighborhood". Other methods like Maximum Variance Unfolding (MVU) [WS06], Self-Organizing Map (SOM) [Koh98], Laplacian Eigenmap (LE) [BN01] and so on all illustrate insightful comprehension on dimension reduction.

Meanwhile, conventional non-linear dimension reduction methods have one or more drawbacks. For instance, MDS and Sammon mapping are expensive to optimize and

easily getting stuck at poor local optima; LLE encounters the collapse problem and it's cumbersome to deal with the outliers [LV07].

Later Hinton and Roweis proposed a non-linear dimension reduction method, Stochastic Neighbor Embedding(SNE) [HR02], to produce a better dimension reduction performance. It soon receives great popularity since it outperforms most of the dimension reduction approaches with plenty of following variants including Symmetric SNE, Student t-distributed SNE [vdMH08] and weight symmetric SNE [YPK14] etc.

We study the NE framework and focus on weighted t-SNE which has not been further discussed to evaluate its performance in data visualization. Meanwhile, current conventional non-linear dimension reduction and visualization software tools can not provide satisfying results. For us, it will be meaningful to develop an NE package in statistic software, R, to achieve the scalability, efficiency and simplicity of NLDR implementation. Conventional MDS algorithms are taken as the counterpart of NE for the performance comparison, not only because of its popularity, but also for Hinton's interpretation of SNE as a probabilistic version of local MDS. Multiple typical datasets in machine learning are utilized for the result evaluation. The corresponding results clearly demonstrate the advantages of the proposed method and software.

This thesis is outlined as follows. We review the NE framework and critical details in Section 2. The optimization techniques are discussed in Section.3, following a time line of its development. Section.4 explains the algorithm implementation in R covering both R and C++ level. Section.5 introduces the datasets used in experiment and illustrates the results visualization of NE and MDS. Conclusion and discussion for future work are contained in Section 6.

2 Neighbor Embedding

In this section, we make a detailed introduction to neighbor embedding (NE) method starting from presenting the most prevalent non-linear dimension reduction method, MDS in Section 2.1. The framework of the original stochastic neighbor embedding (SNE) method is introduced in Section 2.2. Later parts cover multiple components managing to improve the performance of NE methods. Section 2.3 discusses the techniques to characterize the high dimensional probabilities. Divergences and low dimensional probabilities are introduced in Section 2.3 and 2.4 respectively.

2.1 Multidimensional Scaling

Multidimensional Scaling (MDS) is a popular dimension reduction technique which takes proximity data as input, containing the information of dissimilarities between high dimensional pairwise data points. Conventionally, the proximity or dissimilarity is expressed as distance of which the selection may form multiple variants of the MDS method.

In general, MDS attempts to achieve a goal that, given the high dimensional data points $\mathbf{X} = \{\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_N\} \in \mathbf{R}^k$, obtain the mapped (configuration, embedding) points $\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_N$ in low dimensional space $\mathbf{R}^d$ that would optimally maintain high dimensional dissimilarities.

The original or the classical MDS was proposed by Togerson [Tor52]. Meanwhile, classical MDS has a more complicated cost function named *Strain* based on the inner products of low dimensional mappings $\langle y_i, y_j \rangle$ and requires a two-step transformation process to utilize the information. Kruskal in 1964 developed the leading MDS algorithm, a non-metric MDS [Kru64] which has a more trivial interpretation. The main gap between these two methods is the setting of cost function. The leading MDS has a cost function called *Stress* with the form

$$E = Stress_D(\mathbf{y}_1, \mathbf{y}_2, \cdots, \mathbf{y}_N) = \sum_i \sum_j (\text{dist}(\mathbf{x}_i, \mathbf{x}_j) - \|\mathbf{y}_i - \mathbf{y}_j\|)^2$$

where $\text{dist}(\mathbf{x}_i, \mathbf{x}_j)$ is commonly picked as a monotonic function to describe the high dimensional dissimilarity. The dimension d for the low dimensional space is usually set to $d = 2$ or $d = 3$ in order to operate the data visualization.

MDS variants can be divided into two subsets by either the high dimensional dissimilarity measure or the low dimensional mapping description. For high dimensional pairwise data points $\mathbf{x}_i$ and $\mathbf{x}_j$, the dissimilarity can be metric or non-metric, where for instance non-metirc applies the ranks. Meanwhile, low dimensional mapping may take the Togerson inner products or Kruskal distance as a low dimensional distance measure. Buja et al. discussed further for the topic and gave the more general forms of both metric and non-metric families including the power transformation for metric MDS and isotonic transformation for the non-metric [BSL$^+$08].

Although MDS overall produces popular output, it still has flaws. Two main types are that (1) close data points in high dimensional space are mapped far apart in low dimension, (2) dissimilar, far apart high dimensional data points are put closely

in low dimension. Venna and Kaski defined these phenomenons as trustworthy and continuity [VK06]. For MDS, multiple variants have been proposed to overcome or reduce these mismatch.

Venna and Kaski proposed local MDS [VK06], which is inspired by another MDS variant, curvilinear component analysis (CCA) [DH97], to improve the MDS performance by seeking a balance between trustworthy and continuity. The cost function of local MDS is as follows

$$\begin{aligned} E =& \frac{1}{2}[(1-\lambda)(d(\mathbf{x}_i,\mathbf{x}_j)-d(\mathbf{y}_i,\mathbf{y}_j))^2 F(d(\mathbf{y}_i,\mathbf{y}_j),\sigma_i) \\ &+ \lambda(d(\mathbf{x}_i,\mathbf{x}_j)-d(\mathbf{y}_i,\mathbf{y}_j))^2 F(d(\mathbf{x}_i,\mathbf{x}_j),\sigma_i)] \end{aligned}, \tag{1}$$

where

$$F(d(\mathbf{y}_i,\mathbf{y}_j),\sigma_i) = \begin{cases} 1 & if\ d(\mathbf{y}_i,\mathbf{y}_j) \le \sigma_i \\ 0 & if\ d(\mathbf{y}_i,\mathbf{y}_j) > \sigma_i \end{cases}.$$

Coefficient F represents the the influence zone of one data point in the low dimensional space while σ_i would be adjusted for a certain containment. Parameter λ can be tuned to achieve the optimal tradeoff between trustworthy and continuity. The two parts of the cost function penalize the two kinds of mismatch respectively in order to obtain a global balance of attraction and repulsion among the data points. This reveals a significant similarity with NE framework.

In addition to mismatching, the optimization process also annoys the MDS users. Classical and distance MDS just implement the simple straightforward gradient descent methods to minimize the cost function which usually leads to a poor local optima. A plenty of improvements were developed including random initialization of output coordinates and stochastic gradient descent technique. Klock and Buhmann modified the annealing approach for the optimization task and receive satisfying result [KB00]. Such efforts just indicate the critical role for the optimization techniques in dimension reduction practical work.

2.2 NE framework

Stochastic Neighbor Embedding (SNE), which applies the neighborhood among high dimensional data points and emphasizes the local structure preservation, broadens a new horizon for the dimension reduction techniques. A general comprehension of NE is optimizing a divergence between neighborhoods in the input and output spaces.

In the original SNE algorithm, different from MDS using distance functions to evaluate the pairwise data points similarity, a conditional probability is chosen to assess the probabilistic similarity between x_i and x_j meaning that conditional on point x_i, how probable the point x_j will be chosen as x_i's neighbor representing the neighborhood relation.

The conditional probability for high dimensional data point x_i picking x_j as neighbor has the form of

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|x_i - x_k\|^2/2\sigma_i^2)}$$

and

$$p_{i|i} = 0.$$

SNE considers that the conditional neighborhood probabilities have already provided sufficient information to capture the embedded manifold in high dimensional space. With the construction of high dimensional probabilities, the corresponding low dimensional probabilities have an analogous form that

$$q_{j|i} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{i\neq k}\exp(-\|y_i - y_k\|^2)}$$

and again

$$q_{i|i} = 0,$$

where y_i and y_j are the mapped points of x_i and x_j respectively.

Now suppose that if the relation between x_i and x_j in high dimensional space is well-preserved in low dimensional space by y_i and y_j, then the difference between $p_{j|i}$ and $q_{j|i}$ should be infinitesimal. SNE takes advantage of Kullback-Leibler divergence as the criterion to evaluate the goodness of preserving such relation and manifold. The cost function of SNE is

$$C = \sum_i KL(P_i\|Q_i) = \sum_i\sum_j p_{j|i}\log\frac{p_{j|i}}{q_{j|i}}$$

in which P_i denotes all the conditional probabilities given data point x_i and Q_i denotes the corresponding probabilities given y_i in low dimensional space. In the other word, for every $p_{j|i}$, SNE attempts to find a $q_{j|i}$ as close as possible respectively. Since the cost function developed from Kullback-Leibler divergence is asymmetric, trustworthy and continuity defined in Section 2.1 are not equally preferred. In SNE, trustworthy that close data points in high dimensional space are mapped far apart will cause a high cost while continuity only brings a comparably very small cost

due to the log term in the cost function. This indicates that SNE focuses more on preserving the local structure of the manifold embedded in the high-D space.

SNE implements straightforward gradient descent method to minimize the cost function where the gradient has a form of

$$\frac{\delta C}{\delta y_i} = 2\sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j).$$

This gradient has a trivial physical interpretation as the composition of forces among data points [HR02]. The interaction between pairwise data points can be simulated by a spring giving a force as either attraction or repulsion depending on spring extension which is $y_i - y_j$. Simultaneously, $p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j}$ determines the stiffness of the spring. If $p > q$, points y_i and y_j will be attracted together. On the contrary, $p < q$ indicates that y_i and y_j will be repelled further.

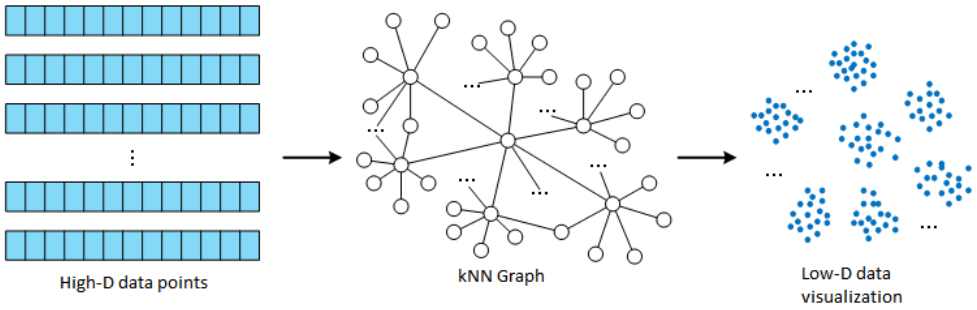
As the earliest version in NE algorithm, SNE outperforms almost all the conventional dimension reduction techniques. However, SNE still exposes several drawbacks such as complexity in optimization and crowding problem in output. A series of variants inspired by SNE have been proposed with modification on different portions of SNE framework like the selection of low-D probabilities and further divergence exploration.

2.3 Similarity

NE methods replace distance functions with probabilities to represent the similarities among high-D data points and low-D mapped points. The original SNE chooses a Gaussian distribution for both $p_{j|i}$ and $q_{j|i}$ calculated by the exact value of vector elements for vectorial data form. However, such Gaussian kernel yields huge computation expense due to the exponential term especially when the size of dataset is large. In fact, suppose two points i and j are excessively far apart in high-D space, the probability for i and j being neighbors will be extremely small meaning that evaluating every pairwise relations of high-D data points is not essential. Only the points satisfying a threshold of "neighborhood" will be taken into the consideration for similarity calculation. This scheme is achieved by constructing a k nearest neighbor graph to significantly simplify the computation.

2.3.1 k Nearest Neighbor Graph

k-Nearest Neighbor Graph(kNNG) [PS12] is developed from the classical k-Nearest Neighbor algorithm by adding a process of graph construction. Consider n objects existing in the space as a set P. For vertices p_k and p_j in P, evaluated by a certain kind of metric such as Euclidean distance or cosine distance, if the distance between p_k and p_l is among the kth smallest distances from p_k to all the other vertices, then p_l is one of the k nearest neighbors of p_k. Adding an edge between p_k and p_l to represent such neighborhood, the k-Nearest Neighbor Graph is constructed if all the corresponding edges are connected. Though kNN is commonly considered



as a classifier algorithm in machine learning, it also makes a great contribution in reducing the complexity and expense of computing the SNE probabilities. The scheme of preserving only the close neighbors naturally matches the emphasis of local structure in SNE method. The implementation of kNN graph displays three obvious advantages that are

1. The local structure gains better accuracy by removing the possible interactions too far away.

2. The computation acquires better scalability. With only nearest neighbor points reserved for a certain data point, it is simple to sparsify the data matrix and utilized high performance methods for sparse matrix.

3. A variety of efficient and matured algorithms have been developed to generate kNN graph, also available for big dataset.

These advantages motivate NE users to implement kNN graph for improvements with a series of accessible construction methods. The most naive way to construct a kNN graph is by brute forces which means for point p_k , pick the k points with 1st to kth smallest distances towards p_k after calculating the pairwise distances between

ALL pairwise data points. In most cases, it is not possible at all to carry out such computation since it has a complexity at least $O(N^2)$. Thus efficient algorithms for kNN graph construction are critical.

Nowadays, popular methods of kNN graph construction in general can be divided into three schools that are space-partitioning trees, local sensitive hashing and neighbor exploring techniques. We focus on introducing the most common-used and matured school, the space-partitioning tree school with subgroups as exact tree and approximate tree.

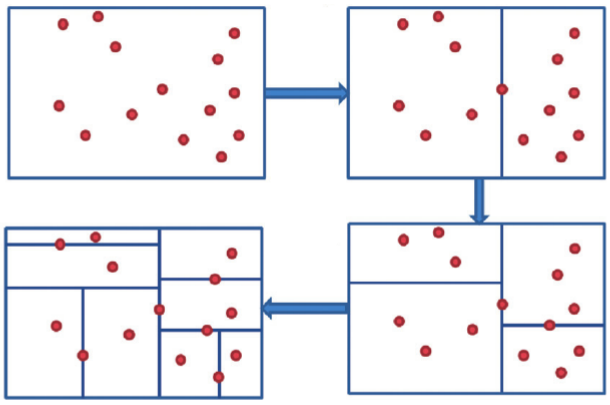
2.3.1.1 k-Dimensional Tree

k-d tree [Ben75] is a binary tree with each node being a k dimensional data point. Each non-leaf node can be regarded as a hyperplane that bisects the space of the k dimensional dataset. Also every node in the tree corresponds to a hyperrectangle in the space.

The idea of constructing the k-d tree is similar to the binary search tree while there is a difference in how to decide the child node that one single data point belongs to. k-d tree manipulates the bisection due to certain dimensions D with greatest variances in order to perform the best separation. That is to choose one dimension D_k at a time, then divide the k-dimensional space with a hyperplane vertical to dimensional D_k. Values of all the k dimensional data points at one side of D_k are smaller than the ones on the other side. Doing such bisection recursively, the k dimensional space will be divided into smaller subspaces corresponding to deeper nodes in the tree until no more bisection can be done meaning that we have reached the leaf nodes. Two main questions of the k-d tree construction still remain (1) on which dimension should we do the bisection and (2) how can we ensure that the amounts of points in the two subspaces are equal or close.

The first one is done by doing the bisection on the dimension with max variance. Before starting the recursive process, we can calculate the variance of every dimension and then sort. After that, following the sequence of dimensions with variances from great to small, the recursive bisection can be done. And the second question is solved by locating the bisection hyperplane vertical to the manipulated dimension just at the median of values at this dimension. This attempts to construct a more balanced tree. Figure 1 illustrates a simple example of k-d tree construction in a 2D plane.

