universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Approximative Linear t-SNE Using k-Means“

verfasst von / submitted by

Sonja Biedermann, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2021 / Vienna 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 921
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Masterstudium Informatik
Betreut von / Supervisor:	Univ.-Prof. Dr. Claudia Plant

Abstract

Dimensionality reduction is the transformation of high-dimensional data into a lower dimensional space while keeping the interesting parts of the data intact. Ideally, the low-dimensional so-called embedding reveals structure that would have been much more difficult to detect in the high-dimensional space.

There are many different approaches to dimensionality reduction, and the most common taxonomy separates them into linear and non-linear methods. Linear methods transform the high-dimensional data solely by applying linear transformations, such as PCA, while non-linear methods are essentially free to transform the data by any means.

In this thesis, we will be focusing on a non-linear method called t-SNE, which is a variant of Stochastic Neighborhood Embedding using the Student-t distribution. t-SNE has become popular due to the visual properties of the embeddings it produces: well-separated clusters consisting of similar items. It is, however, a stochastic method, non-parametric in its original form, possesses a difficult to optimize non-convex objective function and also has a prohibitively high runtime complexity of $\mathcal{O}(n^2)$.

We implement and evaluate a variant of t-SNE named kt-SNE utilizing k-Means and locality sensitive hashing. We achieve an overall runtime complexity of $\mathcal{O}(n)$. We thoroughly evaluate our method and put it to the test against three competing methods. Our method achieves results that are mostly within 10% of the best result of the competing methods on real data and runs significantly faster than Barnes-Hut t-SNE.

Zusammenfassung

Unter Dimensionsreduktion versteht man die Transformation von hochdimensionalen Daten in einen niedrigdimensionalen Raum, ohne die wesentlichen Merkmale der Daten zu verlieren. Idealerweise gibt das niedrigdimensionale sogenannte Embedding seine Struktur leichter Preis als sein hochdimensionales Gegenstück.

Es gibt viele unterschiedliche Ansätze zur Dimensionsreduktion. Die häufigste Unterscheidungsmethode ist zwischen linearer Dimensionsreduktion und nonlinearer Dimensionsreduktion. Lineare Dimensionsreduktionsmethoden transformieren die Daten in einen niedrigdimensionalen Raum nur unter Verwendung von linearen Transformationen, wohingehen nonlineare Methoden die Daten jegliche Art von Transformation anwenden dürfen.

In dieser Thesis beschäftigen wir uns mit einer nonlinearen Methode namens t-SNE, die eine Variante von Stochastic Neighbor Embedding (SNE) unter Verwendung der Student-t Verteilung darstellt. t-SNE ist beliebt aufgrund der visuell ansprechenden Embeddings, die Clusterings ähneln, die es produziert. Allerdings ist sie auch stochastisch, nonparametrisch, verfügt über eine schwierig zu optimierende Zielfunktion und hat zudem noch eine hohe Laufzeitkomplexität von $\mathcal{O}(n^2)$.

Wir implementieren und evaluieren einer variante von t-SNE namens kt-SNE, die den Clusteringalgorithmus k-Means und Locality Sensitive Hashing verwendet. Wir erreichen eine Laufzeitkomplexität von $\mathcal{O}(n)$. Wir evaluieren unsere Methode gründlich und vergleichen sie mit drei vergleichbaren Methoden. Unsere Methode erreicht Resultate die zumeist innerhalb von 10% der besten Methode auf realen Daten liegen und läuft signifikant schneller als Barnes-Hut t-SNE.

Contents

1 Introduction

Dimensionality reduction is an important tool of data analysis. High dimensional data sets may encompass so many dimensions that it becomes difficult to understand let alone interpret the data. A fundamental problem in data analysis is the curse of dimensionality—that is, that many methods struggle to analyze high-dimensional data because high-dimensional data tends to become so sparse that the distance of any point to its nearest neighbor converges to the distance to its farthest neighbor.

Dimensionality reduction can help by reducing the dimensions such that the data can be drawn or can be used as a means of feature extraction so that other methods, e.g. classification algorithms may produce better results. However, the important caveat here is that dimensionality reduction is just as affected by the curse of dimensionality as any other method. Nevertheless, some dimensionality reduction algorithms are able to extract interesting structure from high-dimensional data and are especially useful for visualization as a means of data exploration.

Dimensionality reduction has been studied for more than a hundred years with PCA [49] being the most prominent and also one of the oldest methods. PCA belongs to the class of *linear* dimensionality reduction methods, sometimes also referred to as *projective* dimensionality reduction. Linear dimensionality reduction (LDR) finds a projection P to transform the data into a lower dimensional space. This method is only effective if whatever property should be preserved can be represented linearly.

Nonlinear dimensionality reduction (NLDR) is used when this is not the case. There exist multiple approaches to NLDR, e.g. kernel PCA which is a projective method but in a feature space possibly resulting from a nonlinear mapping, but the method we will be focusing on is *manifold learning*. The idea of manifold learning is that the high-dimensional data sets is artificial and most of the data actually lies on a much lower dimensional manifold plus some noise. Most manifold learning methods do not actually learn the manifold itself but try to approximate the geodesic distance across the manifold, often by computing a k-nearest neighbor graph. As such, a surprising amount of manifold learning methods are actually more or less graph embedding algorithms after the initial step of computing the graph which differs between methods. Of course, many graph embedding methods are not designed to produce "good-looking" low-dimensional embeddings and are instead used as means of feature extraction, and deal with different features and problems inherent to graphs.

A particularly prominent nonlinear dimensionality reduction method is *t*-SNE [66]. *t*-SNE has found widespread usage in many scientific communities dealing with high-dimensional data, such as gene-expression data. *t*-SNE has the property that similar points appear to be clustered in its output, which is visually pleasing and thus readily accepted as a good embedding, even if limitations apply [33, 68]. As such, it is also considered to be a visualization technique. However popular, *t*-SNE comes with a significant downside: its high computational complexity—of quadratic order—make it unusable for particularly large data sets.

1.1 Motivation

The high computational cost of *t*-SNE is the main motivation of this thesis. There have been multiple attempts at speeding it up [65, 42, 45, 60] and in this thesis, we introduce another way of doing so and compare ourselves against the other methods.

t-SNE can be divided into two phases. The initial phase consists of computing probabilities in the high-dimensional pairs. This involves finding all, or at least *some*, distances to all other points or a selected few nearest neighbors. The approach to optimize this phase is to use only *approximate* nearest neighbors. This can be done by e.g. using trees, as done in Barnes-Hut *t*-SNE [65], or in FI*t*-SNE [42], or by any other approximate nearest neighbor finding scheme. We chose to use locality sensitive hashing, a well established method used in many fields such as near duplicate detection (e.g. plagiarism detection), clustering, audio similarity detection, and nearest neighbor search.

The second phase consists of gradient descent. In every iteration, an n-body simulation is computed—the standard variant of *t*-SNE considers all forces between all points in the data set. A simple method of speeding this up is to coarsen the data set such that only interactions between much fewer than n entities need to be computed. In Barnes-Hut *t*-SNE, this is again done with the help of trees—each node in the tree that fulfills a quality requirement (specified by the user) is summarized. We propose a simpler method of running only a few iterations of a clustering algorithm to find a rough segmentation of points, where every segment can be summarized. This method of approximation can be expected to be less precise, but also faster.

Our claim is that this rough segmentation is enough to produce an embedding in which enough of the characteristics of the data set are represented to make deductions, which is useful for experimental data analysis.

1.2 Contribution

We propose an accelerated approximative variant of t-SNE named kt-SNE. The algorithm utilizes locality sensitive hashing and k-Means to achieve an overall linear runtime.

Our algorithm achieves embeddings that are within 10% of the nearest neighborhood purity achieved by the other methods on most real data. We are faster than Barnes-Hut t-SNE, but fall significantly behind FIt-SNE and are roughly as fast as UMAP. The results of our algorithm tend to be distorted, but the underlying structure found by the other methods is also present.

1.3 Structure of Thesis

In Chapter 2 we review the basic definitions and notations used in this thesis, as well as introduce locality sensitive hashing, k-Means and stochastic neighborhood embedding (SNE). Our work builds upon these methods and thus an basic understanding is required.

In Chapter 3 we survey various important methods of dimensionality reduction. We review both linear and nonlinear methods, with a focus on the latter. Next, we discuss our proposed algorithm and explain it in relation to t-SNE, on which it is based, in Chapter 4. Our method is evaluated in Chapter 5. We tune the parameters of our method and compare it against other selected methods on real and synthetic data sets. We discuss our findings and conclude this thesis in chapter 6.

2 Preliminaries

In this chapter we introduce the core concepts of this thesis. In particular, we introduce Locality-Sensitive Hashing, k-Means and Stochastic Neighborhood Embedding, as well as some basic definitions and notations.

2.1 Definitions and Notation

Dimensionality reduction maps high-dimensional data points $\{x_i\}_{i=1}^n \in \mathbb{R}^d$ to low-dimensional points $\{y_i\}_{i=1}^n \in \mathbb{R}^r$ where $r \ll d$ such that the points Y “represent” the points X. The low-dimensional representation is called the embedding. How faithfully the embedding represents the core properties of the data is the property to optimize. What constitutes a faithful embedding is the fundamental question of dimensionality reduction and there are multiple answers.

Linear dimensionality reduction methods often deal with the variance. The supposition is that variance constitutes important structure and thus, a faithful embedding should preserve the variance. Other methods assume that the data actually lies on a lower dimensional manifold which is locally linear but possibly globally nonlinear. The intrinsic dimensionality of the manifold is the dimensionality of the space into which the data should be embedded. Note that the structure of the manifold and its dimensionality is fundamentally unknown and must be supplied as an input parameter—i.e. the problem of dimensionality reduction is ill-posed.

For linear dimensionality reduction we write the matrices as $X^{d \times n}$, i.e. the observations are in the columns. This makes some notation easier and is common practice in this field, although it may confuse computer scientists that are more used to having the observations as rows. For almost all methods a centering is required, so we simply assume that X is already centered, i.e. its mean is zero.

A common element in the problem definitions is the covariance matrix $XX^T \in \mathbb{R}^{d \times d}$ which is often manipulated (e.g. decomposed) to maximize the overall variance or minimize the covariance of the embedding. The variances of each feature is written in the main diagonal, while other fields contain the covariances. The trace of a matrix $\operatorname{tr}(\cdot)$ is the sum of the main diagonal and is also the sum of the eigenvalues. This already implies the connection between maximizing variance and choosing the largest eigenvalues—as is done in PCA.

We always assume that the distance at hand is the Euclidean distance, $d(p,q) = ||p-q||_2 = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$. Most methods use the squared Euclidean distance.

The Frobenius norm of a matrix is written as $||\cdot||_F$ and is defined as $||A||_F = \sqrt{\sum_i \sum_j a_{ij}^2}$ for $A \in \mathbb{R}^{m \times n}$.

The eigendecomposition of a orthonormal matrix A results into two matrices Q and Λ such that $A = Q\Lambda Q^T$. This is only applicable for diagonalizable matrices. The generalization to all matrices is the singular value decomposition $A = U\Sigma V^T$.

A positive semidefinite matrix $M \in \mathbb{R}^{n \times n}$ is positive semidefinite if the scalar $z^T M z$ is positive or zero for all non-zero $z \in \mathbb{R}^n$, a constraint implying semi-definiteness is written as $M \succeq 0$.

2.2 Locality-Sensitive Hashing

Locality-sensitive hashing (LSH) is a family of hashing methods designed to find similar items by hashing them to the same address. In a way, these hashing methods differ from all other common uses of hashing, which try to avoid colliding hash indices at all cost.

The need for LSH arises simply from the quadratic growth of the naïve algorithm for finding similar pairs, which involves examining all possible $\mathcal{O}(n^2)$ pairs. If an approximative solution is acceptable, this can be sped up significantly by partitioning the space into segments by hashing the items. These segments are then the hash buckets, and the hash function transform the vectors into some subspace from which the bucket membership can be derived. The contents of a bucket can be assumed to be sufficiently similar to each other with some probability larger than p_1, while sufficiently dissimilar items hash to different buckets is smaller than p_2. Formalizing this, we get the definition of a locality-sensitive family of hash function **F** and its four defining parameters d_1, d_2, p_1 and p_2:

1. If $d(x,y) \leq d_1$, then the probability of collision is at least p_1.
2. If $d(x,y) \geq d_2$, then the probability of collision is at most p_2.

Such a family of locality-sensitive hash functions **F** is then called (d_1, d_2, p_1, p_2)-sensitive [51]. Intuitively, this simply formalizes the notion that the collision probability for similar points is higher than for dissimilar points.

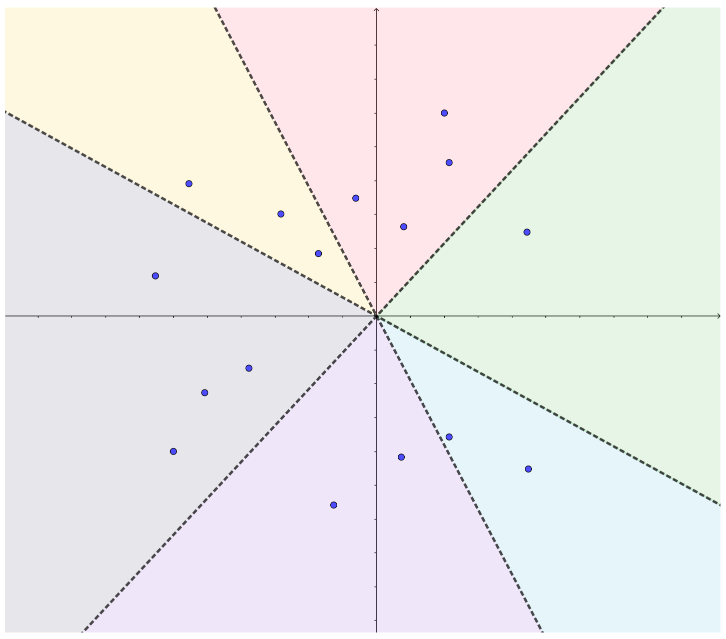


Figure 2.1: Space segmented into 6 buckets by 3 random hyperplanes (SRP)

Various locality-sensitive families of hash functions can be defined for distances such as the Euclidean or Cosine distance. LSH is widely used for data consisting of documents, with applications such a plagiarism detection, finding (approximative) nearest neighbors or many other tasks hinging on similarity detection [51], however it can be used for any kind of data for which a locality-sensitive family of hash functions can be defined.

A simple example of a family of hash function are signed random projections (SRP) in hyperplane LSH. Each hash function is defined by a random vector w defining a hyperplane going through the origin which divides the space into two sub spaces. The hash function is thus

$$h_w(x) = \text{sign}(w^T x). \tag{2.1}$$

Figure 2.1 shows a concrete example. In this case, the colored regions each correspond to a hash bucket. A total of 3 hash functions are used, each corresponding to one of the thick dashed lines delimiting the colored regions. Multiple hash tables, each with their own three hash functions, would be used and the union of all hash buckets that match a query object would form the candidate set, whose contents would be ranked by the distance to the query object. A similar example consists of randomly projecting a point onto a line which is segmented into buckets. More exotic variants of LSH with different means of partitioning the vector space also

Algorithm 2.3.1 *k*-Means

```
procedure $k$-MEANS($X$, $k$)
    centroids ← initial centroids $c_1, c_2, \dots c_k$
    while stopping criterion has not been met do
        for all points $p_i$ in $X$ do
            $c_i$ ← closest centroid to $p_i$
            assign $p_i$ to $c_j$
        end for
        for all centroids $c_i$ do
            $c_i$ ← center of mass of all points assigned to $c_i$
        end for
    end while
end procedure
```

exist, such as clustering LSH [48], but it should be noted that finding the collision probabilities becomes significantly difficult [18] even for the previous example of a segmented line.