Figure 1: k-d Tree Construction on 2-D plane



And the pseudo code for the construction is as follows

Algorithm 1: k-d tree construction pseudo codes

Data: Dataset; Space

Result: k-d Tree

```
if Null Dataset then
    Return null k-d tree;
else
    1.Dimensions for Bisection: Calculate the variances of data on each dimension
    and sort;
    2.Bisection location: Sort the dataset by the value of kth dimension, pick the
    median as the location for bisection on this dimension;
    3.Start form the dimension with largest variance and its median;
    while Subspace able to be bisected = TRUE do
        Left_tree<- points that value smaller than the median on dimension;
        Left_space<- space smaller than the median on located dimension bisected
        by hyperplane;
        Right_tree<- points that value greater than the median on dimension;
        Right_space<- space greater than the median on located dimension
        bisected by hyperplane;
        Move to next bisection location;
    end
end
```

2.3.1.2 Vantage Point Tree

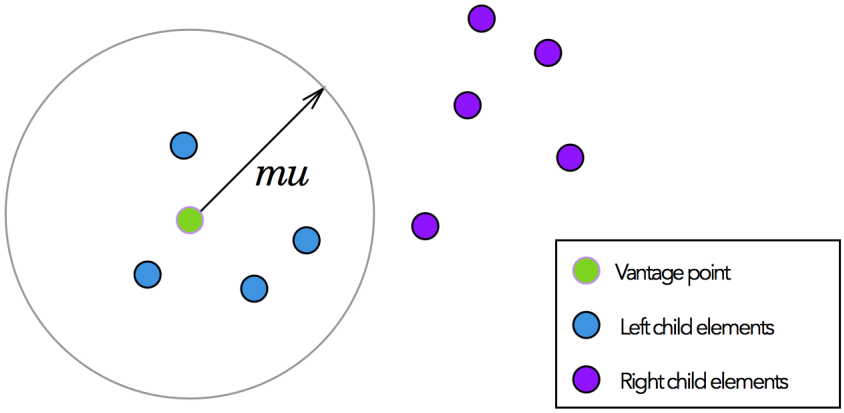
Though k-d tree has the advantage of being simple to construct, it is only feasible when the dimension is moderately high (usually $k < 20$). Because the depth of the tree expands fast, it becomes difficult to find a good bisecting dimension if the space dimension is high. Ball tree is suggested to overcome such drawback using a hypersphere as one node containing a set of data points. However, it still relies on the exact coordinates to determine the space splitting. A further step is called vantage point tree (VP tree) [Yia93] taking only the distances into consideration which brings much convenience in the tree construction.

The route to split the space of dataset is similar to k-d tree but not achieved by splitting the space on certain dimensions. In VP tree, one data point (can be randomly picked) is selected to be the vantage point, also the root node. then the distances from the vantage point to all the other points are computed. And the space

is divided into two parts due to the distances. The general process to construct a VP tree is as follows.

1. Select point v as vantage point.
2. Compute the distance D_i from other point x_i to v.
3. Compute the median M of D_i, allocate points with distance < M to left child branch and others to right child branch.
4. Recursively build left and right branch until leaf nodes reached.

Figure 2: VP splitting illustration



2.3.1.3 Approximation Tree

VP tree improves the performance of kNN graph construction. However the distances are still based on the exact computation which brings much numerical work hard to tackle. A natural perspective is to replace the exact methods with approximate ones so that datasets with much higher dimension can be coped with.

Recent years multiple prevalent approximation methods for kNN graph construction have been proposed such as FLANN [ML14], KGraph [DML11] and LargeVis [TLZM16]. These methods have various starting points. FLANN is a popular library consisting of a series of algorithms for fast kNN construction. It mainly implements the k-d tree but emphasizes the local or nearest nodes by some random techniques like random projection tree. Such randomization and local emphasis lessen the computation burdens caused by those far and less important neighbors. KGraph applies a delicate idea that "he neighbor of my neighbor is more likely to be my neighbor".

After randomly picking a small subset of data points, a neighbor graph is developed by expanding from each of the data point within the subset. Repeating the process we will obtain a global kNN if we compare and merge the generated graphs together. LargeVis also gets inspiration from Kgraph. kNN graph is constructed in two stages. First step is to do a space splitting by random projection tree obtaining a rough kNN graph. Then, with the concept of "neighbor of my neighbor", implement the neighbor exploring techniques to improve the tree.

In summary, by constructing kNN graphs, the number of $p_{j|i}$ we need to compute is greatly reduced. Dealing with large datasets is a complicated procedure, and efficiency is always critical in every phase.

2.3.2 Entropic Affinity

Notice that in original SNE, $p_{j|i}$'s Gaussian kernel has a specific variance according to data point i and variance of $q_{j|i}$ is set to $\frac{1}{2}$ over all low-D points. To interpret this intuitively, we assume variance σ_i being the radius of a hypersphere centered at point x_i containing a certain number of neighbors, suppose k neighbors. From a view of information theory, a hypersphere holding more points contains higher entropy which means that together with its neighbors, x_i contains more information. To ensure all the high-D points have equal entropy, the variance of each point need to be properly tuned. In the zone where points are densely accumulated, σ_i should be comparably small while in sparse zone, the variance should be large. Thus an outlier may lead to a Gaussian kernel approximate to a uniform distribution with too large variance.

SNE adopts the concept *Perplexity* as the parameter to control variance which is the measure of entropy within neighborhood conditional on a data point. For the probability distribution $P(i)$ of data point x_i specified by variance σ_i, perplexity $Perp(P_i)$is defined as

$$Perp(P_i) = 2^{H(P_i)}$$

where $H(P_i)$ is the bits-measured Shannon entropy of P_i with the form

$$H(P_i) = -\sum_j p_{j|i} \log_2 p_{j|i}.$$

van der Maaten [vdMH08] interprets perplexity as it smoothly determines how many neighbors of a certain data point are effective.

In fact, with determined perplexity, the calculation of σ_i is expensive and tricky. Hinton & Roweis [HR02] achieved an interval starting from $[0, 1]$ via searching techniques that are inefficient with large dataset. Vladymyrov & Carreira-Perpinán proposed an efficient algorithm for the computation of *entropic affinity*, a more generalized comprehension of σ_i [VCP13].

To introduce the entropic affinity, we start from a another perspective by Carreira-Perpinán [VCP13]. For a given set of finite data points $x_1, \cdots, x_N$ in space R^d, the conditional probability of x_i choosing x_j is

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-\|x_i - x_k\|^2/2\sigma_i^2)}$$

Now with fixed x_i, consider this probability as a function of input point x given the scale parameter a.k.a width, σ. The probability can be rewritten as

$$p_i(x;\sigma) = \frac{K(\|\frac{x-x_i}{\sigma}\|^2)}{\sum_{k=1}^N K(\|\frac{x-x_k}{\sigma}\|^2)} = \frac{K((\frac{d_i}{\sigma})^2)}{\sum_{k=1}^N K((\frac{d_k}{\sigma})^2)}$$

where $d_k = \|x - x_k\|$. SNE is a special case that K is a Gaussian kernel.

With such transformation, the entropy $H(x, \sigma)$ of $p_i(x;\sigma)$ is expressed as a function of $d_1, \cdots, d_N$

$$\begin{aligned} H(x,\sigma) = &-\sum_{i=1}^N p_i(x;\sigma)\log p_i(x;\sigma) \\ &-\sum_{i=1}^N p_i(x;\sigma)\log K_i + \log\sum_i K_i \end{aligned} \tag{2}$$

In particular, the Gaussian kernel has the form

$$H(x,\beta) = \beta\sum_{i=1}^N p_i(x;\beta)d_i^2 + \log\sum_{i=1}^N \exp(-d_2^2\beta)$$

where $\beta = \frac{1}{2\sigma^2}$

If we set a specific σ_k value for each data point x_k to ensure entropy of each point equalling to $\log K$ set by user, the corresponding $p_i(x;\sigma)$ can be implemented as affinity between x and x_i. Such $p_k(x;\sigma)$ values are *entropic affinities* for that the affinities are sought by aligning the point-specified entropy.

With a clear definition, the problem of searching for point-individual σ or β is transformed into a 1-D root finding problem of the equation

$$F(x,\beta,K) := H(x,\beta) - \log K = 0$$

with given perplexity K.

Vladymyrov and Carreira-Perpinán gave an explicit solution for the lower and upper bound of β as

$$\beta_L = \max\left(\frac{N \log \frac{N}{K}}{(N-1)\Delta_N^2}, \sqrt{\frac{\log \frac{N}{K}}{d_N^4 - d_1^4}}\right)$$

$$\beta_U = \frac{1}{\Delta_2^2} \log\left(\frac{p_1}{1-p_1}(N-1)\right)$$

where p_1 can be obtained by solving

$$2(1-p_1) \log \frac{N}{2(1-p_1)} = \log(\min(\sqrt{2N}, K))$$

taking the solution within interval $[\frac{3}{4}, 1]$.

For each point, the complexity of searching for the bounds is $O(1)$, and the total cost is of $O(N)$. A scalable and efficient algorithm is proposed by Vladymyrov and Carreira-Perpinan. For proofs and algorithm details, see [VCP13].

2.4 Divergence

In general, NE methods achieve the dimension reduction by minimizing the discrepancy between similarities among high dimensional data points and those among low dimensional mapped points. Technically, the minimization is carried out by optimizing the information divergence between neighborhoods of the input high-D and output low-D spaces. Two major classes of NE methods are 1) combining a non-separable divergence with normalized neighborhoods, e.g SNE; 2) combining a separable divergence with non-normalized neighborhoods, e.g Elastic Embedding [CP]. The definition of above separable divergence is a divergence as the sum of pairwise terms where each term is only determined by the locations of one pair of data points. Whether a divergence is separable results in different properties in NE objectives[YPK14]. Separable divergence makes NE cost functions easier to design and optimize while harder to resolve the tradeoff between the attraction and repulsion. On the other hand, non-separable based objective functions are scale invariant and simpler to determine the tradeoff while the optimization will be cumbersome. The main divergence families of both classes will be introduced in the following part. Also a theorem which proves the optimization equivalence between separable and non-separable divergences is displayed. It provides a guideline of divergence selection and optimization.

2.4.1 Divergence Family

Divergence $D(P\|Q)$ a.k.a information divergence is defined to measure the discrepancy between probability distribution P and Q with basic properties that $D(P\|Q) \geq 0$ and $D(P\|Q) = 0$ iff $P = Q$. The concept can be promoted to more generalized form like tensor or matrix. There are four important divergence families α, β, γ and Rényi with abundant variants covering a lot of applications and they are defined as

$$D_\alpha(P\|Q) = \frac{1}{\alpha(\alpha-1)} \sum_i [p_i^\alpha q_i^{1-\alpha} - \alpha p_i + (\alpha-1) q_i]$$

$$D_\beta(P\|Q) = \frac{1}{\beta(\beta-1)} \sum_i [p_i^\beta + (\beta-1) q_i^\beta - \beta p_i q_i^{\beta-1}]$$

$$D_\gamma(P\|Q) = \frac{\ln(\sum_i p_i^\gamma)}{\gamma-1} + \frac{\ln(\sum_i q_i^\gamma)}{\gamma} - \frac{\ln(\sum_i p_i q_i^{\gamma-1})}{\gamma-1}$$

$$D_r(P\|Q) = \frac{1}{r-1} \ln(P_i^r Q_i^{1-r})$$

where p_i and q_i are non-normalized probability entries of P and Q respectively and $P_i = p_i / \sum_k p_k$ as well as $Q_i = p_i / \sum_k q_k$ are normalized ones. These families of divergences contains multiple most popular divergences in machine learning such as

normalized Kullback-Leibler divergence

$$D_{KL}(P\|Q) = \sum_i P_i \ln \frac{P_i}{Q_i}$$

transformed from D_r with $r \to 1$ or D_γ with $\gamma \to 1$

non-normalized Kullback-Leibler divergence

$$D_I(P\|Q) = \sum_i (p_i \ln \frac{p_i}{q_i} - p_i + q_i)$$

transformed from D_α with $\alpha \to 1$ or D_β with $\beta \to 1$

Itakura-Saito divergence

$$D_{IS}(P\|Q) = \sum_i (\frac{p_i}{q_i} - \ln \frac{p_i}{q_i} - 1)$$

transformed from D_β with $\beta \to 1$

Other special cases include Euclidean distance($\beta = 2$), Chi-square divergence($\alpha = 2$) and so on. All these variants of divergences play critical roles in multiple research fields for instance D_{KL} is popular in text processing and D_{IS} is common in audio signal analysis.

2.4.2 Optimization Equivalence

The four major divergence families have clear pros and cons that meet the distinct preferences depending on the implemented algorithms. α and β divergences are separable divergences that are easy to obtain their derivatives indicating a mathematical and computational simplicity in design and optimization for stochastic or distributed computation. However, by the nature of additive terms, the separable divergences are rather sensitive to the scale of P and Q. On the other hand, the non-separable ones, γ and Rényi divergences, are scale invariant which is desirable in various algorithms due to the log terms. However being non-separable brings extra difficulty in optimization since the entries are messed together and it is also hard to transform the divergence into a variant because of the complicated functional form.

Early research mainly concentrated on the internal relationships inside one family rather than the relationships between two divergence families. Yang and Peltonen proposed an optimization equivalence revealing the inter-relationships that connect the four divergence families.

Theorem 1. *[YPK14] For $p_i \geq 0$, $q_i \geq 0$, $i = 1, \cdots, N$ and $\tau \geq 0$,*

$$\arg\min_Q D_{\gamma\to\tau}(P\|Q) = \arg\min_Q [\min_{c\geq 0} D_{\beta\to\tau}(P\|cQ)]$$

$$\arg\min_Q D_{r\to\tau}(P\|Q) = \arg\min_Q [\min_{c\geq 0} D_{\alpha\to\tau}(P\|cQ)]$$

The scalar c has an optimal value with closed form as

$$c^* = \arg\min_c D_{\beta\to\tau}(P\|cQ) = \frac{\sum_i p_i q_i^{\tau-1}}{\sum_i q_i^\tau}$$

$$c^* = \arg\min_c D_{\alpha\to\tau}(P\|cQ) = \left(\frac{\sum_i p_i^\tau q_i^{1-\tau}}{\sum_i q_i}\right)^{\frac{1}{\tau}}$$

with a special case $c^* = \exp(-\frac{\sum_i q_i \ln(q_i/q_i)}{\sum_i q_i})$ for $\alpha \to 0$. The proof is done in [YPK14] and such equivalence holds for both global and local minima. The following proposition ensures the property

Proposition 1. *[YPK14] The stationary points of $D_{\gamma\to\tau}(P\|Q)$ and $D_{\beta\to\tau}(P\|cQ)]$, as well as of $D_{r\to\tau}(P\|Q)$ and $D_{\alpha\to\tau}(P\|cQ)]$ in Theorem 1 are the same.*

Theorem 1 and its proposition give us large flexibility by taking advantages of both separable and non-separable divergences. When constructing a cost function for a

certain algorithm by modifying a divergence, separable divergence will be a start line with optimization and weighted techniques inserted. Then turn to the non-separable form to achieve the scale invariant property. For an optimization task, we can transform the problem to its corresponding separable counterpart for simplicity and efficiency. Hinton used a physical graph of springs-net to describe the tradeoff based on force-directed idea [HR02]. While Theorem.1 produces a mathematical explanation for the attraction and repulsion tradeoff, displayed as c, that c is adjusted during the optimization process inside the separable divergence just in order to make the divergence perform a scale invariant property like a non-separable as good as possible [YPK14].

2.5 Selection of q

As mentioned in NE framework, the original SNE encounters two main drawbacks that are optimization complexity and crowding problem, very common in dimension reduction research. Hinton reported that seeking an embedding of only 3000 data points will cost several hours with steepest descent method which is intolerably expensive [HR02]. This is mainly caused by both the design of low dimension probability q and the optimization framework [vdMH08]. Starting from the design of q, great effort has been put into finding the optimal q expression.

2.5.1 Symmetric SNE

The first step is done by replacing the conditional probability $q_{j|i}$ with a joint probability q_{ij} in low dimensional space. It brings a symmetric version of probabilistic description for similarities among data points, that is for $\forall i, j$, $p_{ij} = p_{ji}$ and $q_{ij} = q_{ji}$, still $p_{ii} = q_{ii} = 0$. Such symmetry reduces the difficulty to optimize the cost function. The low dimensional probability has the form

$$q_{ij} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq l} \exp(-\|y_k - y_l\|^2)}$$

An instant imitation can be extended to high dimensional space that

$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma^2)}{\sum_{k \neq l} \exp(-\|x_k - x_l\|^2/2\sigma^2)}$$

However, such intuition gives rise to another flaw. Suppose there exists an outlier x_k extremely far away from other points resulting that to all $j \neq k$, p_{kj} is extremely

small. This means no matter where its corresponding y_k is mapped in low-D space, it will not effectively influence the cost function. Thus, to avoid this flaw, the joint probability in high-D space is defined as

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2n}$$

where n is the amount of data points. This definition assures the satisfaction of symmetry and keeps p_{ij} above a lower bound that $\sum_j p_{ij} > \frac{1}{2n}$ indicating every x_k can make considerable contribution to the cost function.

The symmetric probabilities simplify the cost function as

$$C = KL(P\|Q) = \sum_i \sum_{j \neq i} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

and the gradient turns to be

$$\frac{\delta C}{\delta y_i} = 4 \sum_{j:j \neq i} (p_{ij} - q_{ij})(y_i - y_j)$$

Compared with original SNE, symmetric SNE (SSNE) has a much simpler gradient. van der Maaten [vdMH08] claims that SSNE slightly performs greater efficiency than the original one. Thus further improvement is still necessary.

2.5.2 UNI-SNE

Besides the optimization complexity, a phenomenon called crowding problem also bothers the users of SNE especially in the data visualization task in 2-D output space.

- **Crowding Problem** [vdMH08]
  The crowding problem is common in data visualization after reducing the dimensions. In fact, crowding problem is not caused by a definite algorithm, it is caused by the difference of the distance distributions between high and low dimension spaces.
  For example, in 2-D plane, we can find 3 equidistant points while in 10-D space, there exist 11 equidistant points. However such high-D equidistant relations cannot be faithfully and completely preserved and presented in much lower dimensions say 2-D. This mean the room in low-D space for high-D dense areas is not enough to unfold the relations honestly. Multiple clusters might crush and mutually overlap. Thus no clear boundary dividing clusters can be observed.

Cook and Hinton [CSMH07] initiated resolving the crowding problem by adjusting the expression of low-D probabilities. The key point is that by strengthening the repulsion among mapped low-D points, the formed clusters can be further partitioned. Extra background forces with uniform distribution are added into q_{ij} turning q_{ij} to

$$q_{ij} = \frac{(1-\lambda)\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq l} \exp(-\|y_k - y_l\|^2)} + \frac{2\lambda}{N(N-1)}$$

where λ is the parameter balancing the background effect and points interactions.

With the background force addition, no matter how far apart the mapped points are, q_{ij} is always greater than $\frac{2\lambda}{N(N-1)}$. Thus, for those far apart points in high-D space, their corresponding mapped low-D points received slightly stronger repulsion. The distance in low-D space is expanded to reduce the larger q_{ij} in order to decrease the difference between p_{ij} and q_{ij}.

UNI-SNE usually outperforms the original SNE. However UNI-SNE needs special tricks to optimize otherwise poor result will be given that two clusters may depart in very early stage and expand to nonsense mapping after optimization iterations [vdMH08].

2.5.3 t-SNE

Taking a big step forward to dealing with crowd problem, t-SNE absorbs the perspective of UNI-SNE to produce stronger low-D repulsion. Instead of adding uniformly distributed background influence, t-SNE adopts heavy-tailed Student-t distribution to determine the low-D probabilities. q_{ij} is set as

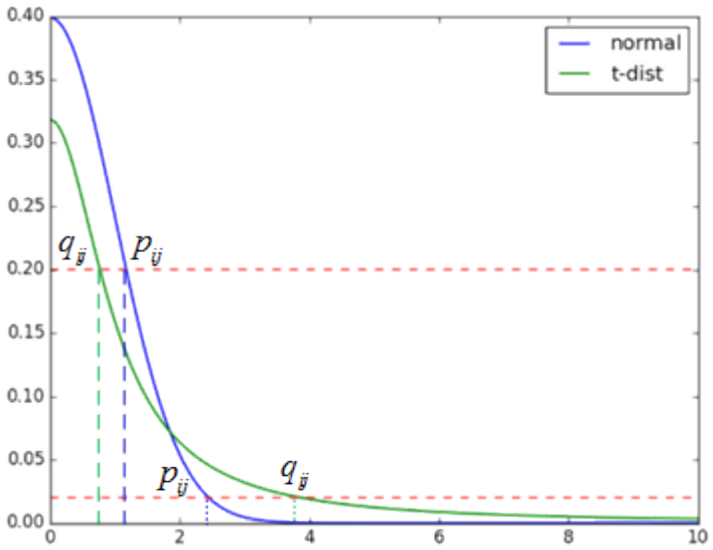
$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l}(1 + \|y_k - y_l\|^2)^{-1}}$$

The Student-t distribution is chosen for multiple reasons [vdMH08]. First, t distribution is tightly correlated with Gaussian distribution as it can be decomposed into infinite Gaussian summation. Then the kernel $(1 + \|y_i - y_j\|^2)^{-1}$ follows an inverse square law of low-D distance resulting in a larger gap between mapped points of dissimilar data points. Furthermore, the Student-t kernel displays a critical property as being approximately scale invariant to the change of low-D mapping. And at last, t-SNE is computationally cheaper since it contains no exponential terms.

In summary, the natural property of t distribution promotes t-SNE to outperform the original SNE. For outliers, t distribution's heavier tails guarantee the robustness

of low-D probabilities. And principle of weakening the crowding problem can be interpreted as follows. Suppose Gaussian distribution corresponds to the probability in high-D space and t distribution corresponds to low-D one. We make the best effort to minimize the difference between high-D data manifold and low-D mapping which is to make p_{ij} and q_{ij} as close as possible. For distant pairwise data points i and j in high dimensional space, high-D probability p_{ij} is smaller than q_{ij} taking normalized distance as variable. Thus in order to realize $p_{ij} = q_{ij}$, as p_{ij} is fixed by high-D data points, a smaller q_{ij} is reached corresponding to a further distance that repels low-D mapped points further. Inversely, the close data points x_i and x_j are also attracted closer. Figure 3 illustrates a rough simulation of this pattern between normal and t distributions.

Figure 3: Influence of tails for close and far apart data points



The cost function of t-SNE is the same as Symmetric SNE

$$C = KL(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

and its gradient is

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1}.$$

van der Maaten discussed further in the following work to alter the t-SNE for high efficiency [vdM14]. A kNN graph was implemented to capture most valuable local data structures in order to reduce the computation expense. He also made clearer

interpretation of t-SNE by decomposing the gradient of cost function into the attraction and repulsion terms. Since the gradient with respect to y_i was considered as the composition of forces affected on y_i, he made the transformation on the gradient that

$$\begin{aligned}\frac{\delta C}{\delta y_i} =& 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1} \\ =& 4\sum_j (p_{ij} - q_{ij}) q_{ij} Z(y_i - y_j)\end{aligned} \tag{3}$$

where $Z = \sum_{k \neq l}(1 + \|y_k - y_l\|^2)^{-1}$ is the normalization term.

One step further, it turned out to be

$$\frac{\delta C}{\delta y_i} = 4(F_{attr} + F_{rep}) = 4\Big(\sum_{i \neq j} p_{ij} q_{ij} Z(y_i - y_j) + \sum_{i \neq j} q_{ij}^2 Z(y_i - y_j)\Big).$$

The decomposition of gradient inspired van der Maaten to implement tree-based methods such as Barnes-Hut method for optimization acceleration and obtained significant improvements [vdM14]. The related optimization techniques will be introduce in Section 4.

t-SNE acquires great popularity since it is proposed for its superiority in data visualization splitting crowding clusters to view the unfolded manifold clearly. However, t-SNE also needs careful tuning of multiple parameters. The optimization process of t-SNE cost function is still tedious and cumbersome with space for promotion. On the other hand, the crowding problem is not perfectly solved by t-SNE yet. This both prompt more innovation on NE algorithm family.

2.5.4 Elastic Embedding

Similar with van der Maaten's work in t-SNE, Carreira-Perpinán [CP] proposed a generic form of NE cost function decomposing the target into attraction and repulsion terms with a parameter λ indicating the tradeoff. The generic form of cost function is

$$C = E(Y, \lambda) = E^+(Y) + \lambda E^-(Y)$$

Combining the main notion of NE with the implementation of spectral method, Perpinán also developed a bridge called Elastic Embedding (EE) [CP] between SNE and conventional spectral methods by applying graph Laplacian. For EE, the exact

cost function is

$$E(Y,\lambda) = \sum_{i,j}^{N} p_{ij}\|y_i - y_j\|^2 + \lambda \sum_{i,j}^{N} \exp(-\|y_i - y_j\|^2)$$

And the corresponding gradient is

$$\nabla E(Y) = 4Y(L_p - \lambda L_q)$$

with graph Laplacian terms

$$L_p = D_p - W_p \quad L_q = D_q - W_q.$$

Here W_p and W_q represent the affinity matrices where elements at ith row and jth column are p_{ij} and $q_{ij} = \exp(-\|y_i - y_j\|^2)$ respectively. D is the diagonal degree matrix in which the ith element of D_p is $D_{(p)ii} = \sum_j p_{ij}$. D_q has the same pattern that $D_{(q)ii} = \sum_j q_{ij}$.

Yang [YPK14] proved EE being a separable divergence variant of symmetric SNE plus a constant being independent of low-D mappings with Theorem 1 and also gave the best λ value that $\lambda^* = \sum_{ij} p_{ij} / \sum_{ij} q_{ij}$ controlling the tradeoff between attraction and repulsion. The spectral nature of EE also allows an out-of-sample plugging-in that is very precious in NLDR.

2.5.5 Weighted Symmetric and t-SNE

At the same time when Theorem 1 is proved, Yang proposes a new variant [YPK14] of symmetric SNE that is closely related to the EE method. This variant starts from modifying the EE method to make the method able to deal with both vectorial data and graph layout. In this variant [YPK14], the edge repulsion strategies borrowed from Noack's work[Noa07] in graph drawing domain are plugged into the EE [CP] cost function because of its pairwise separable property. Weights M are inserted into the repulsive term in the EE cost function, thus the EE cost function becomes

$$C_{weighted-EE} = \sum_{ij} p_{ij}\|y_i - y_j\|^2 + \lambda \sum_{ij} M_{ij} \exp(-\|y_i - y_j\|^2)$$

where $M_{ij} = d_i d_j$ and vectors d_i, d_j are centrality measures indicating how important the points i and j are. Options of centrality measures include closeness, betweenness, degree centrality and eigenvector centrality. Here the degree centrality is selected to represent the point importance. Two disadvantages of weighted EE method are

that the cost function are sensitive to the scale of p and parameter λ need a manual tuning. By the flexibility from divergence optimization equivalence, Yang gave a simpler solution that can avoid such disadvantages.

Proposition 2. *[YPK14] Weighted EE is a separable divergence minimizing method and its non-separable variant is weighted symmetric SNE (ws-SNE).*

Thus a novel variant of SNE can be established with a cost function

$$C_{ws-SNE} = -\sum_{ij} p_{ij} \ln q_{ij} + \ln \sum_{ij} M_{ij} q_{ij} + const$$

where *const* is a constant with none respect to low-D mapping. The selection of q_{ij} is also flexible. Gaussian kernel leads to the weighted symmetric SNE (ws-SNE) while t-distribution kernel turns it to the weighted t SNE (wt-SNE).

ws-SNE and wt-SNE do not treat all the data points equally important, which is very different from original SNE and its early variants. Based on such equal importance assumption, early SNE methods often perform well on vectorial data in neighborhood form for example with p_{ij} calculated from kNN graph. However, for graph or network layout of data where there exists highly imbalanced degree centrality, SNE usually displays distorted result. SNE has a tendency to place the more important points of different natural clusters closer which is not proper in reality [YPK14].

Suppose each cluster as a galaxy in the universe space, data points as stars should accumulate tightly in their own galaxy around the galaxy mass core, the center. Meanwhile, there should also exist clear gaps among galaxies. ws-SNE realizes a similar pattern using edge repulsion to divide the clusters more clearly. The gradient of ws-SNE

$$\frac{\delta C}{\delta y_i} = \sum_j p_{ij} q_{ij}^{\theta} - c d_i d_j q_{ij}^{1+\theta} (y_i - y_j)$$

where $c = \sum_{ij} p_{ij} / (\sum_{ij} d_i d_j q_{ij})$ is the connection scaler where $\theta = 0$ for Gaussian q, $\theta = 1$ for Student-t q. The first term denotes the attraction composition and the second term for the repulsion with weights d_j and d_j. This provides extra repulsion if the data points are more important. The center of one galaxy with greater mass gains greater repulsion from other galaxy centers leading to a farther place. This offers a more significant solution to the crowding problem than the t-SNE using heavy tail shown in [YPK14]. We make a further step combining the heavy tail $t(1)$ a.k.a Cauchy q_{ij} with ws-SNE method to explore the performance of this modified setting. The result is shown in Section 6.

3 Objective and Gradient Approximation

Neighbor embedding methods display better results than the linear and spectral methods but optimization process of NE used to be a great agony since the cost functions and gradients usually are non-convex and computationally expensive. Generally, NE methods attempt to minimize the cost function controlling the mismatch between high-D input data points and low-D mapped points. The mismatch is transformed to achieving a balance between the attraction and repulsion forces among the low-D mapped points with optimal tradeoff. In this optimization framework, the exact solution is to compute the the interactions between every pair of mapped data points. Such exact solution is direct but unfeasible since that for N data points, the exact computation has an $O(N^2)$ complexity which is easy to cause an overload in memory or a extreme long-run computation. For example, a typical `MNIST` dataset with 70000 entries may cost $70000 \times 70000 \times sizeof(double) = 36.5$ GB space in memory which is intolerably expensive. Pioneers in NE research applied indirect technique to avoid the $O(N^2)$ complexity by only randomly picking a subset of the data so that a genral exploration of big dataset can be carried out [vdMH08]. The random subset in summary is not feasible in most cases since it omitted too much information [VCP12]. Thus solution that can accelerate and attack the computation is vital. Fortunately, methods from multiple views have been developed for the improvement and acceleration for the optimization. The first intuition is to avoid computing the exact cost function and gradient.

One common view is applying the methods according to N-body problems in astrophysics dealing with the gravitational interactions between particles. N-body methods attempts to minimize the potential energy by balancing the particle interactions where NE methods has a similar situation trying to minimize the mismatch by getting the optimal tradeoff between attraction and repulsion. All N-body methods with high efficiency are approximate algorithms where tree-based techniques are powerful tools for the acceleration. Two common used methods will be introduced, the Barnes-Hut tree (BH) and Fast Multipole Method (FMM).

3.1 Barnes-Hut

The cost functions and gradient of NE methods have general form as

$$C = \sum_{i=1}^{n}\sum_{j=1}^{n} A_{ij}, \qquad \frac{\delta C}{\delta y_i} = \sum_{j=1}^{n} B_{ij}(y_i - y_j)$$

where A_{ij} and B_{ij} are scalar terms computed between pairwise data points i and j [YPK13]. These terms usually involve coordinates and pairwise distances in output space that vary during the optimization. Considering the summation form in cost function and gradient, one may utilize an approximation solution to reduce the complexity. For data point i, summing up all the interactions between i and all other j points can be divided into summing up multiple interaction subgroups. Interaction produced by each subgroup is approximated by a representative point such as a mass core or a mean.