A notable family of locality sensitive hash functions which was used in our work is Cross-Polytope LSH.

An important modification to standard LSH, beside the multitude of different families of hash functions, is multi-probe LSH [44]. Multi-probe LSH reduces the overhead due to the need of many hash tables—many in this case refers to hundreds for high-dimensional data—to increase query result quality by probing multiple buckets where similar items could have been misplaced. This increases the size of the candidate pool without allocating additional hash tables and is thus a simple but effective optimization that has been widely adopted.

2.3 k-Means

k-Means is a simple and popular clustering method for data with a known amount of clusters (k). It works by assigning points to the nearest cluster-mean (centroid), updating the cluster mean after all points have been assigned, and repeating this process until a stopping criterion is met, this could either be the convergence of the sum of squared distances to the centroids—equivalent to the centroids no longer changing positions (significantly)—or something simpler such as as fixed amount of iterations. Algorithm 2.3.1 briefly outlines the process.

k-Means is linear in the number of input points, however reaching convergence may take many iterations. *k*-Means also has a strong assumption of Gaussian-shaped clusters and is unable to find clusters of other shapes. Finally, finding a suitable k is not straightforward and is usually done by rerunning the algorithm and finding an optimum sum of squared distances to closest centroid.

2.4 Stochastic Neighborhood Embedding

Stochastic Neighborhood Embedding (SNE) [29] is a probabilistic embedding method and the direct predecessor of t-SNE. SNE proceeds by computing the high-dimensional asymmetric probabilities that some point i will pick another point j

$$p_{ij} = \frac{\exp(-d_{ij}^2)}{\sum_{k \neq i} \exp(-d_{ik}^2)} \tag{2.2}$$

where d_{ij}^2 are some kind of dissimilarity, possibly the scaled squared Euclidean distance given by

$$d_{ij}^2 = \frac{||x_i - x_j||^2}{2\sigma_i^2}. \tag{2.3}$$

Here, the value σ_i is either found by binary search such that the entropy of the distribution $p_{i\cdot}$ is equal to $\log k$, where k is the effective number of neighbors, most often called perplexity.

Another probability is defined in a similar way for the low-dimensional points, defined by

$$q_{ij} = \frac{\exp(-||y_i - y_j||^2)}{\sum_{k \neq i} \exp(-||y_i - y_j||^2)}. \tag{2.4}$$

The optimization target is then to minimize the Kullback-Leibler divergence between these two distributions.

Unfortunately, this method has quadratic runtime and is thus not viable for large data sets. Other downsides have been found and summarized by the subsequent t-SNE paper [66] and include the difficult to optimize objective and the so-called crowding problem, of which the latter is amended by using a different probability distribution in the low-dimensional space (namely the Student-t distribution instead of the Gaussian distribution). t-SNE is discussed in the Related Work section of this thesis.

3 Related Work

In this chapter, we survey a few key dimensionality reduction methods, including t-SNE, which our work modifies, and other algorithms that compete with our proposed algorithm. We include both linear methods and nonlinear methods, though we pay special attention to nonlinear methods.

3.1 Linear Methods

Linear dimensionality reduction deals with finding a new basis into which the input data can be transformed with minimal loss of important structural information. What is deemed to be “important” depends on the method—for some, the new basis should be such that the variance of the projected data is maximized, or alternatively, if information about labels is available, the important information must be what separates the classes.

Linear dimensionality assumes that the underlying structure of the data is rather simple, otherwise it would not be able to be represented as linear combinations of the input data. This assumption implies that the variance of the error (or the noise) is lower than the variance of the observed phenomenon. Although this assumption is often correct and has the added bonus of interpretability, on some data sets it is not sufficient to uncover all or even any of the hidden structure. This raises the need for nonlinear dimensionality reduction which is covered in later subsections.

Dimensionality reduction is usually framed in terms of optimization problems. An objective is picked which quantifies either the presence of the desired structure in the projected space (which should be maximized) or the deviation from the desired state (e.g. in form of a reconstruction error, or some other form of loss), in which case it should be minimized.

All methods covered have the added constraint that the resulting projection is a basis, i.e. all basis vectors are orthogonal. This actually significantly simplifies the optimization problem by limiting the search space along orthogonal directions and allows using certain linear algebra solvers (most notably, decomposition techniques). There exist methods that do not constrain themselves to an orthogonal space. For a more extensive survey on linear dimensionality reduction see [16].

3.1.1 Principal Component Analysis

Principal Component Analysis (PCA) [49] is one of the oldest methods of linear dimensionality reduction and remains the one with the most widespread usage. In its original formulation, PCA minimized the squared residuals between the original data and the projected data. The optimization problem is as follows:

$$\begin{aligned} \text{minimize} \quad & ||X - MM^T X||_F^2 && (3.1) \\ \text{s.t.} \quad & M \in \mathcal{O}^{d \times r}, && (3.2) \end{aligned}$$

More modern works (e.g. [10]) detailing PCA favor a different version of PCA whose objective function is $-\text{tr}(M^T - XX^T M)$. The solution is equivalent to that of Equation 3.2 but results from a different and more profound interpretation that PCA not only minimizes residuals, but actually projects the data down into a lower dimensional space while preserving as much variability in the data as possible. This is done by finding new linear combinations of the features which are uncorrelated (i.e. orthogonal) and maximize the variance, which boils down to an eigenproblem with the familiar solution through diagonalization.

The decomposition of XX^T into $Q\Lambda Q^T$ results in the optimization problem in Equation 3.2 having an optimum when $M = Q_r$, where Q_r contains the r eigenvectors in its columns which correspond to the r largest eigenvalues stored in the diagonal of Λ. For a comprehensive review of the linear algebra involved we recommend [59].

Known weaknesses of PCA are that its runtime scales with the dimensionality of the high-dimensional space, resulting in an overall computational complexity of $\mathcal{O}(d^3)$. The objective function in Equation 3.2 also results in the optimization focusing mainly on preserving large distances in the lower dimensional space due to them having a larger impact on the objective, which is a crucial flaw as the small distances are usually much more important [67].

As PCA fundamentally deals with variance, it also succumbs to severe outliers in the data due to the L_2 norm further exaggerating large distances. To deal with this, multiple instances of robust PCA [40, 25, 26, 20, 15, 12, 16] have been proposed, mostly by shifting from the L_2 norm—the traditional PCA—to the L_1 norm. However, this entails a much higher computational need: an exact solution to L_1-PCA has an exponential runtime, and approximation algorithms proposed so far seem to struggle with finding acceptable solutions when the number of features p is high [15].

The solution of PCA has also been made sparse by adding a LASSO penalty to the objective, resulting in the objective $||X - MM^TX||_F^2 + \lambda||M||_1$. This cannot be solved by the eigenvalue approach detailed above, which led to new algorithms [16].

Another important variation of PCA is probabilistic PCA [62] (PPCA). The fundamental idea is that PCA can be reformulated as the solution maximizing the likelihood of a probabilistic latent variable model, which can be efficiently solved using an EM algorithm. Besides being efficient, probabilistic PCA can also handle missing values and can also be used generatively to provide new samples from the distribution [10]. Note that the manifold found by PPCA is arbitrary, i.e. $M \in \mathbb{R}^{r\times n}$. A closed-form maximum likelihood solution also exists [62], but the EM algorithm is usually preferred especially if $d \gg r$ [16].

3.1.2 Multidimensional Scaling

Multidimensional Scaling (MDS) [63] is a an entire class of linear dimensionality reduction methods following a slightly different idea. Instead of minimizing the residuals of the projection, we compute pairwise similarities (or dissimilarities) of the data and compute a mapping into a space such that these similarities are approximated. The classical MDS (sometimes shortened to CMDS, although this is sometimes misleading, or simply classical scaling [67]) optimization program, however, is simpler, and instead maximizes the scatter in the projected space [16] in the belief that such an projection is the most informative one:

$$\begin{aligned} \text{maximize} \quad & \sum_i \sum_j ||M^T x_i - M^T x_j||^2 \\ \text{s.t.} \quad & M \in \mathcal{O}^{d\times r} \end{aligned} \tag{3.3}$$

It is easy to see that this maximization of scatter and the maximization of variance in PCA leads to the same solution [14, 11]. However, classical scaling scales with the number of points n which may make it preferable in cases where $n < d$ [67]. Other downsides of PCA apply to classical scaling due to equivalence.

Modern MDS is almost always regarded in its much more general definition over pairwise dissimilarities and minimizing the corresponding objective $\sum_i \sum_j (d_X(x_i, x_j) - d_Y(y_i - y_j))^2$ is called the Kruskal-Shephard scaling [16, 36].

As MDS works only on the pairwise dissimilarities, the actual underlying data need not be known, which is an useful feature. The aforementioned objective also does not impose any constraints on the mapping to the low-dimensional space—although we could restrict M to be orthogonal and define the mapping to be $Y = M^T X$ [11, 14], the mapping could also be some nonlinear operation [16]. Sammon's mapping [55] is an example of a nonlinear variant of which eases the disadvantage of not preserving small pairwise distances in the embedding.

3.1.3 Linear Discriminant Analysis

A natural approach to dimensionality reduction when dealing with labeled data is Fisher's linear discriminant analysis (LDA). LDA strives to project the data into a space in which points belonging c different classes are separable by a linear discriminant [10]. This is an example of *supervised* dimensionality reduction, which is also a linear classification model.

LDA proceeds by partitioning the covariance matrix XX^T into the covariance within the classes Σ_W and the covariance between the classes Σ_B, such that $XX^T = \Sigma_W + \Sigma_B$. Computing these is done by computing the variance in relation to the class means μ_c and the overall mean μ, i.e.

$$\Sigma_W = \sum_{i=1}^{n} (x_i - \mu_{c_i})(x_i - \mu_{c_i})^T \tag{3.4}$$

and

$$\Sigma_B = \sum_{i=1}^{n} (\mu_{c_i} - \mu)(\mu_{c_i} - \mu)^T. \tag{3.5}$$

The optimum projection minimizes the within-class variance Σ_W and maximizes the between-class variance, resulting in compressing points of the same classes and creating a separation between classes. The optimization problem is thus

$$\begin{aligned} \text{maximize} \quad & \frac{\operatorname{tr}(M^T \Sigma_B M)}{\operatorname{tr}(M^T \Sigma_W M)} \\ \text{s.t.} \quad & M \in \mathcal{O}^{d \times r} \end{aligned} \tag{3.6}$$

This can be solved using the usual approach of selecting the top r eigenvectors iff $r = 1$. A common misconception discussed in [16] and [24] is that the top r eigenvectors are not necessarily orthogonal, so selecting the top r eigenvectors correspond to maximizing $\operatorname{tr}\left((M^T\Sigma_W M)^{-1}(M^T\Sigma_B M)\right)$ over $M \in \mathbb{R}^{d\times r}$. If a projection of the data is desired, solving the problem of Equation 3.7 over orthogonal M is a better choice and outperforms the eigensolver as shown in [16].

An interesting feature of LDA is that as Σ_B can also be written as $\Sigma_B = \sum_{c=1}^{K} n_c(\mu_c - \mu)(\mu_c - \mu)^T)$, i.e. Σ_B consists of a sum of K matrices of rank 1, as they are an outer product of two vectors, the rank of Σ_B can at most be $K-1$. Thus, LDA can extract at most $K-1$ orthogonal linear combinations to be used as low dimensional features [24].

3.2 Nonlinear Methods

While linear methods are constrained to a eponymous linear projection of the data, i.e. the new embedding space must be a linear combination of the origin space, nonlinear methods are free to map the data in whatever way. If the interesting dynamics present in the data result from nonlinear mechanisms, linear methods will likely fail to recover a meaningful embedding—in the worst case, the embedding may appear interesting but in a misleading way.

The belief that the high-dimensionality of most data sets is partially artificial gives rise to the method of manifold learning. The idea is that the bulk of the data actually lies on a lower dimensional manifold which is embedded into the high-dimensional space. For an example, we could look at the Swiss roll in Figure 3.1. The data is three dimensional, but the data lies on a curved manifold which is rolled on itself. A manifold learning approach wishes to unroll the manifold such that we are left with an embedding of uniformly distributed points in a two dimensional space. A number of assumptions are made to achieve this goal. A common assumption is that the data is uniformly distributed across the manifold. Although this is certainly true in the completely fabricated Swiss roll data set, real-life data sets seldom behave this way. Another phenomenon are manifolds with holes in them, or manifolds with loops in them [38]. Many NLDR algorithms fail to anticipate these characteristics and may deliver a flawed embedding.

Although in practice it may seem tempting to pick one of the sophisticated algorithms described below and throw them at whatever problem is at hand, caution should be exercised when interpreting the results of NLDR algorithms. It is also a good idea to first try a (or several) linear algorithms such as PCA in case it is sufficient as [67] shows in its comparison that PCA often outperforms NLDR algorithms. Clearly, more research is needed and manifold learning remains an active field with many application areas.

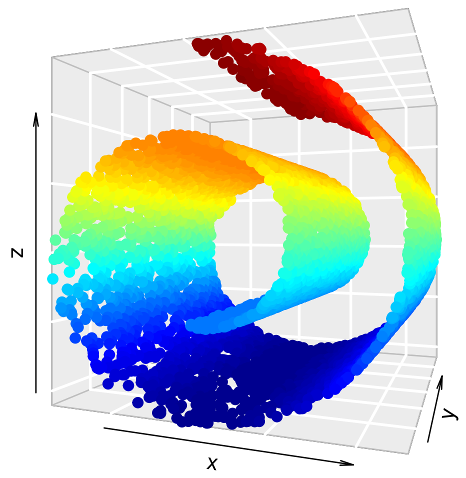


Figure 3.1: The Swiss roll data set

3.2.1 Kernel PCA

Before we dive into manifold learning we want to mention Kernel PCA, which elevates the traditionally linear PCA into potentially nonlinear feature spaces. The famous *kernel trick* has found ample application in the reformulation of linear techniques in recent years, yielding kernelized techniques for regression, classification, clustering and other kinds of pattern analysis [58]. Kernel PCA is one such method where instead of the covariance matrix XX^T we analyze the Gram matrix K of the chosen kernel function κ. Using a nonlinear kernel function results in a nonlinear mapping.

Kernel PCA shares its downsides with traditional linear PCA. Kernel PCA still focuses mainly on large distances, though now in the kernel-mapped feature space, and scales with the number of instances in the data set. Choosing the kernel function is also no small feat.

If the kernel function is isotropic (that is, depends only on $||x - y||$), it can be shown that Kernel PCA is equivalent to metric MDS [71].

3.2.2 Isomap

Isomap [61] is a nonlinear method based on MDS. The problem Isomap seeks to solve can be illustrated using the Swiss roll data set depicted in Figure 3.1—a synthetic data set where the points are arranged in a swirl. Preexisting MDS approaches using e.g. the euclidean distance are ignorant to the intrinsic curved manifold clearly present in the data and would instead base the low-dimensional mapping on the euclidean features of the data, where different pairs of points would appear close than in the embedded manifold.

To circumvent this problem, another distance measure must be used that is aware of the intrinsic manifold. The route taken by the authors is to approximate the geodesic distance by construction of a k-nearest neighbors graph (or, instead of using a k, specifying a range query with some ϵ) and computation of shortest paths using e.g. Dijkstra's algorithm. The shortest paths efficiently approximate the geodesic distance between points x. The estimated distances can then be input to a classical MDS algorithm to construct a low-dimensional mapping with similar euclidean distances between points y, that is, by optimizing the objective $\sum_i \sum_j (d_X(x_i, x_j) - d_Y(y_i, y_j))^2$ or a variation thereof.

The problems of Isomap stem from the approximation of the geodesic distance. Isomap suffers from being topologically unstable [4]. If the parameter k is chosen to be too large with respect to the underlying manifold or noise moves the points ever so slightly off the manifold, the algorithm can construct incorrect paths in the k-nearest neighbors graph, leading to a short-circuiting error [38] which can drastically alter the embedding for the worse.

3.2.3 Maximum Variance Unfolding

Maximum Variance Unfolding [69] takes an approach that is related to Kernel PCA. Instead of choosing a kernel function κ and decomposing its Gram matrix K, we instead derive the Gram matrix from the data by optimizing an objective.

MVU constructs a k-nearest neighbor graph and aims to keep the distances between inputs (high-dimensional points) the same as the distances between outputs. At the same time, the unfolding operation is performed by maximizing the distance between output points that are not nearest neighbors. A naïve formulation as a quadratic problem proves intractable and is also non-convex [70], but the authors have shown [69] that it can be rewritten as a convex semidefinite optimization problem:

$$\begin{aligned} \text{maximize} \quad & \operatorname{tr}(K) \\ s.t. \quad & k_{ii} + k_{jj} - 2k_{ij} = ||x_i - x_j||^2 \; \forall \; (i,j) \in G \\ & \sum_{ij} k_{ij} = 0 \\ & K \succeq 0 \end{aligned} \tag{3.7}$$

The solution Gram matrix K, where $k_{ij} = \langle y_i, y_j \rangle$, is the input to Kernel PCA and the eigendecomposition yields the low-dimensional mapping.

MVU approximates the manifold using a k-nearest neighbor graph, similar to Isomap, and suffers from a similar short-circuiting problem. Incorrect neighbor connections impose additional constraints on the optimization and can prevent the successful unfolding of the manifold. Extensions such as RMVU [30] that detect and eliminate short-circuit edged and relax the constraint that the local distances are retained are relaxed.

3.2.4 Locally Linear Embedding

Locally Linear Embedding [56, 54] (LLE) also proceeds by computing a k-nearest neighbor graph, similar to both MVU and Isomap. The intuition behind LLE is that we can reasonably expect a point and its closest neighbors to lie on a locally linear area of the manifold. Every point can thus be reconstructed by a linear combination of its neighbors. This leads to the objective

$$\mathcal{E}(W) = \sum_i ||x_i - \sum_j w_{ij} x_j||^2 \tag{3.8}$$

The weights are easily found by least squares. The reconstruction of any neighborhood within its region can be seen as a characterization of the local geometry, which should ideally be preserved in the embedding. Finding the low-dimensional representation Y with the known reconstruction weights can thus be performed similarly by

$$\text{minimize} \quad \sum_i ||y_i - \sum_j^k w_{ij} y_j||^2 \tag{3.9}$$

$$\text{s.t.} \quad ||y^{(k)}||^2 = 1 \; \forall k \tag{3.10}$$

with the constraint excluding the trivial solution of $Y = 0$ [67]. The optimum can be found, again, by eigendecomposition and selecting the eigenvectors corresponding to the r smallest eigenvalues.

LLE is a popular method but has also been criticized for subpar embedding results. In [41], the authors found that LLE produces exceedingly distorted embeddings, a problem which Isomap does not exhibit. In [31], it is again found that the results found by Isomap are superior to those of LLE. One possible explanation is the problems LLE has with manifolds which have holes, as already discussed in [54], but LLE has also been criticized for the its covariance constraint, which is deemed to be too simple [67].

Modified LLE [73] improves the stability of LLE and produces less distorted embeddings by introducing additional weights for every neighborhood.

3.2.5 Laplacian Eigenmaps

Similar to LLE, Laplacian Eigenmaps [7] also focuses on the local neighborhoods of points, but instead of reconstructing each point from its neighbors, Laplacian Eigenmaps minimizes the distances between a point and its neighbors. The contribution of the distance to a neighbor is weighted by a heat kernel

$$w_{ij} = e^{-\frac{||x_i - x_j||^2}{t}}, \tag{3.11}$$

resulting in near neighbors contributing more to the cost function than neighbors further away. The optimization problem is then simply

$$\text{minimize} \quad \sum_{i,j} ||y_i - y_j||^2 w_{ij}. \tag{3.12}$$

This minimization problem can be reformulated into an eigenproblem by computing the degree matrix D of the graph defined by W, where $d_{ii} = \sum_j w_{ij}$, and its graph Laplacian $L = D - W$ by showing [7] that

$$\sum_{i,j} ||y_i - y_j||^2 w_{ij} = 2Y^T LY \tag{3.13}$$

which results in the optimization problem

$$\text{minimize} \quad Y^T L Y \tag{3.14}$$
$$\text{s.t.} \quad Y^T D Y = I_n \tag{3.15}$$

The solution Y can be found by solving the generalized eigenvalue problem and selecting the eigenvectors corresponding to the r smallest eigenvalues. Laplacian Eigenmaps suffer due—to their similarity—from much of the same problems of LLE, partly again due to a covariance constraint which should banish a trivial solution but instead prevents from selecting a truly optimal solution [67]. Note that the proof of the algorithm in [7] makes the assumption that the data is uniformly distributed on the manifold, which is a problematic assumption to make because real-world data is seldom that well behaved.

Hessian LLE [22] is a related method that replaces the Laplacian with the local Hessians averaged over all points. The Hessian is found by obtaining the k-nearest neighbors which are used to estimate tangent points in the manifold. The solution is found by eigenanalysis. Due to the similarity to LLE and Laplacian Eigenmaps, Hessian LLE suffers from much of the same drawbacks, but may perform better if the latent manifold of the data is 'curvy' [67].

Another related method is Local Tangent Space Alignment [74] where, similar to Hessian LLE, the manifold is characterized using tangent spaces estimated from the k-nearest neighbors of every point.

3.2.6 t-SNE

t-SNE [66] (t-distributed stochastic network embedding) is a variation of SNE that has gained extreme popularity for visualizing high dimensional data sets. In its original formulation, t-SNE begins by converting the high-dimensional distances between points into conditional probabilities modeling the the probability that a point x_i would pick point x_j as its neighbor. The neighborhood of a point in both the high-dimensional and low-dimensional space is modeled as a probability distribution such that the neighborhood in the low-dimensional space such that the difference between the two distributions can be minimized.

The neighborhood of every point was later optimized to only include the k nearest neighbors of every point in the Barnes-Hut version of t-SNE [65], which corresponds to the construction of a k-nearest neighbor graph as in previously discussed methods. The conditional probabilities are given by

$$p_{j|i} = \frac{\exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k\neq i}\exp(-||x_i - x_k||^2/2\sigma_i^2)}, \tag{3.16}$$

which is symmetrized to a joint distribution $p_{ij} = \frac{p_{j|i}+p_{i|j}}{2n}$. Essentially, this normalizes the different densities found in the data. The parameters σ_i are the variances of a Gaussian centered over point x_i, which is found such that $Perp(P_i) = 2^{H(P_i)}$, where $H(P_i)$ is the Shannon entropy of the probability induced by a certain choice of σ_i and $Perp(P_i)$ is a fixed input. The entropy decreases monotonically as σ_i increases, so a simple binary search can be performed to find the proper σ_i.

The perplexity is an important input parameter which intuitively corresponds to the number of neighbors of every point. While the authors claim that t-SNE is fairly robust to different choices of the perplexity (and suggest a choice between 5 and 50), practical papers such as [33] have criticized that the perplexity is difficult to choose and has significant influence on the quality of the visualization. Schemes for automatic selection of the perplexity such as [13] have also been proposed, although a common strategy for perplexity selection is still trial-and-error.

The remaining work and main idea is to define a joint probability Q that models the pairwise similarities in the low-dimensional space by

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k\neq l}(1 + ||y_k - y_l||^2)^{-1}}, \tag{3.17}$$

which employs a Student t-distribution with one degree of freedom, i.e. a heavy tailed distribution. The authors argue that the mismatch in tail heaviness (compared to the Gaussian) can compensate the mismatch in dimensionalities and alleviate the crowding problem inherent to dimensionality reduction because it maps moderate distances in the high-dimensional space to high distances in the low dimensional space.

t-SNE then minimizes the Kullback-Leibler divergence

$$KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}, \tag{3.18}$$

which measures how different two probability distributions are. The gradient of t-SNE takes the simple form

$$\frac{\partial KL(P||Q)}{\partial y_i} = \sum_{j\neq i} p_{ij} q_{ij} Z(y_i - y_j) - \sum_{j\neq i} q_{ij}^2 Z(y_i - y_j) \tag{3.19}$$

where $Z = \sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}$ is a scaling factor. The first summand represents the attractive forces while the second represents the repulsive forces of all points to the current point. Optimizing this objective is not easy as it is non-convex with potentially many local minima as well as computationally intensive with a complexity of $\mathcal{O}(n^2)$. Various heuristics such as early exaggeration, where the values p_{ij} are multiplied by a constant such as 4 or 12 to force the clusters of the embedding to separate are suggested in the paper, exist. The authors also suggest to use a momentum term dependent on the iteration.

The high computational cost is one of the main drawbacks of t-SNE. The currently most used implementation of t-SNE uses the Barnes-Hut simulation algorithm for computation of n-body forces [5]. Barnes-Hut t-SNE [65] improves the runtime to $\mathcal{O}(n \log n)$ by using a quad- or octree to find sufficiently dense cells in the intermediary embedding such that the points within that cell can be summarized as the repulsive forces of the center-of-mass of that cell multiplied by the number of points in the cell. The matrix of high-dimensional affinities P is made sparse by using only the $3 \times$ perplexity nearest neighbors of every point using vantage trees in $\mathcal{O}(n \log n)$, which makes the complexity of computing the attractive forces for one point constant.

Further optimization attempts include FIt-SNE [42], which further accelerates t-SNE by utilizing the Fourier transform.

Although t-SNE is very widely used, it has also been criticized for, on one hand, its slowness, but also due to common interpretation pitfalls [68] and a lack of mathematical foundation. The interpretation difficulties often stem from the lack of global structure which t-SNE preserves. Often, embeddings consist of multiple clusters with different distances from each other—this may appear like interesting structure that was uncovered by t-SNE to a novice user, but in fact, the distances between clusters may as well be arbitrary [68, 33].

3.2.7 LargeVis

LargeVis [60] tackles one of the main shortcomings of t-SNE: its high computational cost. t-SNE is quadratic in part due to finding nearest neighbors, which is $\mathcal{O}(dn^2)$ naïvely and can be improved to $\mathcal{O}(d \log n)$ by using tree data structures, e.g. k-d trees [8, 52] or ball trees [43], and the quadratic effort of computing the distances between all points in the intermediary embedding during gradient descent.

LargeVis makes use of random projection trees [17, 19] and combines them with NN-Descent [21], a method for efficient k-nearest neighborhood graph (k-NNG) construction. The rationale for this is that while random projection trees are fast and often sufficiently accurate, but when a very accurate k-NNG is needed, many trees must be constructed, which is too costly on average.

They propose to build a fixed few random projection trees, construct a k-NNG and refine it by using NN-Descent. The conditional probabilities P are computed and symmetrized just as in t-SNE.

The rest of the algorithm consists of a likelihood maximization of the objective

$$\sum_{(i,j)\in E} w_{ij} \log P(e_{ij} = 1) + \gamma \sum_{(i,j)\in \overline{E}} \log(1 - P(e_{ij} = 1)). \tag{3.20}$$

The second part of Equation 3.20 considers *negative edges*, i.e. edges that do not exist in the k-NNG graph and γ is an input parameter which dictates the importance of negative edges (i.e. the repulsive forces) to the optimization problem. Since the number of negative edges is quadratic in n, negative sampling is employed by only sampling a fixed number M of negative edges from a noisy distribution for every point. The overall complexity of LargeVis is $\mathcal{O}(rMN)$.

The choice of probability function $P(\cdot)$ is free and discussed in the paper [60]. One of the most important changes from t-SNE to LargeVis is that the low-dimensional similarities no longer play into the objective, which makes it possible to use stochastic gradient descent [60].

3.2.8 FIt-SNE

FIt-SNE [42] accelerates t-SNE by approximating the repulsive forces via a grid of equispaced nodes, polynomial interpolation and the usage of the Fast Fourier Transform due to the properties of the kernels

$$K_1(y,z) = \frac{1}{1 + ||y - z||^2}, \quad \text{and} \quad K_2(y,z) = \frac{1}{(1 + ||y - z||^2)^2} \tag{3.21}$$

which play a major role in the of the scaling factor Z and q_{ij}. These kernels are smooth and translationally invariant, i.e. $K(y,z) = K(y+\delta, z+\delta)$ for any and all δ, thus allowing an acceleration by using FFT. This results in a runtime of $\mathcal{O}(N \cdot p^2 + (N_{\text{int}} \cdot p)^2 \log(N_{\text{int}} \cdot p))$ for the case of a two dimensional embedding. The parameter p equals the number of Lagrangian polynomials used to interpolate the kernel, and N_{int} is the number of intervals that makes up the grid in one dimension. p can reasonably be set to 3 and N_{int} is set to the larger of 20 or $(\max_i y_i - \min_i y_i)$, leaving an indeed linear runtime.

Beside developing this linear approximation, the authors also introduce *late exaggeration*, a pendant to t-SNE's *early exaggeration*, which enhances the cluster separation in the embedding.

They also introduce a randomized out-of-core version of PCA for computing the principal components of data sets which exceed memory limitations.

3.2.9 UMAP

UMAP [45] is a departure from previous algorithms insofar that it takes a significantly different approach to the problem of manifold learning by borrowing from topology and category theory. Nevertheless, the algorithm has certain similarities to t-SNE and LargeVis, and we will use this angle to describe UMAP.

At its core, UMAP is a k-NNG based algorithm just like many of the previously described algorithms. UMAP begins by building a local fuzzy simplical set for every point x_i. What this roughly means (we recommend to peruse the documentation of UMAP[1] to get an intuition for the specific mechanisms at work as this is out of scope for this paper) is that UMAP constructs a weighted k-NNG graph, just like other methods, but does so in a way that is rooted in topological data analysis. The weights of the undirected edges of the final graph represent the probability that either point is connected to the other. The low-dimensional embedding is initialized to a spectral embedding, e.g. by Laplacian Eigenmaps although other methods could be used, and is then optimized by minimizing the cross entropy objective

$$\sum_{e \in E} w_h(e) \log\left(\frac{w_h(e)}{w_l(e)}\right) + (1 - w_h(e)) \log\left(\frac{1 - w_h(e)}{1 - w_l(e)}\right) \tag{3.22}$$

where $w_h(e)$ gives the weight of an edge in the high-dimensional topological representation and $w_l(e)$ does the same for the low-dimensional topological representation. Similarly to t-SNE and LargeVis, this objective consists of attractive forces—the first summand—and repulsive forces. Note that the original paper uses membership functions for w_h and w_l, denoted by μ and ν, which in itself is discrete, so the membership strength is smoothed to make the objective differentiable. The algorithm then proceeds by stochastic gradient descent and applies negative sampling for the repulsive forces.