NE cost functions usually have a summation as

$$\sum_{j} f(\|y_i - y_j\|^2) = \sum_{t}\sum_{j\in G_t^i} f(\|y_i - y_j\|^2) \approx \sum_{t} |G_t^i| f(\|y_i - \hat{y}_t\|^2)$$

where i is the source point, js are the neighbors of i. G_t^i denotes the subgroup that point j belongs to and $|G_t^i|$ indicates how many points are in the subgroup.

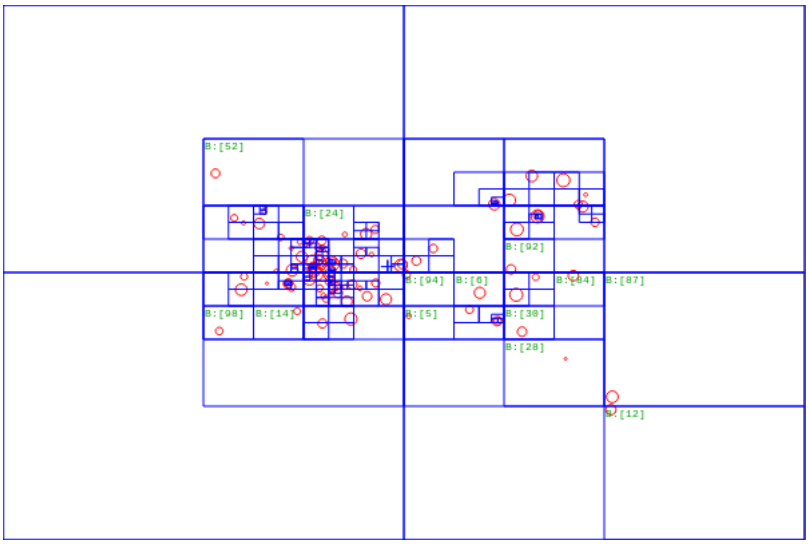
Suppose $g_{ij} = f^{'}(\|y_i - y_j\|^2)$, the gradient can also apply a similar approximation that

$$\sum_{j} g_{ij}(y_i - y_j) = \sum_{t}\sum_{j\in G_t^i} g_{ij}y_i - y_j \approx \sum_{t} |G_t^i| f^{'}(\|y_i - \hat{y}_t^i\|^2)(y_i - \hat{y}_t^i)$$

If the distance between i and subgroup G_t^i is far enough, the diversity of interactions between i and points in G_t^i will not be significant. Hence all the points in G_t^i will be approximated as one. The interaction complexity is reduced from point-to-point to point-to-group. If we repeat the partition action and implement the approximation to smaller subgroup until only one point left in the group, the interaction $f(\|y_i - y_j\|^2)$ or g_{ij} is directly calculated between pairwise points. Such subgroup partition process follows a route similar to constructing a tree hierarchy.

In practice, multiple techniques can be selected to perform above approximation. A common choice is Barnes-Hut tree based on Quadtree for a 2-D output space. It is quite straight forward to construct a Barnes-Hut tree. Notice that the tree depth is correlated with point density that denser area has deeper tree branches. Each tree

Figure 4: A Barnes-Hut tree illustration



node contains a subgroup of data points where G_t indicates the points of subgroup or tree node t inside the edge square. For each G_t, if G_t is far away from i, all the interactions of points inside G_t will be composed as $|G_t| \times \hat{y}_t^i$ where $|G_t|$ is the number of points inside node t and $\hat{y}_t^i$ is the mean of corresponding points.

One important parameter is θ determining the tradeoff between approximation precision and computational cost. θ achieves such function by deciding how far is far enough for a distance between point i and tree node t with an inequality

$$\theta \cdot TreeWidth < \|y_i - \hat{y}_t^i\|$$

where $TreeWidth$ is the edge length of node t's square. The precision of approximation increases with growing up θ and a typical interval for θ is $[1.2, 5]$ [BH86].

Yang successfully applied Barnes-Hut tree in t-SNE acceleration and displayed much better and faster result than random subset method [YPK13]. Since Barnes-Hut has a complexity level of $O(N \log N)$, there still exists space for further improvement.

3.2 Fast Multipole

Barnes-Hut tree generates significant simplification on the optimization complexity from $O(N^2)$ to $O(N \log N)$ accompanying with some restrictions. First, also the most cumbersome one is that the tree size grows exponentially with larger output space dimension, Quadtree for $d = 2$, Octtree for $d = 3$, a 2^p tree for dimension $d = p$. Second, the error of approximation has no bound or estimation. In order to

overcome such drawbacks, another N-body method Fast Multipole Method (FMM) based on grid hierarchy is applied by Carreira-Perpinán [VCP14] combining with a grid partition method.

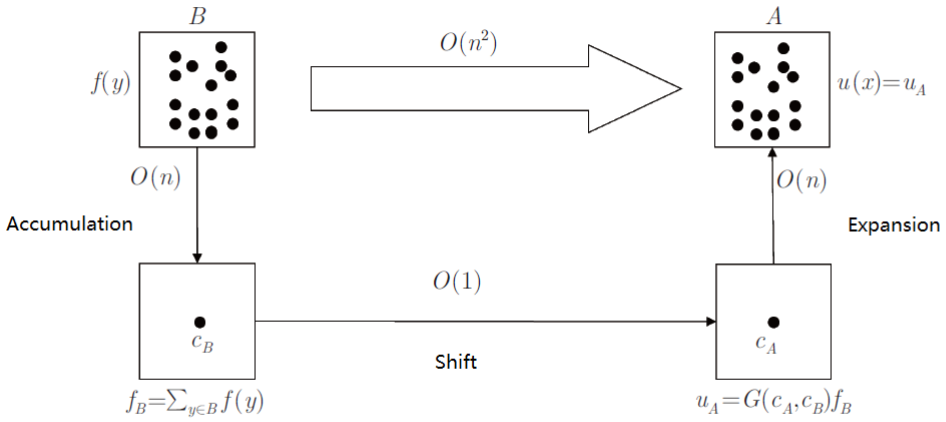
FMM regards the N-body problem as this: with a given set $\mathbf{X} \subset \mathbf{R}^d$ of N target points, a given set $\mathbf{Y} \subset \mathbf{R}^d$ of N source points, kernel function $G(x, y)$, and a weight set of source points $\{f(y) : y \in \mathbf{Y}\}$, we want to calculate the potential function $u(t)$ with respect to every target point $t \in \mathbf{X}$ as

$$u(x) = \sum_{y \in \mathbf{Y}} G(x, y) f(y)$$

This generic form appears in many research domains, for example in astrophysics, $\mathbf{X} = \mathbf{Y}$ indicate the sets of N stars, $G(x, y) = 1/|x-y|$ is the gravitational potential. Direct computation of $u(x)$ requires $O(N^2)$ and it's rather expensive when N is large.

FMM abandons the direct method of brute forces. It decomposes the interactions among target and source points with a 3-stage process as (1)accumulation, (2)shift, (3)expansion for an approximation. Suppose that the dataset distributes in a square $[0, 1] \times [0, 1]$ with normalization and A, B are subsquares of same size without overlapping. The three stage process is shown in Figure 5.

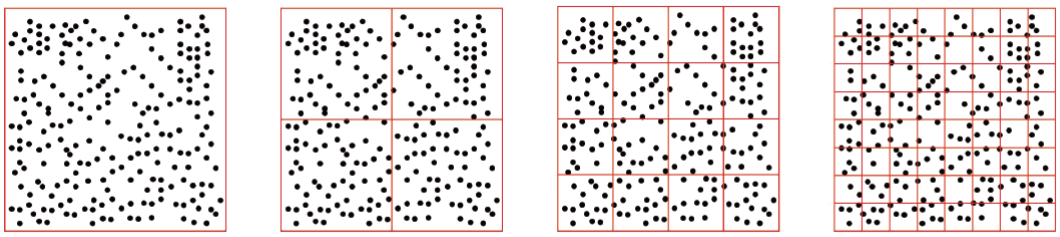
Figure 5: FMM 3 Stages



The first stage accumulation composes the source points in B to the center c_B. Then shift the composed effect to the center c_A of target point in A establishing a

square-square interaction. Finally, execute the expansion on c_A for approximation. Usually, the dataset is normalized in a square and a grid frame is established to ensure the number of points in every square smaller than a threshold. Carreira-

Figure 6: Grid Establishment



Perpinán applied FMM to approximate the cost function and gradient in his work [VCP14]. The Fast Gaussian Transform(FGT) is taken as a generic kernel form as

$$Q(y_i) = \sum_{j=1}^{N} q_j \exp(-\|(y_i - y_j)/\sigma\|^2)$$

Recall the cost function of Elastic Embedding that

$$C(\mathbf{Y}) = \sum_{i,j=1}^{N} \omega_{ij}\|y_i - y_j\|^2 + \lambda \sum_{i,j=1}^{N} \exp(-\|y_i - y_j\|^2)$$

The kernels of cost function and its gradient can both be expressed with modification of the FGT indicating the interaction kernel as $G(x, y)$ introduced above.

Carreira-Perpinán first normalized the dataset to a unit box $[0, 1]$ and partitioned the box with grids. Instead of expanding the interaction form the source points center to the target points center, Carreira-Perpinán did the series expansion locally around EVERY point with a Hermite expansion

$$\exp(-\|(t - s)/\sigma\|^2) = \sum_{k=0}^{\infty} \frac{1}{k!} h_k(\frac{s - s_c}{\sigma})(\frac{t - s_c}{\sigma})^k$$

where t is the target point and s is the source point. $h_k(x) = e^{(-x^2)}H_k(x)$ are the Hermite functions with Hermite polynomials $H_k(x)$. The parameter k controls the tradeoff between precision and computation cost, for a greater k the precision increases and it requires more computation.

Compared with Barnes-Hut tree, though FMM is claimed being able to realize an $O(N)$ complexity, it still remains some suspicions. Suppose the uniform box $[0, 1]$ is partitioned into m subsquares, by Carreira-Perpinán's method, each subsquare will execute the 3-stage expansion on every point. The entire computation then gets a complexity of $O(mN)$. Though the complexity $O(N \log N)$ has not been strictly proved for Barnes-Hut tree, $O(mN)$ of FMM does not always outperform Barnes-Hut tree. For example, the $MNIST$ dataset with 70000 entries has a approximate complexity of $N \log N \approx 12 \times 70000$. While for FMM, the number of subsquares will easily surpass 12 if the grids are tiny enough. Moreover, FMM for NE methods are not feasible for heavy-tailed q_{ij} since the expansion is only applicable for light-tail distributions and relies on their fast decaying property [YPK13]. So, for simplicity and scalability, we apply the Barnes-Hut tree as the approximation techniques in our following works.

4 Optimization

With the approximation of cost function and gradient, the complexity of optimization process decreases to be acceptable. Early developed NE methods applied multiple variants of straight gradient descent methods with numbers of parameters to tune such as the learning step size of gradient descent method and the momentum of annealing simulation. However, it is quite tricky to get the optimal values for such parameters and a long-run optimization may be repeated again and again. Furthermore, the parameters usually highly rely on the certain dataset and are not portable meaning that when the dataset changes, the parameters need to be tuned again. Thus optimization techniques for NE with few or no parameters are developed to avoid the tuning tricks. Three optimization methods that will be discussed in following parts are fixed point [YKXO09], partial Hessian [VCP12] and majorization-minimization [YPK15]. All these methods can provide results that are at least as good as gradient descent methods but much simpler to realize.

4.1 Fixed Point

Optimization based on fixed point borrows the concept of fixed point whose definition is that [GD13]: for a given function φ, if $\varphi(x^*) = x^*$, then x^* is a fixed point of φ. In the other words, the fixed point of φ is the intersection point of function

curve $\varphi(x)$ and straight line $y = x$.

The optimization of cost function by fixed point method is fundamentally a root finding problem that is looking for the zero point of corresponding gradient representing the location of the optima. This can be done following the general path of fixed point iteration which transforms a root-finding problem to a fixed-point finding problem.

The basic setting of fixed point iteration is this: For target function $f(x) = 0$, construct an equivalent function $x = \varphi(x)$ where the root of $f(x) = 0$ is equivalent to the fixed point of $\varphi(x)$. And the iteration follows the procedure as for an arbitrary initial value x_0, compute

$$x_{k+1} = \varphi(x_k) \quad k = 0, 1, 2, \cdots$$

hence an iterative series $x_1, x_2, \cdots, x_n, \cdots$ is obtained approaching the fixed point.

Figure 7: Brief illustration of fixed point iteration

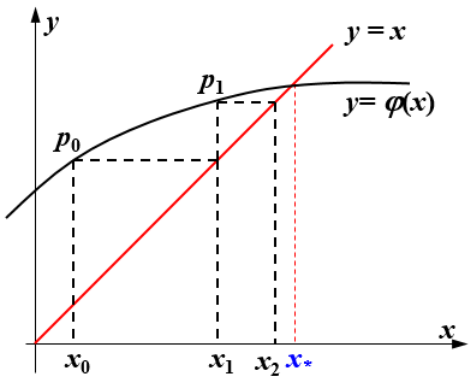


Figure 7 shows an illustration of fixed point iteration. However, in practical work, the convergence of such iteration is not always assured. It requires certain conditions to perform the convergence following a theorem as

Theorem 2. *[GD13] Assume $\varphi(x) \in C[a, b]$ and satisfies*
(1) for $\forall x \in [a, b]$, there is $\varphi(x) \in [a, b]$
(2) there exists a constant L that for $\forall x, y$, it satisfies

$$|\varphi(x) - \varphi(y)| \leq L|y - x|$$

Then for arbitrary initial value x_0, the fixed point $x_{k+1} = \varphi(x_k)$ converges and has the bounds as

$$|x_k - x^*| \leq \frac{L}{1 - L}|x_k - x_{k-1}| \leq \frac{L^k}{1 - L}|x_1 - x_0|.$$

It has also been proved that the above convergence is a global property and in general a smaller L brings faster convergence. Figure 8 shows a divergent process of fixed point iteration that does not satisfy the convergence conditions.

The corresponding pseudo code for fixed point iteration is

Algorithm 2: Fixed point iteration

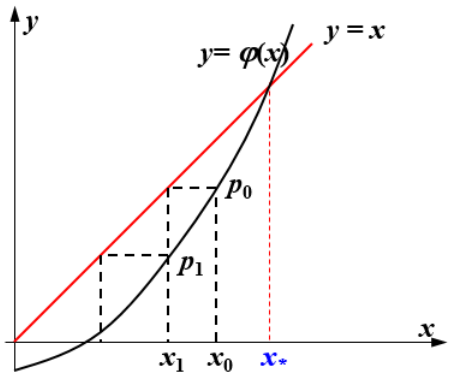
Data: function $f(x) = 0$

input : Function $f(x) = 0$

output: An approximate root x^*

```
Convert f(x) = 0 to the form x = φ(x);
Initialize x_0;
while convergence criterion not met do
    x_{i+1} = φ(x_i)
end
```

Figure 8: Example of diverge fixed point iteration



Implementation in NE

The details of implementing fixed point method to optimize the cost function of NE are interpreted in [YKXO09]. In this article, a general gradient function of different heavy-tailed low-D probability distribution is proposed, which is not only restricted to the t distribution with 1 degree of freedom. Recall the optimization task for symmetric NE methods, the target or the cost function has the form

$$C(Y) = D_{KL}(P\|Q) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{Q_{ij}}$$

where $Q_{ij} = q_{ij} / \sum_{k \neq l} q_{kl}$ are the normalized low-D similarities. The cost function can be transformed to an equivalent form as

$$\underset{q,Y}{maximize}\, \mathcal{L}(q, Y) = \sum_{ij} p_{ij} \log \frac{q_{ij}}{\sum_{k \neq l} q_{kl}}$$

$$\text{subject to } q_{ij} = H(\|y_i - y_j\|^2)$$

where $H(\tau)$ is the embedding similarity function which can be any function as long as it is monotonically decreasing with respect to positive τ. By adding Lagrangian multiplier to the equivalent cost function, we can get the gradient with respect to y_i as

$$\frac{\partial C(Y)}{\partial y_i} = 4 \sum_j (p_{ij} - Q_{ij}) S(\|y_i - y_j\|^2)(y_i - y_j)$$

where $h(\tau) = dH(\tau)/d\tau$ and

$$S(\tau) = -\frac{d \log H(\tau)}{d\tau}$$

is the negative score function of H and we denote $S_{ij} = S(\|y_i - y_j\|^2)$.