The advantages of UMAP over t-SNE are the significantly higher embedding speed and the strong theoretical foundation upon which UMAP is built. Recent applications such as [6] have cited UMAP to produce excellent embeddings. Notably, UMAP claims to preserve the global structure of data better than t-SNE, however, a more recent paper [34] discovered that the deciding factor in preserving global structure is not UMAP's algorithm itself, but the initialization. When initializing plain UMAP with a random embedding, much of the global

[1] `https://umap-learn.readthedocs.io/en/latest/how_umap_works.html`

method	parametric?	stochastic?	time complexity	# parameters
PCA	no	no	$\mathcal{O}(d^3)$	0
MDS	no	no	$\mathcal{O}(n^3)$	0
LDA	yes	no	$\mathcal{O}(d^3)$	0
Kernel PCA	no	no	$\mathcal{O}(n^3)$	1
Isomap	no	no	$\mathcal{O}(n^3)$	1
MVU	no	no	$\mathcal{O}(k^3n^3)$	1
Laplacian Eigenmaps	no	no	$\mathcal{O}(p_{nz}^2n^2)$	2
t-SNE	no	yes	$\mathcal{O}(n^2)$	1
LargeVis	no	yes	$\mathcal{O}(n)$	1
FIt-SNE	no	yes	$\mathcal{O}(n)$	1
UMAP	no	yes	$\mathcal{O}(n^{1.14})$	1

Table 3.1: Summary of important dimensionality reduction algorithm properties

structure is lost, and similarly initializing t-SNE with an informative method such as Laplacian Eigenmaps or PCA, more global structure is preserved as measured by the Pearson correlation between distances in the high and low-dimensional spaces. Using a different initialization method in t-SNE is an easy modification and thus UMAP's remaining main advantage over t-SNE is its runtime.

3.3 Summary

Table 3.1 shows a summary of the core properties of the methods we discussed in this chapter, similar to what is presented in [67]. Note that we do not include the number of dimensions for the low-dimensional space in the parameters. Most discussed methods are non-parametric, with the exception LDA, which has strong assumptions for the data, namely that all classes distributed according to multivariate Gaussian distributions with a shared covariance matrix but different means. PCA by itself is non-parametric, as extracting the directions of maximum variance using SVD does not impose any assumptions, however if hypothesis testing is to be performed to test how many directions are significant does require distributional assumptions. However, typical usage of PCA usually does not include hypothesis testing.

4 Approximative Linear t-SNE Using k-Means

In this chapter we discuss the various aspects of our proposed algorithm, including the initialization method, the approximate nearest neighbors scheme, the usage of k-Means to segment the low-dimensional space and exaggeration.

4.1 Outline

The most widely used version of t-SNE is currently the Barnes-Hut approximation. The speedup achieved by this version comes from the usage of spatial trees, i.e. quadtrees for two-dimensional embeddings and octrees for three-dimensional embeddings. The output dimensionality is thus capped at three, which is a very slight impediment as t-SNE is generally only used for visualization purposes due to its non-parametric nature.

Barnes-Hut t-SNE [65] partitions the embedding space into cells using trees. If cells are sufficiently dense (quantified by a parameter θ, the diagonal of the cell), the cell is summarized such that all points in the cell lie on the center-of-mass of that cell. Since the high-dimensional affinities p_{ij} go quickly to zero, only the $3 \times$ perplexity nearest neighbors are used (and found, again, using spatial trees), which leaves the P matrix sparse and further greatly reduces the computations needed.

The ideas behind our algorithm are similar. Instead of using a tree-based k nearest neighbor algorithm, we use locality sensitive hashing to approximate the nearest neighbors in constant time, and instead of finding cells to summarize using a spatial tree, we run k-means on the embedding and use the cluster centers as a summary. Running k-means is much cheaper than building a data structure and should thus yield a significant speedup when computing the repulsive forces, and it has recently been shown [42] that approximate nearest neighbors perform similarly well as the exact nearest neighbors for the purposes of t-SNE, which ties into using LSH. However, we must note that our approximations have the potential to be more crude and thus may impose a larger approximation error.

4.2 Initialization

The original variant of t-SNE [66] initializes the embedding using a Gaussian distribution with a standard deviation of 10^{-4}. More recent work, particularly UMAP [45] have used informative methods such as Laplacian Eigenmaps or PCA to initialize the intermediary embedding to a low(er) quality, quick-to-compute embedding. It has been argued [34] that this initialization—rather than differences in the optimization process—leads to the conservation of more global structure in the resulting final embedding found by UMAP.

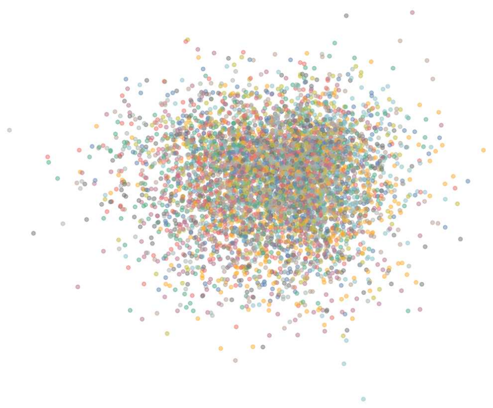
We also utilize PCA as an embedding initialization as it is easier to compute than the Laplacian Eigenmap embedding. Figure 4.1 shows the difference this makes in early embeddings (10 iterations). The PCA initialization scheme offers a head start, but the random initialization quickly catches up. However, the Pearson correlation coefficient is significantly higher for the PCA initialized version. More on this can be read in the Parameter Tuning section of the next chapter.

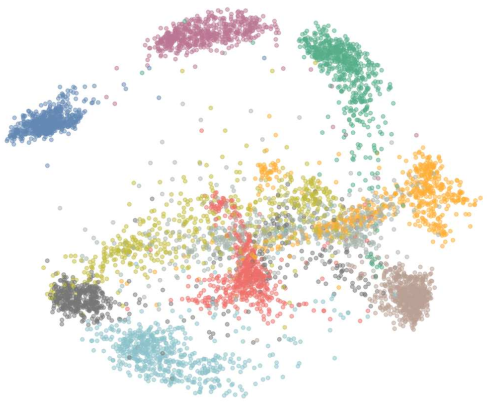
4.3 Approximate Nearest Neighbors

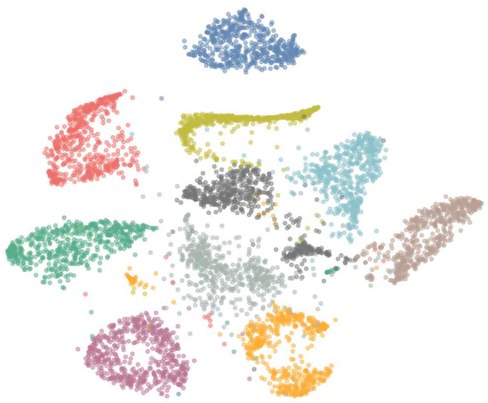
As has been shown in [42] and also exploited in [60], it is sufficient to use approximate nearest neighbors (ANNs) rather than exact nearest neighbors for the computation of the high-dimensional affinities p_{ij}. This allows for further optimizations in finding nearest neighbors beyond a complexity of $\mathcal{O}(n \log n)$.

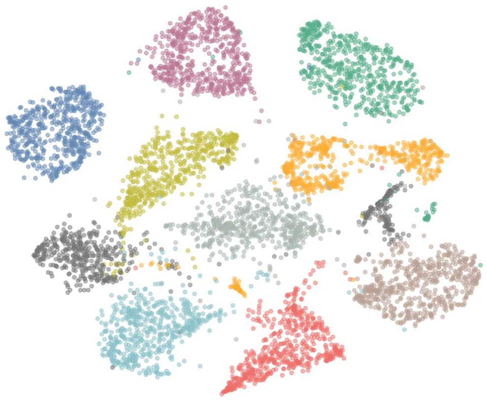
FIt-SNE [42] uses a library called ANNOY [1] which is based on random projection trees. LargeVis [60] similarly uses random projection trees to partition the high-dimensional space and limit the candidate set of (probable) nearest neighbors. UMAP [45] uses NN-descent [21], for which only an empirical runtime of roughly $\mathcal{O}(n^{1.14})$ is known, which is not far off, but *not* linear.

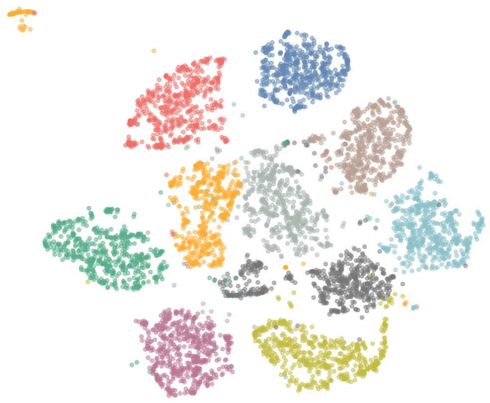
We chose a different class of ANN algorithm and used locality sensitive hashing. The reasoning behind this was the belief that a rather crude nearest neighbors should be good enough and hence the added overhead of many of the aforementioned would be unnecessary (e.g. many trees, high construction cost of trees), i.e. the main reason was simplicity. However, we want to note that any ANN algorithm could be used here as long as its complexity is a true $\mathcal{O}(n)$ as to preserve the overall runtime of $\mathcal{O}(n)$ of our method.


(a) Random initialization, 10 iterations


(b) PCA initialization, 10 iterations


(c) Random initialization, 100 iterations


(d) PCA initialization, 100 iterations


(e) Random initialization, 1000 iterations

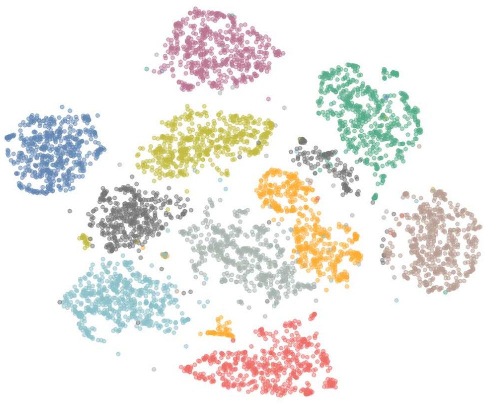

(f) PCA initialization, 1000 iterations

Figure 4.1: Influence of initialization on very early, early and late embeddings

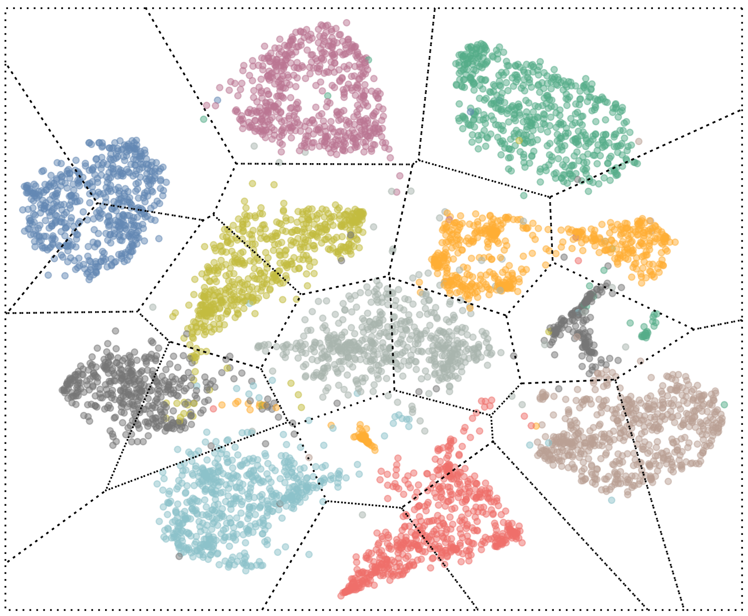
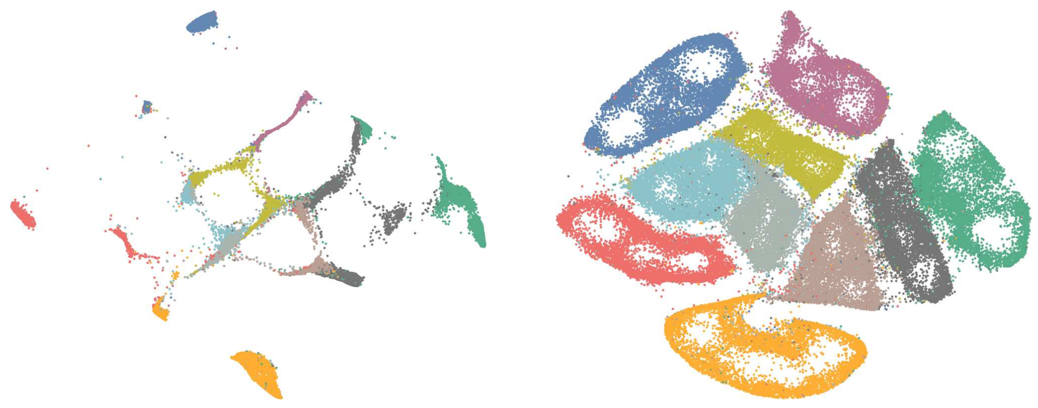


Figure 4.2: Example Voronoi diagram showing the cells implied by a k-means clustering

4.4 Segmentation of Low Dimensional Space

As previously discussed, the major novelty of our algorithm consists of using k-means to segment the low-dimensional space (containing the intermediary embedding) into cells. This idea stems from the familiar way to depict the assignment step of k-means using Voronoi diagram. Figure 4.2 shows an example Voronoi diagram for k-means with $k = 20$ on an early embedding of the optdigits dataset obtained by using our method for 100 iterations. The Voronoi cells are delimited by the dotted black lines, and the center of mass of each cell is a centroid found by k-means. As can be seen, the cells adapt to the general shape of the data, cutting some of the clusters into multiple parts as k is larger than the number of clusters inherent to the data. It stands to reason that these cells would constitute a good way to summarize the points within them, and this is what is done in our algorithm.

The parameter k can be set arbitrarily, however we recommend that it is set significantly larger as the optimization process will otherwise converge to a distorted embedding due to clusters separating early and not being reconsidered (e.g. ripped apart) during optimization. Figure 4.3 shows an example of this. Although the points are relatively class-pure, the clusters are connected and oblong in shape in the version with a k of 5 (Figure 4.4a). On the other



(a) $k = 5$ (b) $k = 30$

Figure 4.3: MNIST dataset with $k = 5$ (too small) and $k = 30$ (just right)

hand, a k of 30 (Figure 4.4b) results in the more round cluster shapes that are typical for t-SNE, and the clusters are better separated.

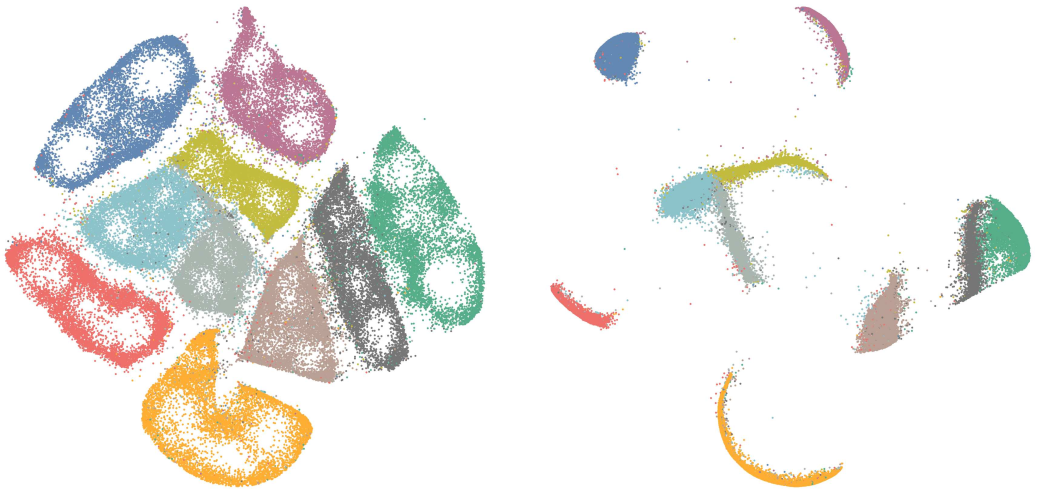
4.5 Early Exaggeration

Early exaggeration is a simple trick that was proposed along with the original version of t-SNE and consists of multiplying the symmetrized high-dimensional affinities p_{ij} by some scalar α, thus making them much more dominant over their low-dimensional counterparts q_{ij} which play a larger role in the repulsive forces. The authors argue that early exaggeration keeps points close in early iterations and allows them to move more freely, compress more and put larger distance between different clusters. We implement early exaggeration and apply it for the first 25% of gradient descent.

4.6 Late Exaggeration

Late exaggeration was proposed along with FIt-SNE [42] and has since been widely adopted. The idea is largely congruent with early exaggeration and yields embedding with better cluster separation by contracting the clusters. Our method also implements early exaggeration. Figure 4.4 shows an example of the effect of late exaggeration.

There is little guidance on how this late exaggeration parameter should be set or for how long late exaggeration should be run. We chose to apply late exaggeration for the last 10%



(a) No late exaggeration (b) Late exaggeration factor of 12

Figure 4.4: MNIST dataset without and with late exaggeration

of gradient descent iterations, however we want to note that there exist suggestions to apply late exaggeration starting immediately after early exaggeration ends [32]. We find this to be extreme and found that it is not the norm for other implementations that implement late exaggeration either.

4.7 Optimization

The objective of t-SNE, the Kullback-Leibler divergence, is difficult to optimize and necessitates various measures to ascertain convergence to a sensible embedding. We utilize momentum along with an individual gains scheme that amplifies elements of the gradient that change their sign between iterations and attenuates elements with unchanging sign. We also center the embedding in every iteration.

The objective in non-convex and thus the optimization process may find any of the possible local minima but not necessarily the global minimum. Furthermore, the exact objective is expensive to compute as it requires evaluating the attractive and repulsive forces between all pairs of points, hence the runtime of $\mathcal{O}(n^2)$. The gradient of t-SNE can be written as

$$\frac{\partial KL(P||Q)}{\partial y_i} = \sum_{j \neq i} p_{ij} q_{ij} Z (y_i - y_j) - \sum_{j \neq i} q_{ij}^2 Z (y_i - y_j), \tag{4.1}$$

omitting a constant factor of 4 [66]. The first sum constitutes the attractive forces, while the second part constitutes the repulsive forces. If P is sparse due to only computing the high-dimensional affinities of the $3p$ nearest neighbors, the number of summands in the first sum is equal to $3p$. In the case of using LSH, this may even be slightly lower if the candidate set is too small.

Recollect that

$$Z = \sum_{i \neq j} \frac{1}{1 + ||y_i - y_j||^2}, \tag{4.2}$$

that is, an expensive sum involving all pairs of points, and

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}}, \tag{4.3}$$

i.e. the Z term cancels out significant computational cost, and leaves us with

$$F_{\text{attr}} = \sum_{i \neq j} p_{ij} \frac{y_i - y_j}{1 + ||y_i - y_j||^2} \tag{4.4}$$

The repulsive forces are more difficult to approximate. p_{ij} is not included, so sparsity is not an option, and q_{ij} is squared, so the expensive scaling factor sum will not cancel out. Thus, we multiply both sides with Z to get

$$F_{\text{rep}} Z = -(q_{ij} Z)^2 (y_i - y_j) \tag{4.5}$$

causing Z to cancel out the expensive sum in the denominator of q_{ij}, leaving

$$q_{ij} Z = \frac{1}{1 + ||y_i + y_j||^2}. \tag{4.6}$$

We now substitute y_j with the centroid of the cell c $\bar{y}_c$, multiply with n_c, the number of points in the cell, and get

$$F_{\text{rep}} Z = \sum_{i=1}^{n} \sum_{c=1}^{k} -\frac{y_i - \bar{y}_c}{1 + ||y_i - \bar{y}_c||^2} \cdot n_c. \tag{4.7}$$

The last thing left to do is to approximate Z in a similar way by

$$\hat{Z} = \sum_{i=1}^{n} \sum_{c=1}^{k} \frac{n_c}{1 + ||y_i - \bar{y}_c||^2} \tag{4.8}$$

and then divide $F_{\text{rep}}Z$ by $\hat{Z}$ to get $\hat{F}_{\text{rep}}$.

Note that this approach is the same as taken with Barnes-Hut t-SNE [65] because our method is functionally the same with regard to the optimization process, and only outlined here to fully explain our method.

In contrast to Barnes-Hut the number of cells is fixed. Barnes-Hut t-SNE uses an input parameter θ to summarize cells based on their distance to the current point and the cell size, namely

$$\frac{r_{\text{cell}}}{||y_i - y_{\text{cell}}||^2} < \theta, \tag{4.9}$$

where r_{cell} is the diagonal of the cell. If θ is set to zero, exact t-SNE is performed. While this method may perform better as is adapts to the specifics of the data, our method is easier to understand and has an especially straightforward scaling behavior. We will see how it performs in the next chapter.

Algorithm 4.7.1 summarizes our method and puts the parts together, and Figure 4.5 shows a higher-level diagram of our method. Refer to Appendix A for an overview of our implementation.

An important technical detail of the implementation is that we accelerated the computation of the all-pairs squared distance by utilizing the binomial theorem yielding the formula $(a - b)^2 = a^2 - 2ab + b^2$. For A and B in $\mathbb{R}^2$ this means component-wise summation of the squared values in A and B, and a matrix multiplication $-2AB^T$—our code utilizes the heavily optimized matrix-matrix multiplication routine included in Eigen [28], which is significantly faster than a naïve computation. See Appendix B for a small experiment on this.

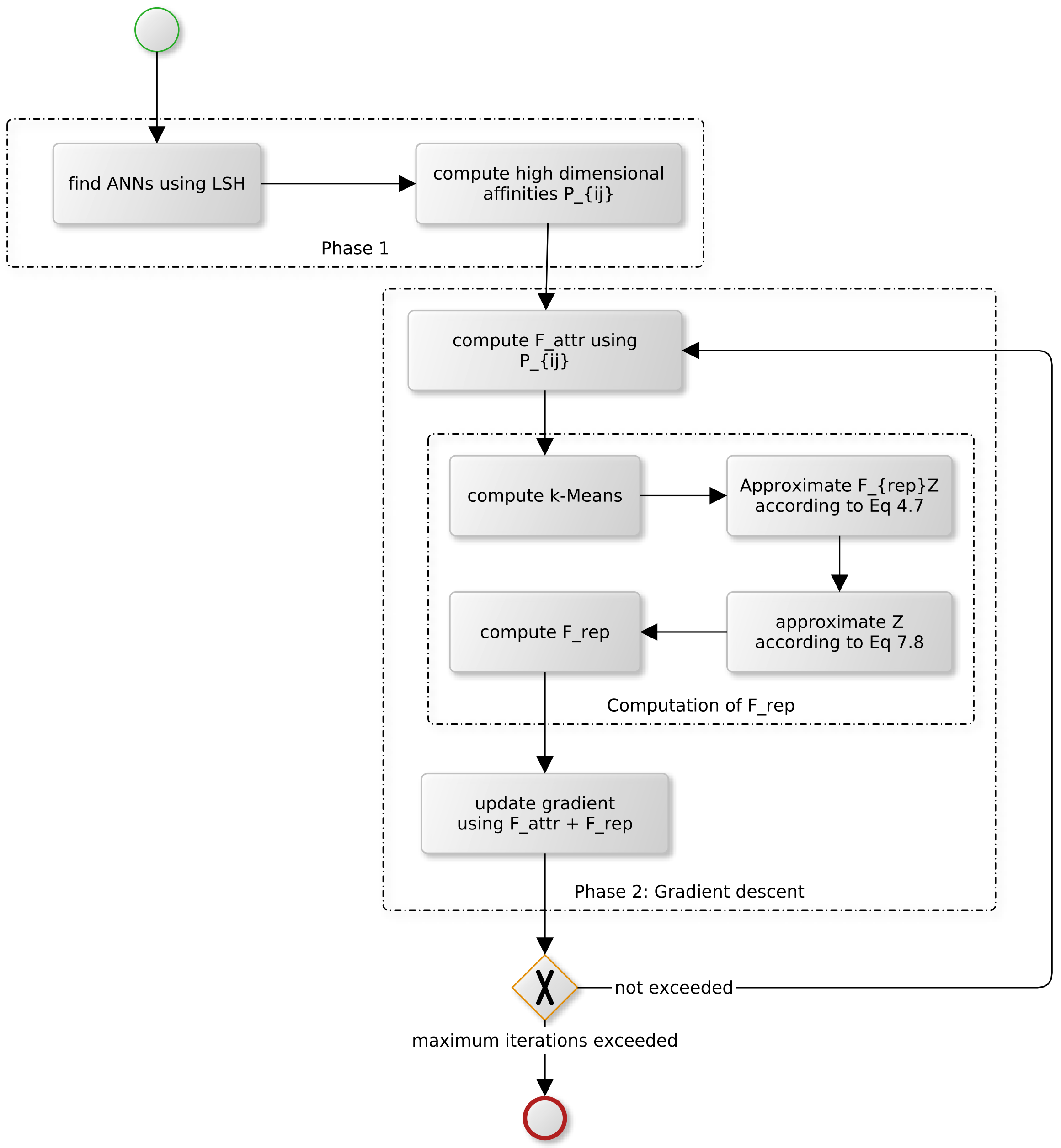

find ANNs using LSH
compute high dimensional affinities P_{ij}
Phase 1
compute F_attr using P_{ij}
compute k-Means
Approximate F_{rep}Z according to Eq 4.7
approximate Z according to Eq 7.8
compute F_rep
Computation of F_rep
update gradient using F_attr + F_rep
Phase 2: Gradient descent
not exceeded
maximum iterations exceeded

Figure 4.5: Diagram of our method using BPMN notation

Algorithm 4.7.1 Our proposed algorithm ktsne

```
procedure KTSNE($X$, $p$, $k$)
    for all points $x_i$ do                    ▷ Phase 1: Computing high-dimensional affinities
        determine $3 \times$ perplexity nearest neighbors of $x_i$ using LSH
        for all neighbor points $x_j$ do
            determine $\beta$
            $p_{j|i} \leftarrow \frac{\exp(-\beta||x_i - x_j||^2)}{\sum_j \exp(-\beta||x_i - x_j||^2)}$
        end for
    end for
    $P_{ij} \leftarrow \frac{P_{j|i} + P_{i|j}}{2n}$ for all $i, j$                    ▷ $P$ is sparse!
    $Y \leftarrow$ initial embedding (Gaussian or PCA-reduction of $X$)
    repeat                                     ▷ Phase 2: Gradient descent
        for all $i, j$ for which $p_{ij}$ is nonzero do
            compute $F_{\text{attr}}$ as specified by Equation 4.4, exaggerating $p_{ij}$ if required
        end for

        compute clustering of embedding using $k$-Means, yielding centroids $c_1, c_2, \ldots c_k$
        compute $F_{\text{rep}}Z$ as specified by Equation 4.7
        approximate $Z$ as shown in Equation 4.8
        divide $F_{\text{rep}}Z$ by $\hat{Z}$
        compute the gradient as $F_{\text{attr}} - F_{\text{rep}}$
        update $Y$ using the gradient, momentum and individual gains
    until maximum iterations exceeded
end procedure
```

5 Experimental Evaluation

In this chapter we evaluate our previously proposed method. We perform parameter tuning for parameters such as the number of cluster centers and exaggeration. We present the data sets that we used and discuss the approach to our experiments. Finally, we compare our method to to three competing methods.

5.1 Data Sets

We evaluated our proposed algorithm on 17 data sets obtained from various sources, e.g. the Stanford Network Analysis Project or the UCI Machine Learning Repository. We supplemented with graph embeddings. For these, we used VERSE [64] with a parameterization of $\alpha = 0.85$ and $n_{\text{samples}} = 3$. Table 5.1 lists all data data sets sorted by their size. We reduced all data sets to a dimensionality of 50 using PCA prior to running any algorithm, as was done in the experimental evaluations of the original t-SNE as well as the Barnes-Hut version.

For the cifar and svhn data sets, we trained a convolutional neural network and obtained the 128 activations of the first dense layer after the convolutional stages. The CNN achieved an accuracy of over 90% on held-out test data.

The youtube, com-amazon, com-dblp and hollywood data sets were unlabeled graphs. We generated labels using the graph clustering algorithm VieClus [9].

5.2 Setup and Methodology

Experiments were conducted on a Intel Xeon Gold 6130 CPU @ 2.10GHz with 16 hyperthreaded cores and 93 GiB memory. Experiments were run in parallel, but not exceeding 16 threads. Methods that support multithreading were run with 4 threads. Every method was run 5 times with different fixed seeds.

For comparability and due to limitations of our method discussed below we rely on using the average k-nearest neighbor class purity as a quality measure of an embedding, with $k = 100$.

data set	n	d
emailEuCore [39]	1005	128
coil-20 [46]	1440	16,384
cora [53]	2708	128
citeseer [53]	3264	128
optdigits [23]	5620	64
youtube [2]	13,723	128
dblp [50]	17,278	128
pubmed [53]	19,717	128
cifar [35]	50,000	128
mnist [37]	70,000	784
fashion_mnist [72]	70,000	784
com-dblp [39]	317,079	128
com-amazon [39]	334,862	128
svhn [47]	630,420	128
hollywood [53]	1,069,125	128
timit [27][2]	1,105,455	273
wiki-topcats [39]	1,791,488	128

Table 5.1: Data sets used for our evaluation sorted by n

This constitutes a local measure, i.e. methods that better preserve local structure will achieve a better nearest neighborhood purity. For smaller datasets, the average absolute Pearson coefficient

$$r_{xy} = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_{i=1}^{n}(x_i - \bar{x})^2}\sqrt{\sum_{i=1}^{n}(y_i - \bar{y})^2} \tag{5.1}$$

between the high-dimensional distances and low-dimensional distances in the PCA embedding and the final embedding is calculated to quantify how much global structure was kept intact.