Function S can be considered as the tail-heaviness function and H is the similarity function. The heavy-tailed distribution family thus can be written as

$$H(\tau) = (\alpha\tau + c)^{-1/\alpha}$$

where a larger α indicates heavier tails. Gaussian similarity takes $\alpha = -1, c = 0$ and Student-$t(1)$ has $\alpha = 1, c = 1$.

With the above generic form of heavy-tailed SNE family, the iterative update solution is given by setting the partial derivative of cost function $\partial C / \partial y_i = 0$ as

$$Y_{ki}^{(t+1)} = \frac{Y_{ki}^{(t)} \sum_j B_{ij} + \sum_j (A_{ij} - B_{ij}) Y_{kj}^{(t)}}{\sum_j A_{ij}}$$

where $A_{ij} = p_{ij} S(\|y_i^{(t)} - y_j^{(t)}\|^2)$ and $B_{ij} = Q_{ij} S(\|y_i^{(t)} - y_j^{(t)}\|^2)$. The iterative update for $Y_{kj}^{(t+1)}$ can be derived by setting $\partial C / \partial y_j = 0$.

Still the convergence of fixed point ietration is not guaranteed and Yang did a deep discussion in [YKXO09] in which two theoretical justifications of algorithm approximation were provided.

4.2 Spectral Direction

Another popular technique to cope with such non-linear optimization problem is Newton method. Different from the gradient descent methods with first order convergence applied in early NE works, Newton method performs second order convergence showing a much higher speed to converge. A lot of variants originated from Newton method have achieved great successes in many research domains. The basic idea of standard Newton method is that: at the current minimum value, we carry out a second order Taylor expansion according to target function $f(x)$ in order to find out next closer location of global optima. Now suppose our current location is x_k, then we denote

$$\varphi(x) \approx f(x_k) + \nabla f(x_k) \cdot (x - x_k) + \frac{1}{2}(x - x_k)^T \cdot \nabla^2 f(x_k) \cdot (x - x_k)$$

as the second order Taylor expansion of $f(x)$ around x_k. Here $\nabla f(x_k)$ is the gradient and $\nabla^2 f(x_k)$ is the Hessian matrix. For notation simplicity, $\nabla f(x_k)$ is expressed as g_k and $\nabla^2 f(x_k)$ as H_k.

Due to the essential condition of being an optima, $\varphi(x)$ should satisfy $\nabla\varphi(x) = 0$ which means by adding gradient operator on both sides of above expansion, the following is obtained

$$g_k + H_k(x - x_k) = 0.$$

Furthermore, if H_k is non-singular, we can get a solution

$$x = x_k - H_k^{-1} g_k.$$

Hence the iterative procedure for optimization is constructed with a given initial point x_0 as

$$x_{k+1} = x_k - \eta_k \cdot H_k \cdot g_k = x_k + \eta_k \cdot d_k, \quad k = 0, 1 \cdots$$

where $d_k = -H_k \cdot g_k$ is called search direction. η_k is the step size determined by a line search to avoid the procedure falling into poor local optima or diverging. The step size has the property that

$$\eta_k = \arg\min_{\eta \in R} f(x_k + \eta \cdot d_k).$$

In most cases, Hessian mstrix H_k or its inverse H_k^{-1} is very hard to be computed directly. A compromise is to use the approximation of Hessian and the corresponding Quasi-Newton method turns to be attractive. Prevalent Quasi-Newton methods

include DFP [Dav91], BFGS [SK70] and L-BFGS [LN89]. DFP algorithm approximates H_k^{-1} while BFGS and L-BFGS directly approximate H_k iteratively. BFGS keeps the history of first order gradient during the iterations to obtain more accurate result. L-BFGS is the realization of BFGS on huge dataset where how long would the history be kept can be manually chosen to balance the memory consumption and computation precision.

Implementation in NE

The notion of Quasi-Newton is also feasible in NE optimization because Hessian is able to be well approximated by a graph Laplacian [BN01] with probabilistic proximities encoded within. Carreira-Perpinán proposed specific way to carry out Quasi-Newton for NE problems [VCP12]. Recall that Carreira-Perpinán decomposed the cost function as

$$C = E(Y, \lambda) = E^+(Y) + \lambda E^-(Y)$$

where $E^+(Y)$ is the attractive part and $E^-(Y)$ denotes the repulsion.

In most cases a complete Hessian of cost function is not necessary and the *spectral direction* (SD) is purely yielded from the Hessian of attractive part $\nabla^2 E^+(Y) = 4L^+ \otimes I_d$ which is positive semi definite and consisting of $d \times d$ identical diagonal blocks of $N \times N$ with d being the dimension of low-D space. Graph Laplacian L^+ plays an important role in construction the search direction.

Even with the partial Hessian, implementing Quasi-Newton is still often cumbersome. Perpinán in his work applied some more tricks for improvement. First, the Hessian is sparsified to k nearest neighbors. Then small positive ϵ is added to the diagonal of partial Hessian to guarantee the positive definiteness. At last, instead of calculating search direction $d_k = -B_k^{-1} \cdot g_k$ where B_k is the partial Hessian, he tried to solve a linear system $B_k \cdot d_k = -g_k$. For better performance, a Cholesky decomposition is executed on B_k turning the system to $R_k^T \cdot R_k \cdot d_k = -g_k$. After twice backsolving, the search direction or the spectral direction is acquired. Then following other Quasi-Newton steps, the optimization is carried out. For more details, please see [VCP12] and its supplement.

4.3 Majorization-Minimization

Besides Newton method, Majorization-Minimization (MM) [OR00] is also an optimization technique that can apply the second order convergence. The prevalent Expectation-Minimization (EM) algorithm [DLR77] is just a special case of MM algorithm. The main idea of MM algorithm is simple that if the target function $J(x)$ is hard to optimize, we will seek a replacement function $G(x)$ which is simpler to optimize. As long as $G(x)$ satisfies the following three conditions, the minimum of target function $J(x)$ thus can be infinitely approximated by the minimum of $G(x)$. The conditions $G(x)$ should satisfy are

- $G(x)$ is easy to optimize.
- At step k, $G_k(x) \geq J(x_k)$ for all x.
- $G(x_k) = J(x_k)$

For a minimization procedure, $G(x)$ majorizes the target function $J(x)$ and "slides" on $J(x)$ during the iteration. The iteration for optimization can be described as below

1. Set $k = 0$, initialize x_0.
2. Choose G_k such that $G_k(x) \geq J(x_k)$ for all x and $G_k(x_k) = J(x_k)$.
3. Set x_{k+1} as the minimizer of $G_k(x)$.
4. Set $k = k + 1$ and go to Step 2.

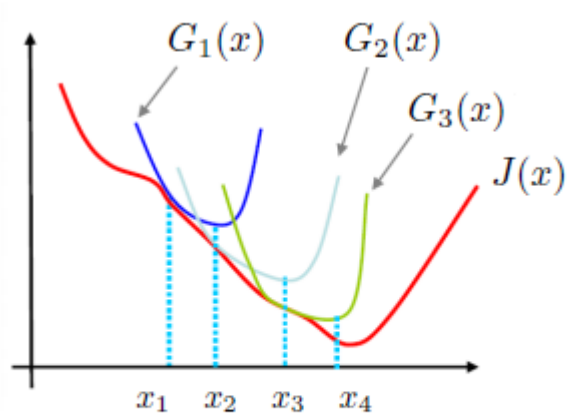
Figure 9 illustrates the optimization iteration. From the above description we can see the key point to apply MM method is whether we can construct a tight majorization or bound of the target function. Many matured solutions have been proposed including quadratic bound, Lipschitz upper bound, Jensen's inequality etc.

Implementation in NE

The implementation of MM method in NE also relies on the decomposition of cost function [YPK15]. Now assume the cost function can be divided into two parts

$$J(\tilde{Y}) = A(P, \tilde{Q}) + B(P, \tilde{Q})$$

Figure 9: Majorization-minimization iterative process



Here Y, $\tilde{Y}$, and Y^{new} denote the current estimate, the variable and the new estimate respectively. The proximities Q and $\tilde{Q}$ are computed from the current estimate and the variable respectively.

The $A(P, \tilde{Q})$ term can be upper bounded by a summation of quadratic pairwise terms that $A(P, \tilde{Q}) \leq \sum_{ij} W_{ij} \|\tilde{y}_i - \tilde{y}_j\|^2$+constant where the multipliers W_{ij} do not depend on $\tilde{Y}$. Part $B(P, \tilde{Q})$ has an Lipschitz surrogate upper bound as $\langle \Psi, \tilde{Y} - Y \rangle + \frac{\rho}{2} \|\tilde{Y} - Y\|^2$+constant where $\Psi = \frac{\partial B}{\partial \tilde{Y}}|_{\tilde{Y}=Y}$ and ρ is the Lipschitz constant.

With the above upper bounds based on quadratification and Lipschitzation, the majorization function $G(\tilde{Y}, Y)$ is thus constructed where

$$G(\tilde{Y}, Y) = \sum_{ij} W_{ij} \|\tilde{y}_i - \tilde{y}_j\|^2 + \langle \Psi, \tilde{Y} - Y \rangle + \frac{\rho}{2} \|\tilde{Y} - Y\|^2 + const$$

Setting the gradient of $G(\tilde{Y}, Y)$ with respect to $\tilde{Y}$ to zero and solving for Y, we can get the iterative update rule

$$Y^{new} = (2\mathbf{L}_{W+W^T} + \rho \mathbf{I})^{-1}(-\Psi + \rho Y)$$

where $\mathbf{L}_{W+W^T}$ denotes the graph Laplacian of matrix $W + W^T$.

Taking the implementation of MM in t-SNE as an example. In t-SNE, $A(P, \tilde{Q}) = \sum_{ij} P_{ij} \ln(1 + \|\tilde{y}_i - \tilde{y}_j\|^2)$ and $B(P, \tilde{Q}) = \ln \sum_{ij} (1 + |\tilde{y}_i - \tilde{y}_j\|^2)^{-1}$ where $q_{ij} = (1 + \|y_i - y_j\|^2)^{-1}$ and $\tilde{Q}_{ij} = \frac{(1+|\tilde{y}_i - \tilde{y}_j\|^2)^{-1}}{\sum_{kl}(1+|\tilde{y}_k - \tilde{y}_l\|^2)^{-1}}$. The corresponding update rule is

$$Y^{new} = (\mathbf{L}_{P \circ q} + \frac{\rho}{4}\mathbf{I})^{-1}(\mathbf{L}_{Q \circ q} + \frac{\rho}{4} Y)$$

where $\mathbf{L}$ again denotes the graph Laplacian and $\circ$ is the elementwise product operator.

In practice, MM is realized by two loops: an outer loop for the update of Y and an inner loop to search for the Lipschitz constant ρ. Different from spectral direction (SD) method where a small ϵ is heuristically added to ensure the positive definiteness of Hessian, MM applies an adaptive process to determine ρ which brings greater robustness and accuracy [YPK15].

5 Implementation

We manage to realize the Weighted t-SNE with majorization-minimization framework (wtsnemm) method in statistic software R. Some works have been done to perform NE method in R such as the `tsne` package developed by van der Maaten. However most of these early works focus on a specific NE variant and are not well improved. Thus we determine to realize the Weighted t-SNE which can provide better visualization and apply majorization-minimization to achieve faster, more accurate and robust output. Also, a package is built to wrap all the functions and source codes in order to strengthen the portability. The details of NE method implementation and package build will be shown in the following parts.

R, a.k.a GNU S, is a free and open source platform and language for statistical computing and visualization. Hence it has active communities and users developing extensions to accomplish a variety of statistical tasks. Besides directly sharing the work by other users from official Comprehensive R Archive Network (CRAN), R allows users to write their own extensions to cope with specific problems. Such flexibility comes from the fact that R is able to communicate with multiple popular programming languages such as Python, Java and Scala. Fortran and C/C++ can be conveniently involved in R codes to achieve significant scalability and efficiency improvements. In this article, our work is done by applying both R language and C++ to acquire both high efficiency and easy manipulation. The R version used in this work is 3.3.3.

5.1 R Integration

Our source codes are mainly transplanted from a MATLAB package, NE, developed by Zhirong Yang [1]. The original package consists of multiple low dimension probability selections and optimization techniques. Most of the functions are written in C/C++ and compiled by MATLAB function `-mex`. To realize our target, two major tasks must be accomplished. First, the C/C++ files must be correctly compiled in R platform. Second, the communication between R and underlying C/C++ level should be well constructed so that the arguments and objects can be correctly passed.

In early versions of R, such compiling and interface adjusting are quite tricky and tedious. One might have to write a complicated `makefile` for correct compiling and make sure that arguments are passed correctly in every C/C++ function. When the amount of C/C++ source files is huge, the developer will get trapped in these boring tasks. Fortunately, a matured development environment has been revealed to liberate people into a very simple workflow that allows developers building a package in several trivial steps.

5.1.1 Development Environment

The R package we build utilizes another powerful R package, `devtools`, and a popular integrated development environment (IDE) of R called Rstudio. In general, we need R and Rstudio installed in the computer and packages `devtools`, `roxygen2`, `Rcpp` and `testthat` at the same time. Package `devtools` is a set of useful tools for R package development, especially it can help to compile the C/C++ source codes by simple manipulation. `roxygen2` will generate documentations automatically by a special type of code comment, and it also determines which C/C++ functions are visible to R code layer. `Rcpp` provides tools for seamless R and C/C++ integration. `testthat` is not always necessary. This package helps the developer to carry out the unit test reducing the sorrow of updating.

With all the above mentioned components installed, to get started, a prototype of package called project should be setup first in Rstudio. In Rstudio's GUI, click the `Project` button on the top right corner and properly set the configuration, then a project e.g named `wtsnemm` is created with a certain structure.

[1] `https://sites.google.com/site/neighborembedding/files/ne.zip`

5.1.2 Package Structure

Once the project is created, Rstudio immediately makes a directory with the same name as the project containing two metadata files, `DESCRIPTION` & `NAMESPACE`, and two subdirectories `/R` & `/man`. These are the basic components of an R package while user can add more directories to realize more complicated goals. By default, the directories and files that Rstudio will recognize are as follows:

- **DESCRIPTION**
  `DESCRIPTION` is a pure text file without extension containing the metadata describing the package. Notice that there are two important entries: `Depends` and `Imports` indicating whether other packages or functions will be loaded when the user installs our package. More details can be seen in official guidance [Tea99]. For our package, it depends on R and imports package `Rcpp`.

- **NAMESPACE**
  `NAMESPACE` is usually automatically created in the compiling and building process with the help from package `roxygen2` as long as the developer correctly marks which functions will be exported and visible to users. This file is critical since it ensures no conflicts will occur between our codes and other packages. No manual writing or change is suggested in case of causing bugs.

- **/man**
  Documentations of the package are located in `/man` folder, saved as `*.Rd` files. If `roxygen2` package is used in the developing process, this directory does not need manual operation. The `roxygen2` package applies special type of comments in the codes to automatically generate the documents.

- **/R**
  This directory contains the codes written in R language, including the functions directly written in R and the wrappers for compiled C/C++ codes.

- **/src**
  Directory `/src` consists of source codes of other programming languages such as C, C++ and Fortran. During the compiling or installation, these codes will be compiled into dynamic-link library files(`wtsnemm.dll` or `wtsnemm.so`). Our package stores multiple C/C++ libraries and functions in this directory, e.g `Csparse` library.

Except the most important files and directories mentioned above, R package also accepts other manually created directories for certain usage. `/data` may contain some data file saved as `*.Rdata`. `/demo` directory includes some R codes to achieve demonstration. All files in `/inst` will be copied to the package's root directory during the installation usually providing news and change logs. For more details, see [Wic14].

The simplest structure of one R package can be just the `DESCRIPTION` file plus `/R` directory containing some `*.R` script files. When facing with the complicated tasks that need high performance based on C/C++, such simple structure is not sufficient. The workflow to combine R with C/C++ is rather simple and is shown below.

5.1.3 Workflow

The workflow to build an R package can be divided to three main steps: process the underlying C/C++ codes, works on the R codes and documentation. The first two parts are usually confusing since the development framework has been changing in recent decade due to the update of R and corresponding tools. Now suppose we want to develop our `wtsnemm` package, the brief workflow is as follows.

Pre-work

1. Install software R, IDE Rstudio and packages `devtools`, `Rcpp`, `roxygen2`.
2. Create a project named `wtsnemm` and enter the development environment.
3. In console, run `devtools::use_rcpp()`. This instruction creates a `/src` directory in the project folder. Simultaneously, two roxygen tags that we have to add to our package will be returned.

```
#' @useDynLib wtsnemm
#' @importFrom Rcpp sourceCpp
NULL
```

With the pre-work done, environment for package construction is established and the next step is to deal with the C/C++ codes. With the powerful tool `Rcpp`, the whole process becomes trivial and highly automated.

C/C++ Level

1. Put all the C/C++ source codes into `/src` directory and determine which functions will be exported.

2. Modify the functions that will be exported. Add statements before the function.

```
#include<Rcpp.h>
using namespace Rcpp;
//[[Rcpp::export]]
```

3. If external header files are included in the function, do the following operation

```
extern "C" {
    #include "*.h"
}
```

4. Alter the arguments C/C++ received and the function returns to the data type that can be passed between R and C/C++ levels.

5. Press `CTRL+SHIFT+B` or click `Build & Reload` in the `Build` pane in Rstudio to compile the source codes.

6. Debug and repeat.

Behind the scenes, Rcpp constructs DLL and make it available to R. Notice that running `devtools::load_all()` will also do the compiling work but it is not recommended. When dealing with C codes, loading and unloading may give a chance to memory corruption. However, if the source codes get change, `devtools::load_all()` with argument `recompile=TRUE` will bring convenience for an instant test.

All the exported functions will automatically get a rough wrapper in R while only a simple wrapper will not satisfy the need in practical work such as error report and argument check. It is better to write an R wrapper manually satisfying more requirements.

R Level

1. Edit the wrappers for exported C/C++ functions.

2. Add the two roxygen tags generated in pre-work into any one of the wrapper. Press `CTRL+SHIFT+D` to generate documents and edit the `NAMESPACE` file by `roxygen2` automatically.

3. Run `devtools::load_all` in console or press `CTRL+SHIFT+L` to load all the R scripts.
4. Inspect the codes in console.
5. Debug and repeat.

Now the main skeletons of our `wtsnemm` have been established. The roxygen tags will automatically configure the `NAMESPACE` file declaring and linking to corresponding components for users. The last important job left is the documentation writing part which also relies on the powerful `roxygen2` package. For more details, please check the book by Hadley Wickham[Wic15] and manual of `roxygen2` [WDE]. Instructions of other extra parts like unit test and demo are also included in the book.

With every part ready, one can run `devtools::check()` to check whether there are fatal errors for package compilation and installation. Once the project passes the check, run `devtools::build()` to build the final bundle of package. The built bundle is operation system specific if the parameter `binary` of `devtools::build()` is `FALSE`. Windows will get a zip file and Linux will get a tar.gz one. If `binary=TRUE`, a binary package will be build where the user need an extra compiling step but such package is not sensitive to operation system.

5.2 Hierarchy

Our built `wtsnemm` package carrying out weighted t-SNE method with majorization-minimization algorithm contains two functions that are `fastkNN` and `wtsnemm` available for users.

Table 1: Accessible Functions

Function	Input	Output
`fastkNN`	Data matrix with each row as an entry	A symmetric sparse matrix intriplet form indicating mutual kNN graph with third column all equal to 0.
`wtsnemm`	The path of a text file storing the sparse matrix obtained from `fastkNN`	Coordinates for the low dimensional mapped points.

To carry out dimension reduction on a high dimensional dataset in matrix form, `fastkNN` will be executed first to obtain the sparse similarity matrix in triplet form.

Such sparse matrix is the simplified result based on k nearest neighbors graph. The `wtsnemm` is called to solve the low dimensional mapped coordinates. The two functions with parameters are interpreted in details below

- `fastkNN(X, k, verbose)`

  - `X`
    The data matrix that will be processed to generate kNN. Each COLUMN as an entry of one data point.
  - `k`
    The number determining how many neighbors will be selected of one data point to construct kNN graph.
  - `verbose`
    A boolean to determine whether extra information will be shown during the kNN construction process.

- `wtsnemm(input, output, attr, initial)`

  - `input`
    The path of the input file storing sparse matrix of input data in triplet form. The absolute path is suggested or R will only search for the file in current working directory.
  - `output`
    The path of output file that the final result of low dimensional embedding coordinates will be written in. Absolute path is still suggested and if the file does not exit, a new file will be created for output storage.
  - `attr`
    A parameter controlling the intensity of attraction among low dimensional points during optimization. Usually between 0.9 and 5, and 1 by default.
  - `initial`
    An initial matrix for low dimensional data points deciding whether the initialization will be carried out. `NULL` by default meaning no external initialization will be done. If an initialization matrix is not given, `wtsnemm`

will automatically initialize the low-D points with very small variances around zero point following identical Gaussian distributions

Figure 10 illustrates the internal hierarchy of package `wtsnemm`. Within functions `fastkNN` and `wtsnemm`, underlying C/C++ functions are called for higher efficiency. Our package highly relies on three C libraries. `vptree.h` is in charge of kNN graph construction. `cs.h` allows efficient sparse matrix computation, manipulation and storage that significantly promote the performance. `wtsne.h` is the library developed by Yang doing the main computation jobs of cost function and gradient. Multiple functions written in C combined with sparse matrix techniques from `cs.h` guarantee the high performance.

Algorithm 3 provides the pseudo codes of `wtsnemm` workflow. the search for Lipschitz constant ρ is in an inner loop done by backtracking method.

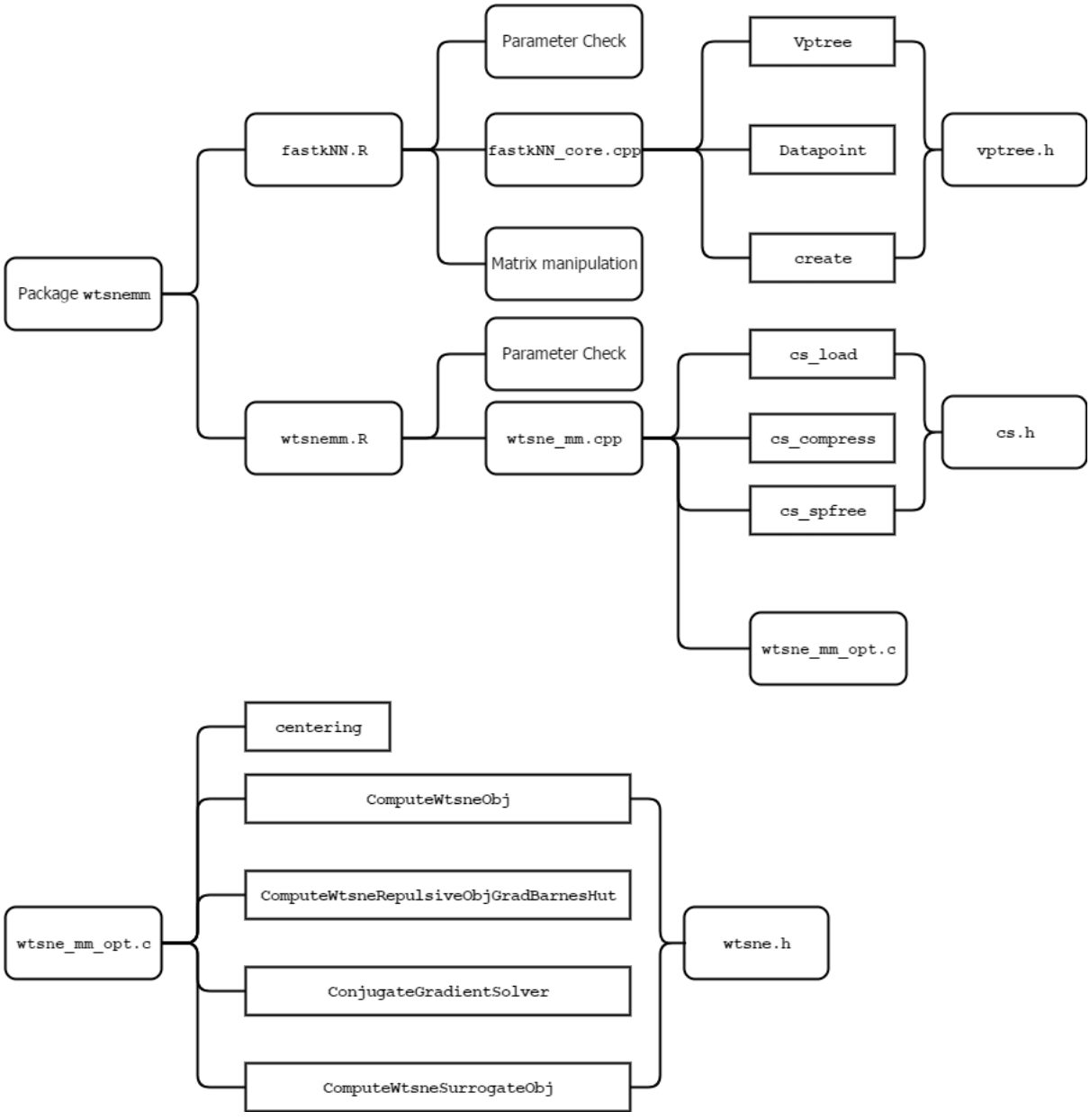
Algorithm 3: `wtsnemm` function workflow

Data: Text file recording sparse matrix
input : Path of data file

```
Load input file and compress;
Initialize low-D coordinates Y;
Call wtsne_mm_opt to optimize target function;
    Copy input matrix;
    Remove diagonal;
    Compute target function constant part;
    repeat
        Compute outer loop of MM;
        Search for ρ by backtracking;
        Refresh Y;
        Compute change of Y;
        if Y change is small enough then
            break ;
    until stopping criterion is met;
Pass Y to wtsne_mm.cpp;
Write Y into output file;
```

output: Text file of low-D coordinates

Figure 10: Function hierarcy



5.3 Interfaces

Rcpp builds up a bridge integrating R and C++ with simple and explicit interfaces. Our package heavily relies on such convenience passing arguments between R and C++ levels. Data types in R and C++ differ quite a lot and must be altered carefully in order to get the codes smoothly compiled and avoid underlying bugs.

In C++, we are familiar with a small set of basic data types including `bool`, `char`, `int`, `float`, `double` and `void`. R has more complicated corresponding data types communicating with C++ defined in Rcpp. These data types can be mainly divided into three sorts:

- Scalar: `int, double, bool, String`
- Vector: `IntegerVector, NumericVector, LogicalVector, CharacterVector`
- Matrix: `IntegerMatrix, NumericMatrix, LogicalMatrix, CharacterMatrix`

Some advanced data types defined in C/C++ library are also applicable as long as the name space is explicitly notified. In our package, both kinds of interfaces are used.

- `fastkNN(X, k, verbose)` return `NumericMatrix` back to R
  - X `matrix` of `double` elements received in R, `NumericMatrix` passed to C++ level.
  - k `double` received in R, `int` to C++ with forced conversion.
  - verbose `logical` received in R, `bool` to C++.
- `wtsnemm(input, output, attr, initial)`
  - `input` `character` in R, `std::string` to C++.
  - `output` `character` in R, `std::string` to C++.
  - `attr` `double` in R, `double` to C++.
  - `initial` `matrix` of `double` elements in R, `NumericMatrix` to C++.

Such mixture of data types may cause bugs hard to detect. It is strongly recommended to follow the Rcpp coding styles. The official manual [Tea99] for R package development also provides thorough instructions.

6 Demonstration

The performance of our `wtsnemm` implementation will be compared with the most conventional and prevalent classic MDS methods in R. Eight publicly accessible datasets are chosen to illustrate the low dimensional visualizations including vectorial form and graph layout. The superiority of `wtsnemm` method will be clearly shown by multiple figures. Section 6.1 introduces the datasets we implemented. The experiment setting will be shown in Section 6.2 and Section 6.3 demonstrates the results of visualization.

6.1 Datasets

Th chosen datasets are: (1) the `Five Phoneme` dataset, (2) the `optdigits` dataset, (3) the `coil-20` dataset, (4) the `pie` dataset, (5) the `MNIST70k` dataset, (6) the `20 News group` dataset , (7) the `coil-100` dataset and (8) the `worldtrade` dataset. Datasets (1)-(7) are datasets in vectorial form and (8) is in a graph layout.

The `Five Phoneme` dataset can be accessed in [JL10] online. It is extracted as a subset from `TIMIT` database, which is commonly used in speech recognition. The dataset contains 4509 data points where each data point has 256 dimensions as a piece of voice record. All the data points belong to five classes indicating five different phonemes.

Dataset `optdigits` consists of 5620 data points where each has 64 dimensions. Every data point is in fact extracted from a bitmap of one hand-written digit. The bitmap is encoded in the 64 dimensions and the vector element is between 0 and 16. Dataset `coil-20` contains 1440 images of 20 different objects expressed in grey scale with 1024 pixels as dimensions. The images are taken rotationally around the object with equal rotation angle from 72 directions. Dataset `pie` is an image dataset expressed in grey scale of human face. `pie` has 1166 face images with a change of illumination conditions and it has a dimension of $32 \times 32 = 1024$.

`MNIST70k` and `20 News group` are two much bigger datasets. `MNIST70k` has 70000 grey scale images of hand-written digits with 784 dimensions. `20 News group` is a dataset of 20000 text documents in 20 classes. `coil-100` is an updated version of `coil-20` dataset with 7200 images of 100 objects having a same number of pixels as `coil-20`. The last dataset `worldtrade` is a weighted undirected graph with nodes indicating 80 different countries and the edges represent the mutual trade amounts.

6.2 Experiment Settings

2-D figures are drawn to illustrate the impressive performance of our `wtsnemm` method. The figures are based on the dimension reduction results of all datasets mentioned above for `wtsnemm`. Datasets `Five Phoneme`, `optdigits` and `coil-20` are chosen to illustrate the results of MDS as comparison.

For MDS, the built-in function `cmdscale` in basic R session will realize the classical multidimensional scaling method. It accepts a distance matrix being a data type `dist` as input and gives out dimension reduction results being the coordinates of data points in low dimensional space. Parameter k controls how many dimensions will be reduced to for the target space. To get the distance matrix required by `cmdscale`, function `dist` should be run to obtain the distance matrix where the dataset is expressed as a matrix with each row indicating a data point. For more details, users can check function manuals by typing `?dist` and `?cmdscale` in the console.

For our `wtsnemm` method, the parameter need to be set is the number of nearest neighbors in function `fastkNN`. This parameter is not constant and influences the performance of dimension reduction. The numbers we chose can be seen in the following table

Table 2: kNN selection

Dataset	5Phonemes	optdigits	coil20	pie	MNIST70k	20NG	coil100
No. of NN	5	6	3	50	10	30	3

Dataset `worldtrade` has already been in a neighborhood graph layout. It does not require extra graph generalization process where only the symmetrization and normalization are required.

Computing such distance matrix for MDS is rather expensive in R. Restricted by the memory and CPU consumption, only three comparably smaller datasets `Five Phonemes`, `opdigits`, and `coil-20` can be computed. Thus the evaluation for MDS and `wtsnemm` by visualizing the low dimensional output will only be carried out on these three datasets. Other datasets will be visualized with just the results of `wtsnemm` method.