For the comparison, the perplexity value of all methods is set to the same value. The step size is set to the same value as well for all methods but UMAP, which suggests a much lower value of 1, possibly related due to scaling internal to UMAP.

[2] A preprocessed version of the TIMIT data set was graciously provided by Laurens van der Maaten. Many thanks!

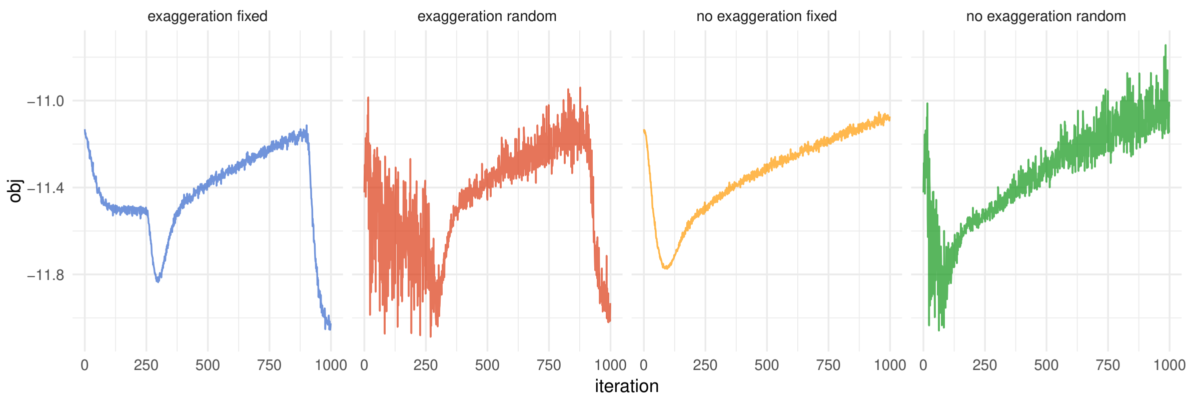


Figure 5.1: Objective function for the pubmed data set, varying in random k and exaggeration

5.3 Performance of Approximated Objective

An important and not at all gratifying finding is the significant oscillation and non-monotony of the Kullback-Leibler divergence. The modifications done to the computation process of the repulsive forces result in an objective that no longer appears to be indicative of the actual quality of the embedding. First and foremost, the objective is almost always negative, possibly due to the estimated q_{ij} being too large, and secondly, the approximated objective increases during the optimization.

Computing the exact objective is again of quadratic runtime order and thus not feasible.

Figure 5.1 shows a comparison of 4 different constellations: random and non-random k, and with and without exaggeration (a value of 12 for both early and late). The objective with random k oscillates strongly. Exaggeration has an interesting effect: it causes the objective to drop. This reinforces our suspicion that the q_{ij}s are overestimated, and this is alleviated by the amplified p_{ij}s when exaggeration is applied.

However, the nearest neighborhood purity does not decrease as suggested by the increasing approximated objective. Figure 5.2 shows the nearest neighborhood purity measured every 100 iterations. The nearest neighborhood purity more or less increases with every 100 iterations, thus showing that our method of approximation does actually improve the embedding quality during optimization, at least as measured by the nearest neighborhood purity.

5.4 Parameter Tuning

Our proposed method has multiple parameters fit for parameter tuning. These include

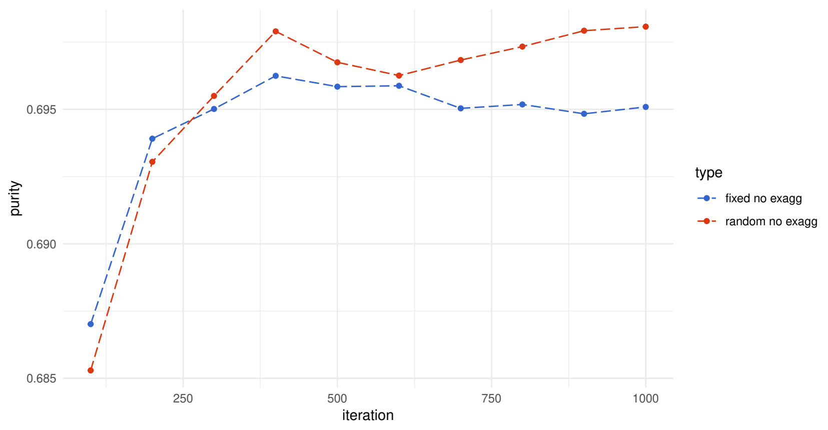


Figure 5.2: Nearest neighborhood purity as measured every 100 iterations

- the perplexity p,
- the gradient descent step size η,
- the number of clusters for k-means k, including randomized values,
- the initialization method (random vs. PCA),
- early and late exaggeration values,
- the hash family,
- as well as the number of hash tables and probes for LSH.

We evaluate using the nearest neighborhood purity metric of the final embedding due to the previously discussed issues with the non-monotony of the approximated Kullback-Leibler divergence. The optimization process is always run for 1000 iterations. We chose two data sets for parameter tuning, namely the optdigits and MNIST data set. They are of moderate size and well-behaved, i.e. they tend to yield well separated embeddings.

Most of these parameters are data dependent and thus it is necessary to re-tune them to get the best embeddings. We simply want to find parameters that are good enough to use for the rest of the evaluation and also study the impact of different parameterizations.

5.4.1 LSH-related Parameters

We first examine the LSH-related parameters as they majorly influence the computation of the high-dimensional affinities which, in turn, have a large influence on the end result of our algorithm. These include the number of hash tables ℓ, the number of probes t used for

LSH family	$\ell = 10$	$\ell = 20$	$\ell = 50$	$\ell = 100$
hyperplane	0.7726	0.8505	0.9006	0.9314
crosspolytope	0.8888	0.9253	0.9329	0.9333

(a) optdigits

LSH family	$\ell = 10$	$\ell = 20$	$\ell = 50$	$\ell = 100$
hyperplane	0.7593	0.8285	0.9075	0.9183
crosspolytope	0.8404	0.8936	0.9217	0.9291

(b) MNIST

Table 5.2: Nearest neighborhood purity per number of hash tables and LSH family

multi-probing and the locality sensitive hash family of choice. For the number of hash bits we use

$$b = \max(16, \log_2 n),$$

where n is the number of data points. We do not tune the number of additional probes t.

We chose to use FALCONN [3] for our implementation, which supports the therein proposed Cross-Polytope hash family as well as the older Hyperplane hash family. Cross-Polytope LSH is reported to be faster as well as yielding superior results, however, the Cross-Polytope locality sensitive hash family is originally designed for data which lies largely on a sphere and for which the angular distance is used. In contrast, our work uses the Euclidean distance exclusively. If data is centered and normalized, the Cross-Polytope hash family can be expected to perform reasonably well, however this is a claim that is worth checking.

Configurations of 10, 20, 50 and 100 hash tables were tried for both Hyperplane and Cross-Polytope LSH. Table 5.2 shows the resulting average nearest neighborhood purity for both data sets. Cross-Polytope LSH clearly results in better embeddings, and more hash tables are also clearly beneficial. It is important to note that running the algorithm with a larger number of hash tables results in a longer runtime—in this case, there was a slowdown of 25% (optdigits) to 55% (MNIST) between using 10 and 100 tables. This slowdown is not so much due to the larger cost of the hashing itself, but also due to the fact that using only 10 tables results in additional sparsity in the high dimensional affinities as the returned candidate sets are often smaller than $3p$. Consequently, there is hardly a difference in runtime between $\ell = 50$ and $\ell = 100$. We proceed with a value of $\ell = 100$.

	$\eta = 0.1$	$\eta = 10$	$\eta = 20$	$\eta = 50$	$\eta = 100$	$\eta = 200$	$\eta = 400$
$p = 3$	0.8092	0.9249	0.9277	0.9233	0.9221	0.9209	0.9157
$p = 5$	0.8208	0.9312	0.9293	0.9368	0.9332	0.9294	0.9226
$p = 15$	0.8398	0.9377	0.9383	0.9369	0.9354	0.9406	0.9336
$p = 30$	0.8491	0.9307	0.9322	0.9329	0.9340	0.9333	0.9262
$p = 50$	0.8476	0.9261	0.9206	0.9299	0.9313	0.9315	0.9293

(a) optdigits

	$\eta = 0.1$	$\eta = 10$	$\eta = 20$	$\eta = 50$	$\eta = 100$	$\eta = 200$	$\eta = 400$
$p = 3$	0.3800	0.7456	0.8049	0.8841	0.9285	0.9350	0.9379
$p = 5$	0.3833	0.7813	0.8310	0.9036	0.9401	0.9492	0.9511
$p = 15$	0.3909	0.8263	0.8468	0.8862	0.9290	0.9381	0.9406
$p = 30$	0.3959	0.8314	0.8587	0.8900	0.9245	0.9291	0.9305
$p = 50$	0.3996	0.8357	0.8566	0.8723	0.8964	0.9183	0.9238

(b) MNIST

Table 5.3: Effects of varying perplexity p and step size η on the nearest neighborhood purity

5.4.2 Basic Parameters: Perplexity and Step Size

Next, we consider the two fundamental parameters inherent to t-SNE: the perplexity p and the step size of the gradient descent η. The base t-SNE and Barnes-Hut t-SNE use a perplexity of 50 and a quite large step size of 200 [66, 65], however our modified variant might fare better with different parameterizations. For this experiment, we again set the number of centroids k to a fixed 30 and used cross-polytope LSH with 100 hash tables. Table 5.3 shows the results for the two tuning data sets.

As the results show, picking a data-independent perplexity value is somewhat difficult. While the optimum value for optdigits seems to be $\eta = 200$ and $p = 15$, the larger MNIST data set fares better with a much smaller perplexity of 5. This value is at the very bottom of the recommended value range for the perplexity suggested in the paper [66], thus we choose to err on the side of caution and pick a perplexity of 30, which will likely perform better especially with larger data sets. We set η to 200. Coincidentally, these are also the default values of the implementation contained in scikit-learn.

dataset	$k = 5$	$k = 10$	$k = 30$	$k = 50$	$k = 100$	$k \sim U(5, 50)$
optdigits	0.9318	0.9449	0.9348	0.9291	0.9391	0.9351
mnist	0.9169	0.9271	0.9291	0.9300	0.9318	0.9269

(a) Nearest neighborhood purity

dataset	$k = 5$	$k = 10$	$k = 30$	$k = 50$	$k = 100$	$k \sim U(5, 50)$
optdigits	18.94	20.51	28.26	38.41	64.13	29.3
mnist	447.69	467.28	583.87	993.72	1639.68	546.73

(b) Runtime in seconds

Table 5.4: Effects of varying k on nearest neighborhood purity and runtime

5.4.3 Number of Cluster Centers

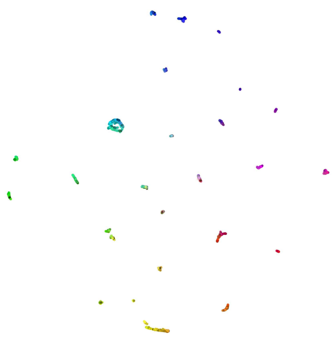
Another important parameter impacting the quality of the resulting embedding but also the runtime is the number of centroids to be used for k-means. Table 5.4a shows the results for ks of 5, 10, 30, 50, 100 and also for uniformly random ks between 5 and 50 (i.e. a different k for every iteration). Other parameters were set as previously determined.

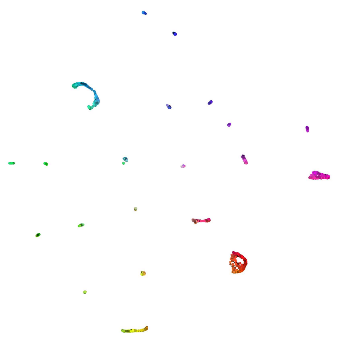
Unsurprisingly, a larger k results in a higher nearest neighborhood purity. However, the runtime when using a large k also increases considerably. Table 5.4b shows the corresponding runtimes. The runtimes for $k = 30$ and $k \sim U(5, 50)$ are very similar, which is reasonable as the expected value of k is 27.5—but the increase in nearest neighborhood purity is negligible or even negative in the case of optdigits. Considering the runtime costs versus the nearest neighborhood purity, we pick a k of 30.

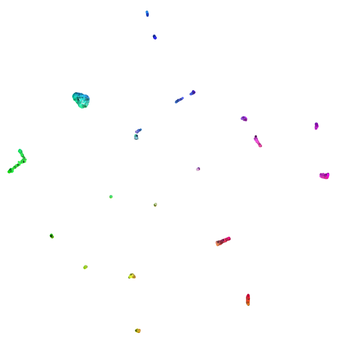
Figure 5.3 shows a matrix of embeddings of 10,000 samples from the RGB color cube, similar to an experiment that is shown in the UMAP [45] paper, for the number of clusters and the perplexity. A very small k has a much more drastic effect at limiting the optimization and results in a scattered embedding with tightly compressed clusters of similar colors, though this could also be considered a desirable outcome, depending on the requirements. Larger k allow the colors to spread out more, and the resulting embedding looks more like expected.

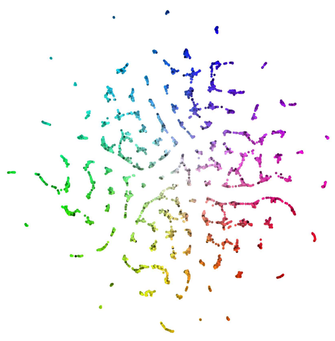
5.4.4 Initialization Method

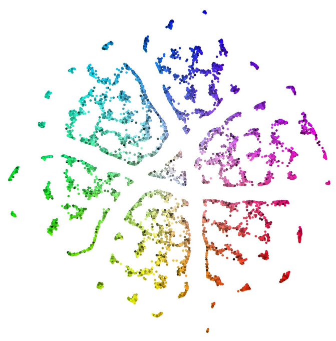
Next, we want to compare two initialization methods: random initialization sampled from a Gaussian distribution with variance 10^{-4} and initialization by PCA of the input data set (i.e. a reduction to the target dimensionality of 2). Table 5.5 shows the results.