All datasets except `worldtrade` experienced 2-stage processing in `wtsnemm` dimension reduction. First, an over attraction initialization is done which means the dataset will receive a pre-processing with fewer iterations and larger attraction.

This stage fast pushes the mapped low dimensional points to rough positions. Approximate low dimensional clusters will form and a refreshed initialization will be exported for next stage. The second stage will be run with moderate attraction parameter until it reaches the iteration limit or converges. In our experiments, over attraction stage was set to be non-initialization and `attr=3` for 30 iteration in `wtsnemm` function. And the second stage took the result of last stage as initialization. The attraction parameter was set to `attr=1` with 300 iterations.

6.3 Demonstration

All the figures shown below are drawn by `ggplot2` package [Wic16] in R. Figures 11, 12 and 13 compare the dimension reduction results by MDS and `wtsnemm` methods. Figure 11 visualizes the result of hand written `optdigits` dataset. MDS gives out an output more or less like a ball where only four classes at four corners are somewhat separated. Other classes accumulate and get overlapped in the narrow central area. No clear gaps can be observed in the 2-D visualization. In contrast, `wtsnemm` produces a quite satisfying result that all classes are mapped rather dense and clear gaps between classes appear indicating the crowding problem has been well avoided.

Figure 12 shows the results of `coil-20` dataset. MDS method does not successfully separate the mapped data points according to their natural clusters. Data points disperse in the area and no significant topologies or gaps can be seen. `wtsnemm` gives an illustration where many circle-like topologies exist denoting the rotationally photo taking process. The gaps among object classes are also clear enough and the tiny overlapping in the central part is induced by four similar car-like objects in original dataset [vdMH08].

For dataset `Five Phonemes`, MDS performs a little better. Classes "dcl" and "sh" have been almost recognized. Though "ly" also forms a cluster, it is too close to "aa" and "ao" causing some crowding. `wtsnemm` significantly outperforms MDS where similar classes are kept close. Dissimilar classes are pushed far apart where gaps shape between two classes. In the visualization by `wtsnemm`, "aa" and "ao" classes are close and mutually infiltrated since "ao" and "aa" are quite similar in pronunciation.

Besides the datasets with clear labels, `wtsnemm` is capable to align points without labels. Figure 14 reveals the result of `pie` dataset where the variable is the illu-

Figure 11: Dimension reduction results of `optdigits`

(a) MDS (b) `wtsnemm`

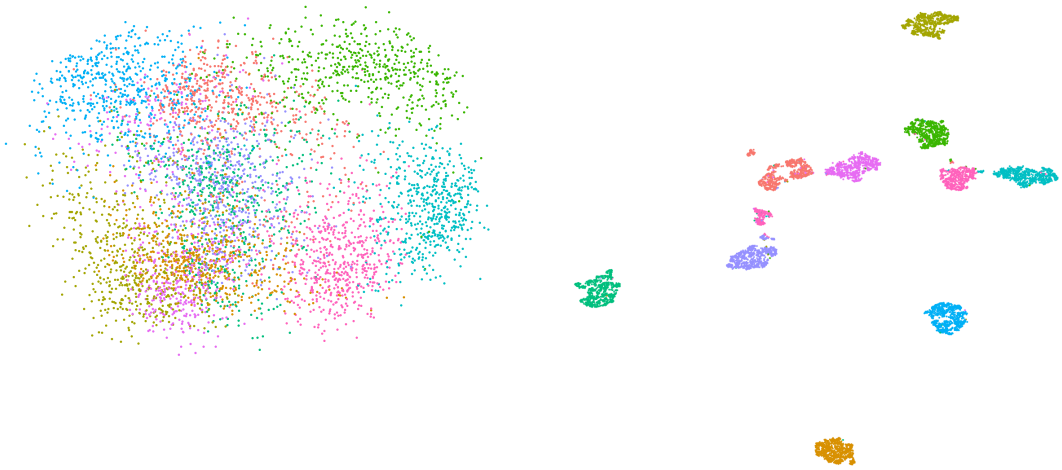


Figure 12: Dimension reduction results of `coil-20`

(a) MDS (b) `wtsnemm`

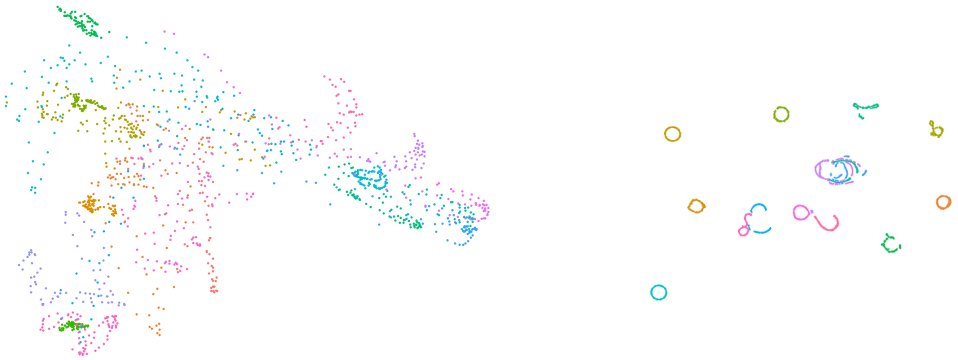
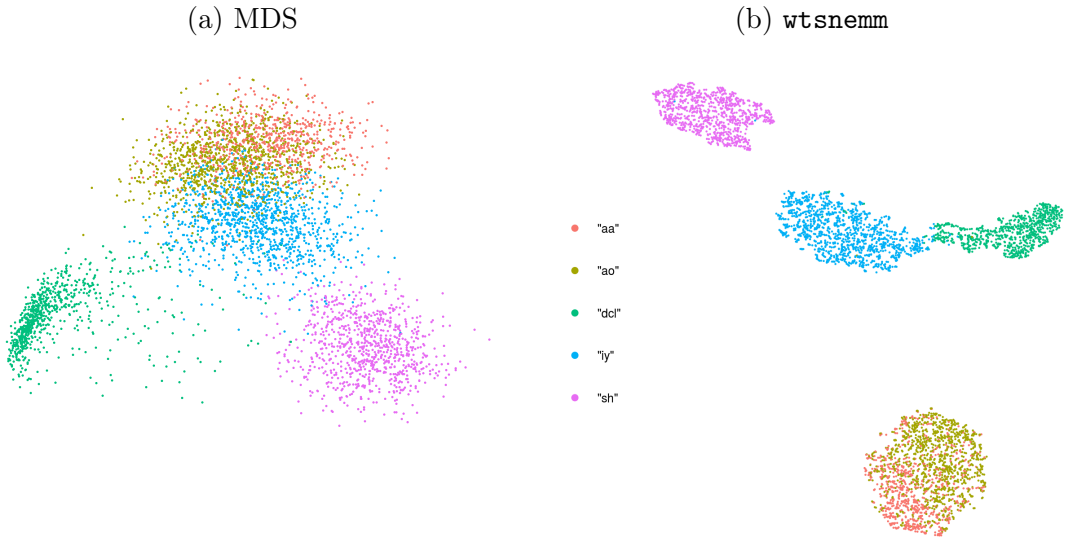


Figure 13: Dimension reduction results of `Five Phonemes`



mination condition. Obvious change of light directions can be found in the plot. Compare the left with right part of the plot, the light direction changes from left to right. From bottom to top, the light is becoming weaker.

For those larger datasets, `wtsnemm` shows great capability to solve complicated dimension reduction tasks. Dataset `coil-100` requires about 6.25 GB to get loaded into memory. `wtsnemm` finishes the dimension reduction process in dozens of minutes with satisfying result. Figure 15 shows the visualization of `coil-100`. Those typical circle structure are well preserved. Figure 16 demonstrates the result of `20 News Group`. Though classes have been preserved, the gaps are not really visible. The result of `MNIST70k` is shown in Figure 17. Ten classes of handwritten digits are well separated without obvious overlapping. Figure 18 shows the dimensional reduction result of `worldtrade`, a dataset in graph layout. We can see that countries of different continents are properly clustered.

The PR curves evaluating the performances of MDS and `wtsnemm` are presented in Figure 19. Table 3 contains the corresponding AUC values. In all three datasets, `wtsnemm` significantly outperforms MDS. In `optdigits` dataset, `wtsnemm` achieves a result very close to the perfect point (1,1).