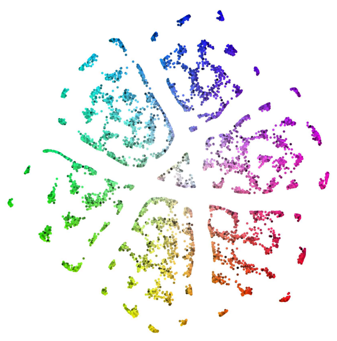

(a) $p = 5;\ k = 5$

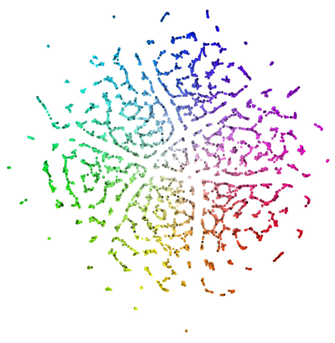

(b) $p = 25;\ k = 5$

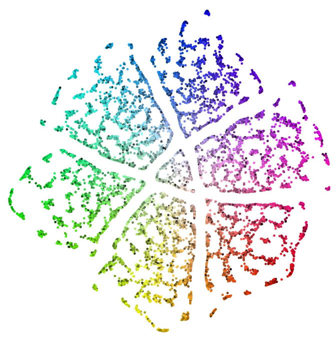

(c) $p = 50;\ k = 5$


(d) $p = 5;\ k = 50$


(e) $p = 25;\ k = 50$


(f) $p = 50;\ k = 50$


(g) $p = 5;\ k = 100$


(h) $p = 25;\ k = 100$

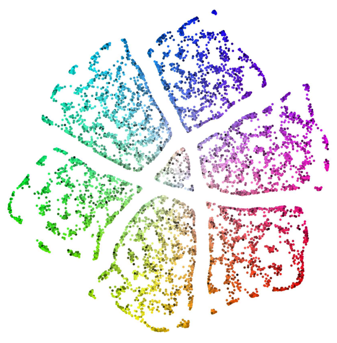

(i) $p = 50;\ k = 100$

Figure 5.3: Effects of varying perplexity p and number of cluster centers k

dataset	pca	random
optdigits	0.9332	0.9396
mnist	0.9291	0.9217

(a) Nearest neighborhood purity

dataset	pca	random
optdigits	0.5579	0.4909
mnist	0.4015	0.3621

(b) Average Pearson correlation coefficient between high and low-dimensional distances

Table 5.5: Impact of different initialization schemes on nearest neighborhood purity and correlation

The PCA initialization performs slightly better on MNIST, but slightly worse than random for optdigits. There are no significant runtime differences.

These results are somewhat surprising as we expected PCA to be a much better initialization, but in light of what is reported in [34] it is the global structure that is more likely to be preserved; and the here used metric of nearest neighborhood purity is more apt to quantify the local structure. The Pearson correlation coefficient between high and low-dimensional distances is often used to quantify the amount of preserved global structure and is shown in Table 5.5b and indeed shows that the PCA has a much larger positive influence here.

5.4.5 Early and Late Exaggeration

Finally, we turn to the late and early exaggeration factors. These factors artificially inflate the high-dimensional affinities P_{ij} during a certain number of iterations and cause the resulting embedding to consist of more well-separated and/or compressed clusters as the attractive forces play a larger role.

We denote early exaggeration values as α and late exaggeration values as ω and try five values each: $\alpha = 1, 1.5, 4, 8, 12$ and $\omega = 1, 1.5, 4, 8, 12$. Table 5.6 shows the results for optdigits and MNIST. It is clear that the largest value performs best in both cases, for both early and late exaggeration. We thus pick $\alpha = \omega = 12$ for further experiments.

	$\alpha = 1$	$\alpha = 1.5$	$\alpha = 4$	$\alpha = 8$	$\alpha = 12$
$\omega = 1$	0.9318	0.9431	0.9311	0.9340	0.9333
$\omega = 1.5$	0.9409	0.9485	0.9409	0.9403	0.9410
$\omega = 4$	0.9544	0.9559	0.9566	0.9572	0.9574
$\omega = 8$	0.9551	0.9566	0.9547	0.9587	0.9581
$\omega = 12$	0.9549	0.9554	0.9534	0.9585	0.9585

(a) optdigits

	$\alpha = 1$	$\alpha = 1.5$	$\alpha = 4$	$\alpha = 8$	$\alpha = 12$
$\omega = 1$	0.8683	0.8963	0.9261	0.9267	0.9291
$\omega = 1.5$	0.8718	0.8981	0.9266	0.9274	0.9299
$\omega = 4$	0.8951	0.9099	0.9292	0.9309	0.9326
$\omega = 8$	0.9182	0.9236	0.9331	0.9350	0.9366
$\omega = 12$	0.9222	0.9285	0.9333	0.9346	0.9380

(b) MNIST

Table 5.6: Effects of varying early and late exaggeration on nearest neighborhood purity

5.4.6 Observations

With the exception of very small step sizes and too few hash tables when using the hyperplane locality sensitive hashing family, our method appears fairly stable and yields embedding of fairly high quality.

The parameters η and the perplexity appear highly data dependent and thus our tuning attempt did not seem too successful. However, we will set them to the similar values between methods anyway to ensure comparability when possible, anyway.

Somewhat surprisingly, we observed that late exaggeration has a very strong positive effect on the nearest neighborhood purity, especially considering that it is only run in the last 10% (corresponding to 100 iterations) of gradient descent. It may be worthwhile to extend the duration of late exaggeration, perhaps even as far as [32] suggests to all iterations without early exaggeration.

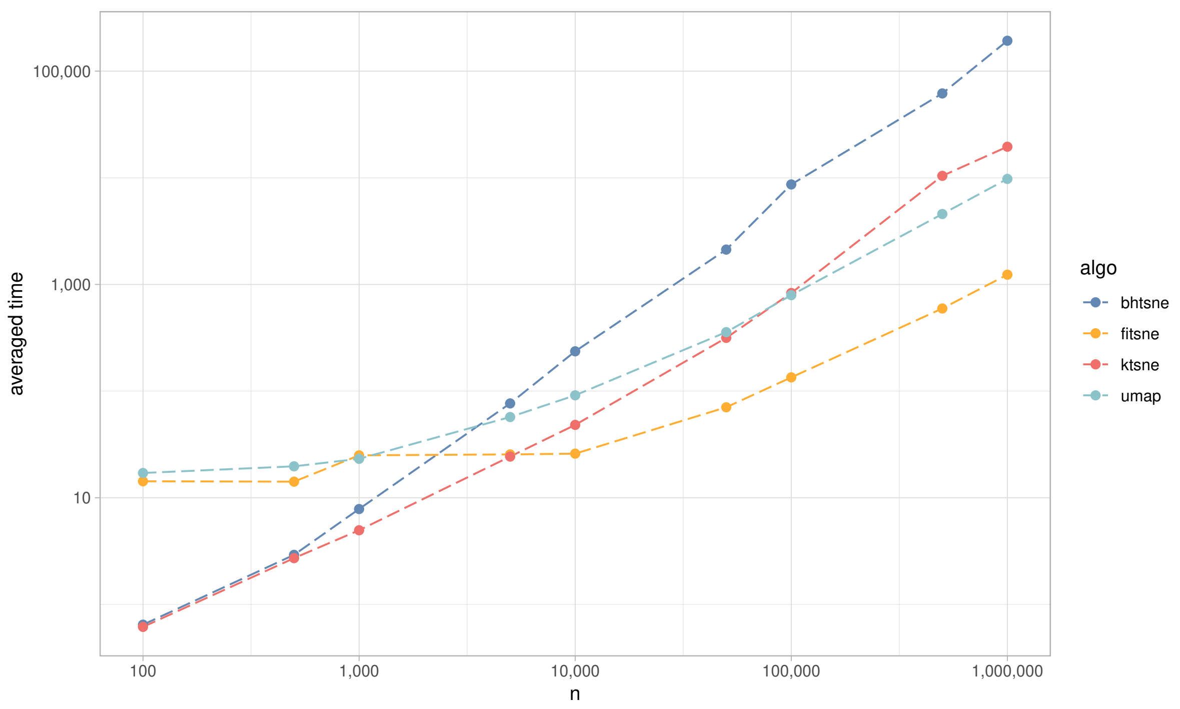


Figure 5.4: Runtime comparison on random data sets

5.5 Comparison Against Other Methods

We will now compare our method to 3 different related methods: Barnes-Hut t-SNE, FIt-SNE and UMAP. Note that Barnes-Hut t-SNE has a runtime complexity of $\mathcal{O}(n \log n)$ while the others, including our method, are linear.

Figure 5.4 shows the resulting curve obtained by running all methods on uniformly random data sets, averaged over 5 repeats. Unsurprisingly due to its higher runtime complexity, Barnes-Hut t-SNE is by far the worst method in this regard, followed by our method and UMAP, and finally FIt-SNE, which performs the best. Note that in the cases UMAP and FIt-SNE, multithreaded implementations and 4 threads were used. Barnes-Hut t-SNE and k-Means t-SNE are strictly singlethreaded.

Our method presents clearly linear and performs particularly well on very small data sets due to its low overhead caused by the lack of multithreading. This characteristic is shared by Barnes-Hut t-SNE. If an efficient way of parallelizing k-Means t-SNE is found, it may be able to compete with FIt-SNE.

Table 5.8 shows the results achieved on our benchmarking data sets, both measured by the nearest neighborhood purity and the runtime in seconds. The best result per data set is printed

	bhtsne	fitsne	umap	ktsne
$n = 100$	0.2424	0.2424	0.2424	0.2424
$n = 500$	1.0000	1.0000	1.0000	0.5371
$n = 1000$	1.0000	1.0000	1.0000	0.6018
$n = 5000$	1.0000	1.0000	1.0000	0.9203
$n = 10{,}000$	1.0000	1.0000	1.0000	0.9458
$n = 50{,}000$	1.0000	1.0000	1.0000	0.9761
$n = 100{,}000$	1.0000	1.0000	1.0000	0.9796
$n = 500{,}000$	1.0000	1.0000	1.0000	0.9921
$n = 1{,}000{,}000$	1.0000	1.0000	1.0000	0.9944

Table 5.7: Nearest neighborhood purity on the synthetic data set

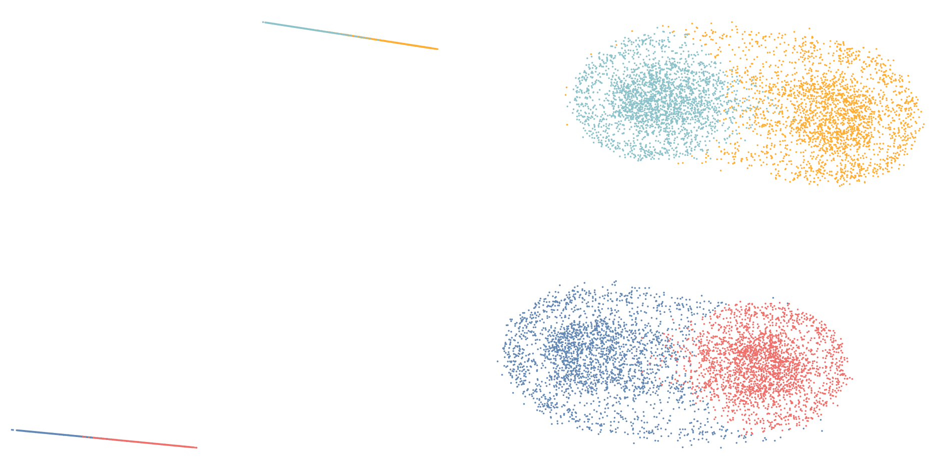
in bold. Data sets marked with an asterisk have labels generated by graph clustering. The runtime results are unsurprising given the previous figure—our methods excels at smaller data sets and is then superseded by FI*t*-SNE for larger data sets. FI*t*-SNE performs extraordinarily fast on large data sets and also achieves the best result purity-wise on 3 data sets. UMAP overall wins regarding the nearest neighborhood purity, followed by FI*t*-SNE, our method and finally Barnes-Hut *t*-SNE.

Overall, the methods perform similarly w.r.t. the nearest neighborhood purity, though there is a remarkable outlier for the emailEuCore data set where our method achieves a much lower nearest neighborhood purity of 0.07, compared to the purity values of over 0.3 for the other methods.

However similar, our method tends to achieve the worst results out of the four methods, which is likely due to the more crude approximation our methods performs. The fact that our method does not trail behind too far on all but one data set is promising as it shows that our method is practical enough to yield sensible, though not the best, embeddings. See Appendix C for images of selected embeddings.

5.5.1 Synthetic data

To further our comparison, we generated a simple data sets containing 4 clusters using the data generation package of ELKI [57]. The data is clearly clustered in all 50 dimensions. Table 5.7 shows the resulting neighborhood purity. Our method performs the worst, though its performance gets better as n increases. The bad performance at small n is likely due to the approximation error being especially prominent when there are only few points.



(a) Late exaggeration set to 4 (b) No late exaggeration

Figure 5.5: Results of our algorithm on the synthetic data set

Figure 5.5 shows the results of our algorithm for the synthetic data set with 10,000 points, with and without late exaggeration. Late exaggeration results in extremely compressed clusters, which is not visually appealing. In both cases our method failed to fully separate the clusters, while all other methods did so. This is likely due to k-Means consistently grouping points from both clusters together, making a separation difficult.

data	bhtsne	fitsne	umap	ktsne
emailEuCore	0.3127	0.3160	**0.3301**	0.0722
coil-20	0.5906	0.5641	**0.5920**	0.4922
cora	0.6331	0.6283	0.6232	**0.6404**
citeseer	0.4217	0.4262	0.4280	**0.4297**
optdigits	0.9677	0.9656	**0.9736**	0.9585
youtube*	0.7978	0.8011	**0.8124**	0.7946
dblp	0.3393	**0.3401**	0.3370	0.3100
pubmed	**0.7007**	0.6985	0.6986	0.6948
cifar	0.8784	0.8773	**0.8801**	0.8519
mnist	0.9536	**0.9539**	0.9521	0.9380
fashion_mnist	0.7417	**0.7426**	0.7034	0.6889
com-dblp*	0.5303	**0.5907**	0.5855	0.4580
com-amazon*	0.6982	0.7449	**0.7606**	0.6047
hollywood*	0.6695	0.7894	**0.8172**	0.7844
timit	0.3125	0.3445	**0.3641**	0.3481
svhn	0.1236	0.1238	**0.1240**	0.1234
wiki-topcats*	0.5060	0.7477	**0.8939**	0.7968

(a) Average nearest neighborhood purity

data	bhtsne	fitsne	umap	ktsne
emailEuCore	6.02	36.28	22.5	**4.54**
coil-20	9.74	52.67	24.16	**6.9**
cora	27.26	52.1	34.54	**12.04**
citeseer	44.52	56.02	39.49	**14.39**
optdigits	68.22	52.12	57.48	**27.46**
youtube	220.16	**50.52**	101.55	66.7
dblp	402.66	**53.22**	131.77	86.16
pubmed	478.61	**58.06**	142.53	100.36
cifar	1507.18	**70.11**	362.86	333.41
mnist	2284.13	**80.76**	512.21	466.5
fashion_mnist	2003.8	**80.63**	534.86	515.43
com-dblp	25,277.6	**345.12**	2336.02	4517.64
com-amazon	27,031.37	**360.54**	2486.53	5450.24
svhn	25,749.11	**656.79**	6191.86	10,212.89
hollywood	123,778.42	**964.93**	16,137.21	11,624.3
timit	190,110.22	**1146.17**	11,170.68	13,598.08
wiki-topcats	256,839.2	**1421.12**	17,196.37	22,242.01

(b) Summary of all runtimes per method, averaged

Table 5.8: Nearest neighborhood purity and runtime comparison of methods

6 Conclusion

This thesis introduced dimensionality reduction as a technique for experimental data analysis, with a particular focus on non-linear dimensionality reduction and t-SNE. We surveyed both linear and non-linear dimensionality reduction methods and outlined important methods in this field.

We then presented our algorithm which is a modification of the popular t-SNE. In our method, we made use of locality sensitive hashing to speed up the initial phase of finding (approximate) nearest neighbors in the high-dimensional space, and used k-Means, a simple, fast and well-known clustering method to summarize points to speed up the computation of the gradient—in particular, this step significantly accelerates the computation of the repulsive forces in comparison to the Barnes-Hut variant of t-SNE.