Table 3: AUC values

Figure 14: Dimension reduction result of `PIE` dataset

Figure 15: Dimension reduction result of `coil-100`

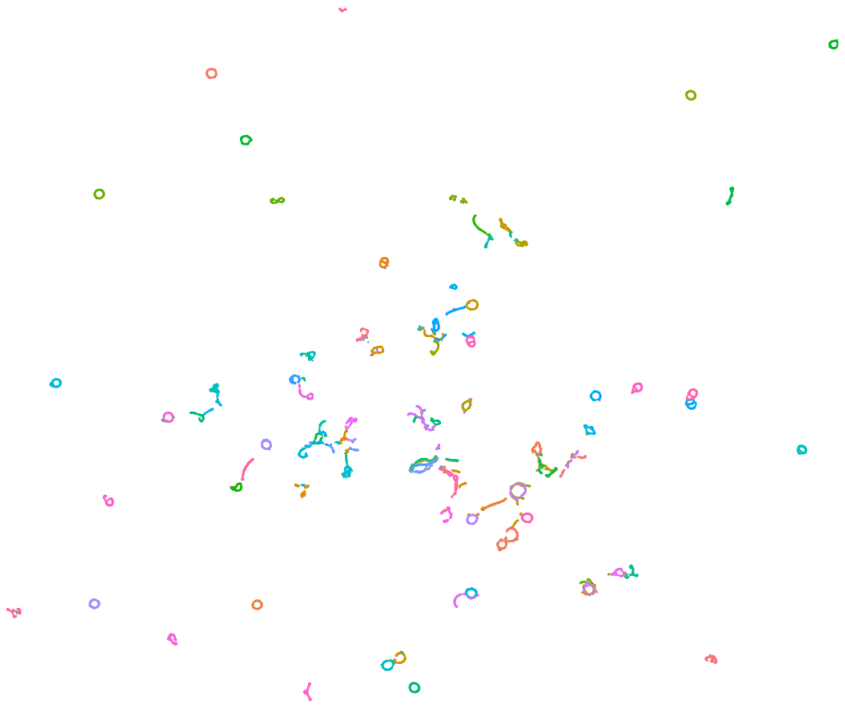


Figure 16: Dimension reduction result of `NG20`

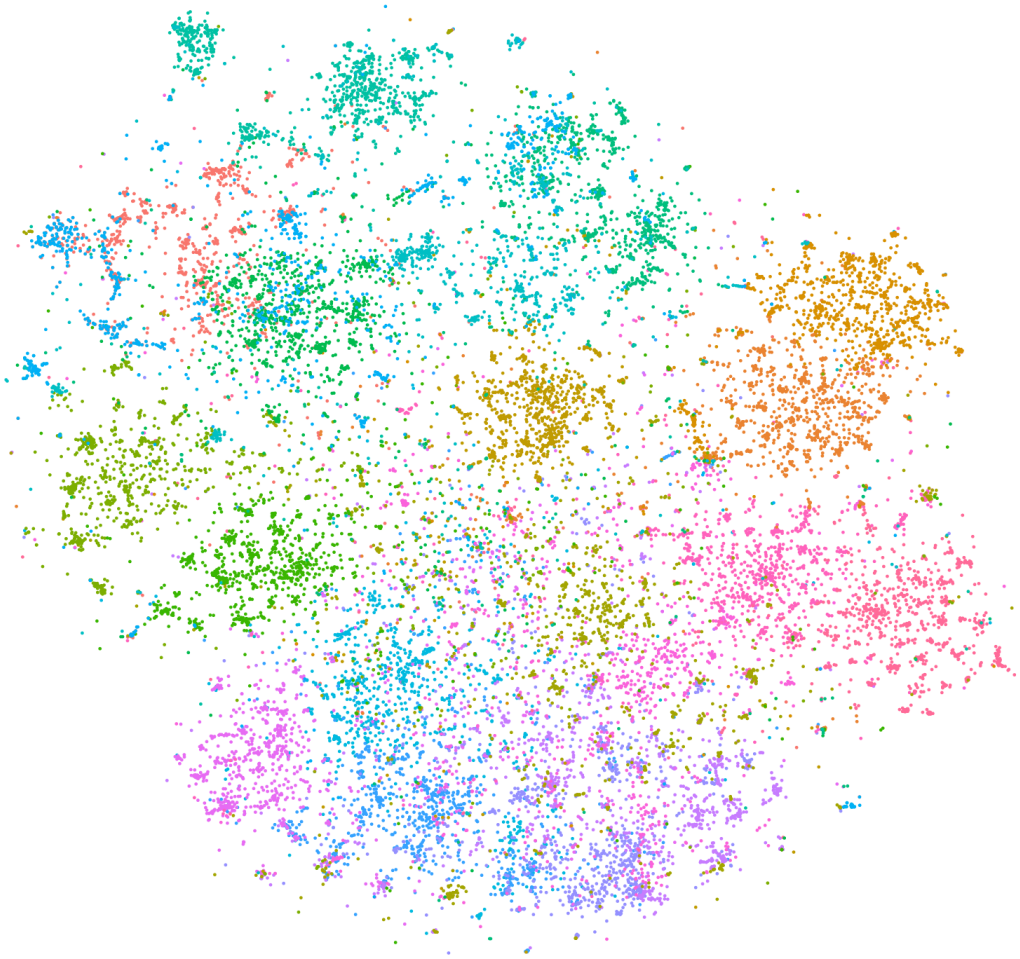


Figure 17: Dimension reduction result of `MNIST70k`

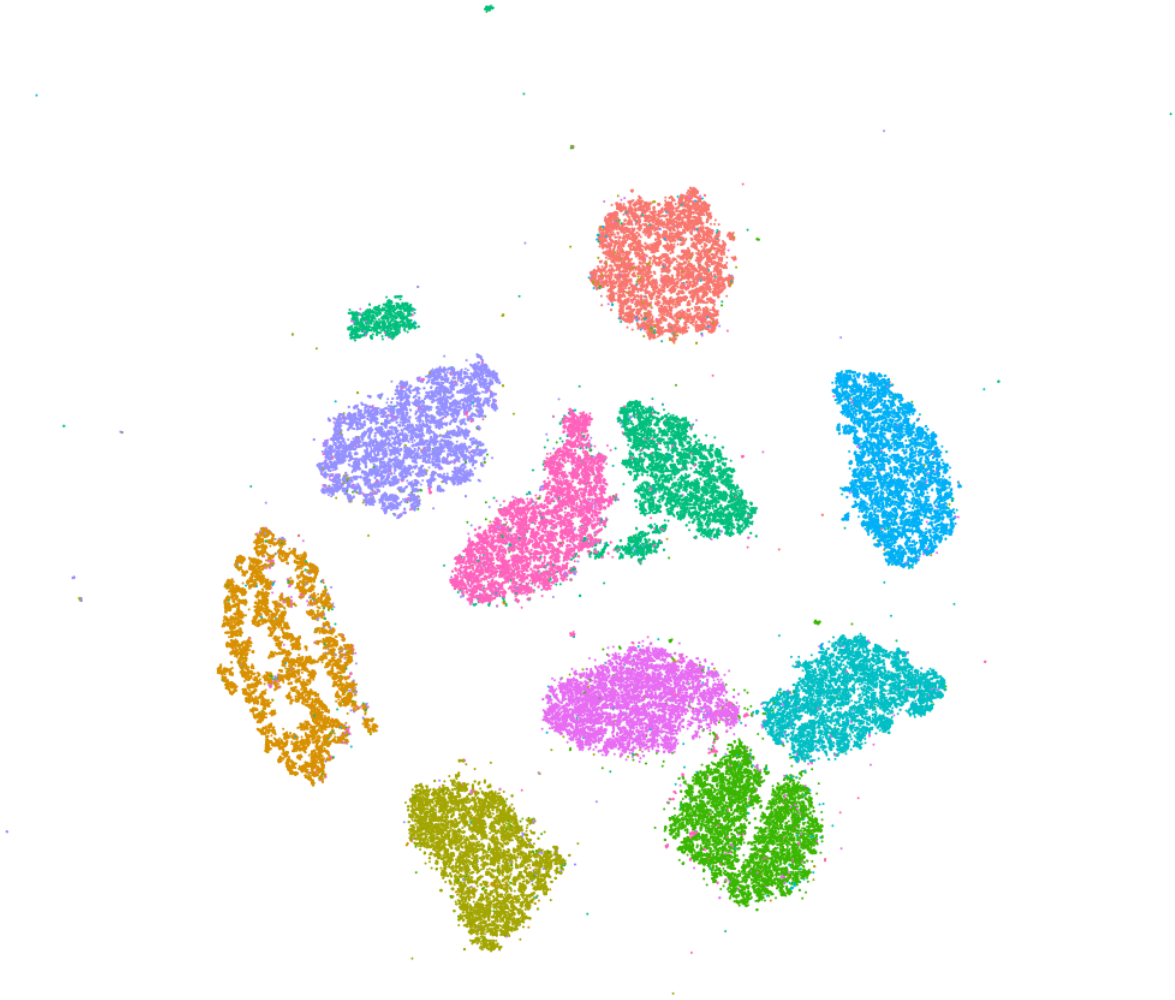


Figure 18: Visualization of graph layout dataset `worldtrade`

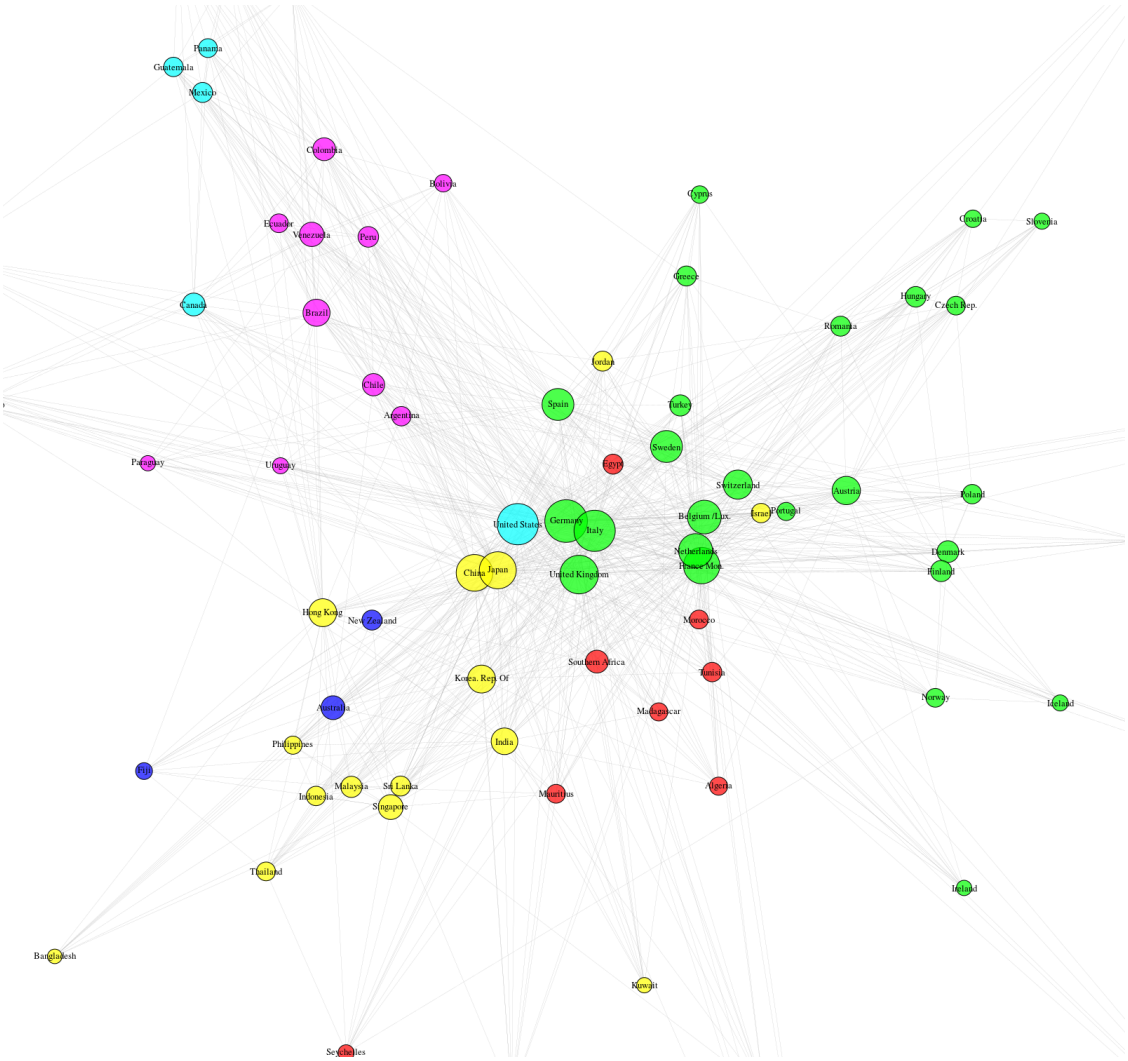
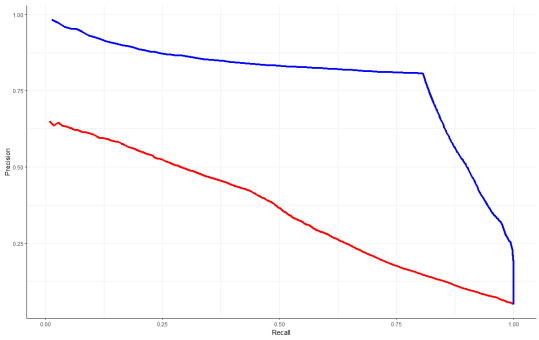
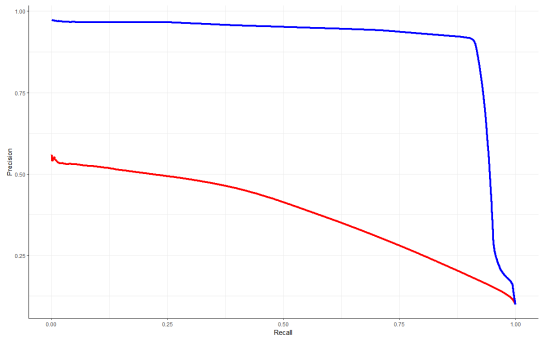


Figure 19: PR curves of `coil-20, optdigits and 5 phonemes`

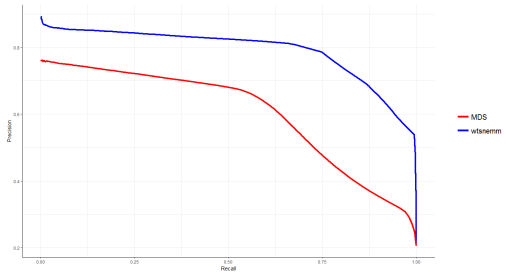
(a) `coil-20`



(b) `optidigits`



(c) `5 Phonemes`



AUC	`coil-20`	`optdigits`	`5 Phonemes`
wtsnemm	0.7929	0.8973	0.7984
MDS	0.3813	0.3756	0.608

7 Summary and Future Work

In this thesis, we have introduced the theoretical interpretation and statistic software implementation of neighbor embedding (NE) method. The conventional MDS method is reviewed together with original SNE framework. We have presented skills of construction and simplification of probabilistic models have been introduced with details of affinity determination, kNN graph and low dimensional probability selection. Cost function related parts cover the divergence transformation, approximation imitating N-body problem and optimization techniques.

For the algorithm realization in statistical software, we have provided a general introduction to R package structure. Then we have described the common workflow of developing an R package utilizing both R and C/C++ level codes by explaining the interfaces between R and C/C++ levels. Finally, we have realized the weighted

t-distributed neighbor embedding method by majorization-minimization technique in R and compared the results with conventional classic MDS. Results by visualizing the output of both MDS and `wtsnemm` have been strongly proved that our `wtsnemm` method is faster, more convenient and robust than the conventional MDS. The two main drawbacks of original NE, computational complexity and crowding problem, are better solved by `wtsnemm`.

There still remains space for improvements. For the R package, more options of optimization techniques could be considered such as BFGS and DFP. Developing parallel computing variant of NE would also be attractive and the deployment in R could utilize R's powerful parallel computing capability. Another topic would be a parametric form of NE method in order to deal with the out-of-sample data point which might make sense for broader application. Further consideration on improving the metric learning, for example by learning similarity directly, instead of learning it from distances, or how to expand the application of NE beyond plain visualization (d>3) would be worth exploring in detail.

References

Ben75 Bentley, J. L., Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18,9(1975), pages 509–517.

BH86 Barnes, J. and Hut, P., A hierarchical o (n log n) force-calculation algorithm. *nature*, 324,6096(1986), pages 446–449.

BN01 Belkin, M. and Niyogi, P., Laplacian eigenmaps and spectral techniques for embedding and clustering. *NIPS*, volume 14, 2001, pages 585–591.

BSL+08 Buja, A., Swayne, D. F., Littman, M. L., Dean, N., Hofmann, H. and Chen, L., Data visualization with multidimensional scaling. *Journal of Computational and Graphical Statistics*, 17,2(2008), pages 444–472.

CP Carreira-Perpinán, M. A., The elastic embedding algorithm for dimensionality reduction.

CSMH07 Cook, J., Sutskever, I., Mnih, A. and Hinton, G., Visualizing similarity data with a mixture of maps. *Artificial Intelligence and Statistics*, 2007, pages 67–74.

Dav91 Davidon, W. C., Variable metric method for minimization. *SIAM Journal on Optimization*, 1,1(1991), pages 1–17.

DH97 Demartines, P. and Hérault, J., Curvilinear component analysis: A self-organizing neural network for nonlinear mapping of data sets. *IEEE Transactions on neural networks*, 8,1(1997), pages 148–154.

DLR77 Dempster, A. P., Laird, N. M. and Rubin, D. B., Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38.

DML11 Dong, W., Moses, C. and Li, K., Efficient k-nearest neighbor graph construction for generic similarity measures. *Proceedings of the 20th international conference on World wide web*. ACM, 2011, pages 577–586.

EFA+11 Eddelbuettel, D., François, R., Allaire, J., Chambers, J., Bates, D. and Ushey, K., Rcpp: Seamless r and c++ integration. *Journal of Statistical Software*, 40,8(2011), pages 1–18.

GD13 Granas, A. and Dugundji, J., *Fixed point theory*. Springer Science & Business Media, 2013.

Hot33 Hotelling, H., Analysis of a complex of statistical variables into principal components. *Journal of educational psychology*, 24,6(1933), page 417.

HR02 Hinton, G. and Roweis, S., Stochastic neighbor embedding. *NIPS*, volume 15, 2002, pages 833–840.

JL10 John Lu, Z., The elements of statistical learning: data mining, inference, and prediction. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 173,3(2010), pages 693–694.

KB00 Klock, H. and Buhmann, J. M., Data visualization by multidimensional scaling: a deterministic annealing approach. *Pattern Recognition*, 33,4(2000), pages 651–669.

Koh98 Kohonen, T., The self-organizing map. *Neurocomputing*, 21,1(1998), pages 1–6.

Kru64 Kruskal, J. B., Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29,1(1964), pages 1–27.

LN89 Liu, D. C. and Nocedal, J., On the limited memory bfgs method for large scale optimization. *Mathematical programming*, 45,1(1989), pages 503–528.

LV07 Lee, J. A. and Verleysen, M., *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.

ML14 Muja, M. and Lowe, D. G., Scalable nearest neighbor algorithms for high dimensional data. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36,11(2014), pages 2227–2240.

Noa07 Noack, A., Energy models for graph clustering. *J. Graph Algorithms Appl.*, 11,2(2007), pages 453–480.

Oks10 Oksanen, J., Cluster analysis: tutorial with r. *University of Oulu, Oulu*.

OR00 Ortega, J. M. and Rheinboldt, W. C., *Iterative solution of nonlinear equations in several variables*. SIAM, 2000.

PS12 Preparata, F. P. and Shamos, M., *Computational geometry: an introduction*. Springer Science & Business Media, 2012.

RS00 Roweis, S. T. and Saul, L. K., Nonlinear dimensionality reduction by locally linear embedding. *science*, 290,5500(2000), pages 2323–2326.

Sam69 Sammon, J. W., A nonlinear mapping for data structure analysis. *IEEE Transactions on computers*, 100,5(1969), pages 401–409.

SK70 Shanno, D. F. and Kettler, P. C., Optimal conditioning of quasi-newton methods. *Mathematics of Computation*, 24,111(1970), pages 657–664.

TDSL00 Tenenbaum, J. B., De Silva, V. and Langford, J. C., A global geometric framework for nonlinear dimensionality reduction. *science*, 290,5500(2000), pages 2319–2323.

Tea99 Team, R. C., Writing R extensions, 1999.

TLZM16 Tang, J., Liu, J., Zhang, M. and Mei, Q., Visualization large-scale and high-dimensional data. *arXiv:1602.00370*.

Tor52 Torgerson, W. S., Multidimensional scaling: I. theory and method. *Psychometrika*, 17,4(1952), pages 401–419.

van

VCP12 Vladymyrov, M. and Carreira-Perpinan, M., Partial-hessian strategies for fast learning of nonlinear embeddings. *arXiv:1206.4646*.

VCP13 Vladymyrov, M. and Carreira-Perpinán, M. A., Entropic affinities: Properties and efficient numerical computation. *ICML (3)*, 2013, pages 477–485.

VCP14 Vladymyrov, M. and Carreira-Perpinán, M. A., Linear-time training of nonlinear low-dimensional embeddings. *AISTATS*, 2014, pages 968–977.

vdM09 van der Maaten, L., Learning a parametric embedding by preserving local structure. *International Conference on Artificial Intelligence and Statistics*, 2009, pages 384–391.

vdM14 van der Maaten, L., Accelerating t-sne using tree-based algorithms. *Journal of machine learning research*, 15,1(2014), pages 3221–3245.

vdMH08 van der Maaten, L. and Hinton, G., Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9,Nov(2008), pages 2579–2605.

VK06 Venna, J. and Kaski, S., Local multidimensional scaling. *Neural Networks*, 19,6(2006), pages 889–899.

WDE Wickham, H., Danenberg, P. and Eugster, M., roxygen2: In-source documentation for r, 2015. *URL http://CRAN. R-Project. org/package= roxygen2. R package version*, 5,1.

Wic14 Wickham, H., *Advanced R*. CRC Press, 2014.

Wic15 Wickham, H., *R packages*. O'Reilly Media, Inc., 2015.

Wic16 Wickham, H., *ggplot2: elegant graphics for data analysis*. Springer, 2016.

WS06 Weinberger, K. Q. and Saul, L. K., An introduction to nonlinear dimensionality reduction by maximum variance unfolding. *AAAI*, volume 6, 2006, pages 1683–1686.

Yia93 Yianilos, P. N., Data structures and algorithms for nearest neighbor search in general metric spaces. *SODA*, volume 93, 1993, pages 311–21.

YKXO09 Yang, Z., King, I., Xu, Z. and Oja, E., Heavy-tailed symmetric stochastic neighbor embedding. *Advances in neural information processing systems*, 2009, pages 2169–2177.

YPK13 Yang, Z., Peltonen, J. and Kaski, S., Scalable optimization of neighbor embedding for visualization. *ICML (2)*, 2013, pages 127–135.

YPK14 Yang, Z., Peltonen, J. and Kaski, S., Optimization equivalence of divergences improves neighbor embedding. *ICML*, 2014, pages 460–468.

YPK15 Yang, Z., Peltonen, J. and Kaski, S., Majorization-minimization for manifold embedding. *AISTATS*, 2015.