We further evaluated our method thoroughly, with a particular focus on finding parameters yielding a good nearest neighborhood purity based on two data instances. We want to note that this proved to be somewhat difficult and presented us with a few surprises that ran counter to our expectations. We then put our method to the test and let it compete against 3 other methods—the Barnes-Hut version, FIt-SNE, and UMAP—on 17 data sets. Though our algorithm did not perform better than the other methods, we did not fall behind too far and were able to compete with UMAP in regards to runtime.

6.1 Future Work

An important aspect that should be considered more to further the performance of our algorithm is parallelization. The most expensive step in our method is the computation of the distances to the centroids while performing k-Means. Note that k-Means is only run for 10 iterations, so coarse-grained parallelization does not seem to be a promising candidate.

Performance gains seem to be more likely with using fine-grained parallelization, e.g. SIMD instructions. We did attempt to leverage these technologies by using the Eigen matrix library, however unfortunately enabling various SIMD flags did not yield any improvements, and it seems that this is indeed a problem with Eigen. More work needs to be put into this and it will likely be necessary to re-write the performance-critical parts of code using SIMD-instructions.

More experiments pertaining to early and late exaggeration are also in order. While we focused on finding a one-size-fits-all exaggeration value in the parameter tuning section, which turned out to be quite large, it seems that such a value results in distorted embeddings particularly for small data sets. It would be interesting to possibly derive a rule of thumb for exaggeration based on perhaps the data size and general (expected) structure of the data.

Bibliography

[1] ANNOY library. `https://github.com/spotify/annoy`. Accessed: 2021-03-11.

[2] ASU Socialcomputing YouTube data set. http://socialcomputing.asu.edu/datasets/YouTube. Accessed: 2019-01-28, unfortunately now offline.

[3] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. Practical and optimal lsh for angular distance. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 1225–1233, Cambridge, MA, USA, 2015. MIT Press.

[4] Mukund Balasubramanian and Eric L. Schwartz. The isomap algorithm and topological stability. *Science*, 295(5552):7–7, 2002.

[5] Josh Barnes and Piet Hut. A hierarchical o(n log n) force-calculation algorithm. *Nature*, 324(6096):446–449, 1986.

[6] Etienne Becht, Leland McInnes, John Healy, Charles-Antoine Dutertre, Immanuel W. H. Kwok, Lai Guan Ng, Florent Ginhoux, and Evan W. Newell. Dimensionality reduction for visualizing single-cell data using umap. *Nature Biotechnology*, 37(1):38–44, 2019.

[7] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003.

[8] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[9] Sonja Biedermann, Monika Henzinger, Christian Schulz, and Bernhard Schuster. Memetic graph clustering. 02 2018.

[10] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[11] Ingwer Borg and Patrick Groenen. Modern multidimensional scaling: Theory and applications. *Journal of Educational Measurement*, 40:277 – 280, 06 2006.

[12] Emmanuel J. Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? *J. ACM*, 58(3):11:1–11:37, June 2011.

[13] Yanshuai Cao and Luyu Wang. Automatic selection of t-sne perplexity. *CoRR*, abs/1708.03229, 2017.

[14] Michael A. A. Cox and Trevor F. Cox. *Multidimensional Scaling*, pages 315–347. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[15] C. Croux, P. Filzmoser, and M.R. Oliveira. Algorithms for projection–pursuit robust principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 87(2):218 – 225, 2007.

[16] John P. Cunningham and Zoubin Ghahramani. Linear dimensionality reduction: Survey, insights, and generalizations. *J. Mach. Learn. Res.*, 16(1):2859–2900, January 2015.

[17] Sanjoy Dasgupta and Yoav Freund. Random projection trees and low dimensional manifolds. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 537–546, New York, NY, USA, 2008. ACM.

[18] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, SCG '04, page 253–262, New York, NY, USA, 2004. Association for Computing Machinery.

[19] Aman Dhesi and Purushottam Kar. Random projection trees revisited. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 496–504. Curran Associates, Inc., 2010.

[20] Chris Ding, Ding Zhou, Xiaofeng He, and Hongyuan Zha. R 1-pca: Rotational invariant l 1-norm principal component analysis for robust subspace factorization. *ICML 2006 - Proceedings of the 23rd International Conference on Machine Learning*, 2006:281–288, 01 2006.

[21] Wei Dong, Charikar Moses, and Kai Li. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th International Conference on World Wide Web*, WWW '11, pages 577–586, New York, NY, USA, 2011. ACM.

[22] David Donoho and Carrie Grimes. Hessian eigenmaps: Locally linear embedding techniques for high-dimensional data. proc. national academy of science (pnas), 100, 5591-5596. *Proceedings of the National Academy of Sciences of the United States of America*, 100:5591–6, 06 2003.

[23] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017.

[24] Keinosuke Fukunaga. *Introduction to Statistical Pattern Recognition (2Nd Ed.)*. Academic Press Professional, Inc., San Diego, CA, USA, 1990.

[25] Jacqueline S. Galpin and Douglas M. Hawkins. Methods of l1-estimation of a covariance matrix. *Comput. Stat. Data Anal.*, 5(4):305–319, September 1987.

[26] Junbin Gao. Robust l1 principal component analysis and its bayesian variational inference. *Neural Comput.*, 20(2):555–572, February 2008.

[27] John Garofolo, Lori Lamel, William Fisher, Jonathan Fiscus, David Pallett, Nancy Dahlgren, and Victor Zue. TIMIT Acoustic-Phonetic Continuous Speech Corpus, 1993.

[28] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[29] Geoffrey Hinton and Sam Roweis. Stochastic neighbor embedding. *Advances in neural information processing systems*, 15:833–840, 2003.

[30] Chenping Hou, Yuanyuan Jiao, Yi Wu, and Dongyun Yi. Relaxed maximum-variance unfolding. *Optical Engineering - OPT ENG*, 47, 07 2008.

[31] Odest Jenkins and Maja Mataric. Deriving action and behavior primitives from human motion data. volume 3, pages 2551 – 2556 vol.3, 02 2002.

[32] D. Kobak and Philipp Berens. The art of using t-sne for single-cell transcriptomics. *Nature Communications*, 10, 2019.

[33] Dmitry Kobak and Philipp Berens. The art of using t-sne for single-cell transcriptomics. *bioRxiv*, 2019.

[34] Dmitry Kobak and George C. Linderman. Umap does not preserve global structure any better than t-sne when using the same initialization. *bioRxiv*, 2019.

[35] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[36] J. B. Kruskal. Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika*, 29(1):1–27, Mar 1964.

[37] Yann LeCun and Corinna Cortes. MNIST handwritten digit database. 2010.

[38] John Aldo Lee and Michel Verleysen. Nonlinear dimensionality reduction of data manifolds with essential loops. *Neurocomputing*, 67:29 – 53, 2005. Geometrical Methods in Neural Networks and Learning.

[39] Jure Leskovec and Andrej Krevl. SNAP Datasets: Stanford large network dataset collection. `http://snap.stanford.edu/data`, June 2014.

[40] Guoying Li and Zhonglian Chen. Projection-pursuit approach to robust dispersion matrices and principal components: Primary theory and monte carlo. *Journal of the American Statistical Association*, 80(391):759–766, 1985.

[41] Ik Soo Lim, Pablo de Heras Ciechomski, Sofiane Sarni, and Daniel Thalmann. Planar arrangement of high-dimensional biomedical data sets by isomap coordinates. pages 50–55, 07 2003.

[42] George C. Linderman, Manas Rachh, Jeremy G. Hoskins, Stefan Steinerberger, and Yuval Kluger. Efficient algorithms for t-distributed stochastic neighborhood embedding. *CoRR*, abs/1712.09005, 2017.

[43] Ting Liu, Andrew W. Moore, and Alexander Gray. New algorithms for efficient high-dimensional nonparametric classification. *J. Mach. Learn. Res.*, 7:1135–1158, December 2006.

[44] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. Multi-probe lsh: Efficient indexing for high-dimensional similarity search . pages 950–961, 01 2007.

[45] Leland McInnes, John Healy, and James Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *arXiv e-prints*, page arXiv:1802.03426, Feb 2018.

[46] Sameer A. Nene, Shree K. Nayar, and Hiroshi Murase. Columbia object image library (coil-20. Technical report, 1996.

[47] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Ng. Reading digits in natural images with unsupervised feature learning. *NIPS*, 01 2011.

[48] Loïc Paulevé, Hervé Jégou, and Laurent Amsaleg. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348 – 1358, 2010.

[49] Karl Pearson. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901.

[50] Claudia Plant, Sonja Biedermann, and Christian Böhm. Data compression as a comprehensive framework for graph drawing and representation learning. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '20, page 1212–1222, New York, NY, USA, 2020. Association for Computing Machinery.

[51] Anand Rajaraman, Jure Leskovec, and Jeffrey D. Ullman. *Mining Massive Datasets.* 2014.

[52] Parikshit Ram and Kaushik Sinha. Revisiting kd-tree for nearest neighbor search. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 1378–1388, New York, NY, USA, 2019. ACM.

[53] Ryan A. Rossi and Nesreen K. Ahmed. The network data repository with interactive graph analytics and visualization. In *AAAI*, 2015.

[54] Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, 2000.

[55] J. W. Sammon. A nonlinear mapping for data structure analysis. *IEEE Trans. Comput.*, 18(5):401–409, May 1969.

[56] Lawrence Saul and Sam Roweis. An introduction to locally linear embedding. *Journal of Machine Learning Research*, 7, 01 2001.

[57] Erich Schubert and Arthur Zimek. ELKI: A large open-source library for data analysis - ELKI release 0.7.5 "heidelberg". *CoRR*, abs/1902.03616, 2019.

[58] John Shawe-Taylor and Nello Cristianini. *Kernel Methods for Pattern Analysis.* Cambridge University Press, New York, NY, USA, 2004.

[59] Jonathon Shlens. A tutorial on principal component analysis. *CoRR*, abs/1404.1100, 2014.

[60] Jian Tang, Jingzhou Liu, Ming Zhang, and Qiaozhu Mei. Visualization large-scale and high-dimensional data. *CoRR*, abs/1602.00370, 2016.

[61] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[62] M. E. Tipping and Christopher Bishop. Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B*, 21(3):611–622, January 1999. Available from http://www.ncrg.aston.ac.uk/Papers/index.html.

[63] Warren S. Torgerson. Multidimensional scaling: I. theory and method. *Psychometrika*, 17(4):401–419, Dec 1952.

[64] Anton Tsitsulin, Davide Mottin, Panagiotis Karras, and Emmanuel Müller. VERSE: versatile graph embeddings from similarity measures. *CoRR*, abs/1803.04742, 2018.

[65] Laurens van der Maaten. Accelerating t-sne using tree-based algorithms. *Journal of Machine Learning Research*, 15:3221–3245, 2014.

[66] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[67] Laurens Van Der Maaten, Eric Postma, and Jaap Van den Herik. Dimensionality reduction: a comparative review. *J Mach Learn Res*, 10:66–71, 2009.

[68] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. How to use t-sne effectively. *Distill*, 2016.

[69] Kilian Q. Weinberger and Lawrence K. Saul. Unsupervised learning of image manifolds by semidefinite programming. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'04, pages 988–995, Washington, DC, USA, 2004. IEEE Computer Society.

[70] Killan Q. Weinberger and Lawrence K. Saul. An introduction to nonlinear dimensionality reduction by maximum variance unfolding. In *Proceedings of the 21st National Conference on Artificial Intelligence - Volume 2*, AAAI'06, pages 1683–1686. AAAI Press, 2006.

[71] Christopher K. I. Williams. On a connection between kernel pca and metric multidimensional scaling. *Mach. Learn.*, 46(1-3):11–19, March 2002.

[72] Han Xiao, Kashif Rasul, and Roland Vollgraf. Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms. *CoRR*, abs/1708.07747, 2017.

[73] Zhenyue Zhang and Jing Wang. Mlle: Modified locally linear embedding using multiple weights. In B. Schölkopf, J. C. Platt, and T. Hoffman, editors, *Advances in Neural Information Processing Systems 19*, pages 1593–1600. MIT Press, 2007.

[74] Zhenyue Zhang and Hongyuan Zha. Principal manifolds and nonlinear dimension reduction via local tangent space alignment. *CoRR*, cs.LG/0212008, 2002.

Appendix A

Implementation

Our method is written in C++17 and uses the most recent version of FALCONN[3] and Eigen[4] 3.3.4.

The input must be a CSV (i.e. comma-separated) file and may or may not have a header consisting of alphanumeric column names. The following listing shows the helptext of the program and lists the default parameters.

Listing A.1: Helptext of our implementation

```
usage: ./ktsne opts FILE
where opt in opts is one of the following:

  -p                        ... perplexity (effective number of neighbors per point).
    ↪ tunable parameter, default = 30
  -n                        ... stepsize eta of gradient descent, default = 200
  -x, --early-exaggeration ... early exaggeration value, default = 12
  -X, --late-exaggeration  ... late exaggeration value, default = 12
  -i, --max-iter           ... number of gradient descent iterations, default = 1000
  -s, --seed               ... random seed

  -k, --k-lo ... lower bound for k-means k, default = 30
  -K, --k-hi ... upper bound for k-means k, default = 30

  --random-init ... initialize using a Gaussian distribution with stddev 10e-4
  --pca-init    ... initialize with PCA
```

[3] https://falconn-lib.org/
[4] https://eigen.tuxfamily.org

```
  --num-hash-tables ... number of hash tables for FALCONN lsh. tunable parameter,
      ↪ default = 100
  --num-hash-bits   ... number of hash bits, controls number of buckets per table.
      ↪ automatically set to max(16, log2(n)) if -1 is passed, default = -1
  --num-probes      ... number of probes for multi-probe LSH. tunable parameter (
      ↪ inverse relation to L), default = -1

  --[no-]compute-objective  ... compute objective in every iteration, default = off
  --[no-]print-intermediate ... print intermediate embedding to file every iteration
      ↪ (for creating GIFs), default = off

  --use-hyperplane          ... use hyperplane LSH (instead of cross-polytope),
      ↪ default = off
  --use-cross-polytope      ... use cross-polytope LSH (instead of hyperplane LSH),
      ↪ default = on

  -h ... this message

and FILE is a csv file.

ktsne is an accelerated approximative version of tsne which uses LSH and kmeans in
    ↪ its computation.
```

Upon starting, the program prints an overview of all parameter settings and then prints the iteration number every 100 iterations to communicate progress. Per default, the objective is not printed because its computation is tedious and the approximated objective is not very meaningful (see Chapter 5).

Appendix B

Squared Distance Computation Performance Experiment

An important part of the computation of the gradient in our method, as well as all other variants of t-SNE, are the pairwise squared Euclidean distances between all points in the low-dimensional space. We attempted to optimize this process and compared three different approaches using Eigen. The first is a naïve implementation:

Listing B.1: Naïve approach

```
Eigen::MatrixXd D(X.rows(), Y.rows());
for(size_t i = 0; i < X.rows(); ++i) {
    for(size_t j = 0; j < Y.rows(); ++j) {
        D(i, j) = (X.row(i) - Y.row(j)).squaredNorm();
    }
}
```

This approach explicitly computes all $n \times m$ pairs. The second approach uses Eigen's internal iteration mechanism by making use of `rowwise`, but still uses external iteration. Using internal iteration should result in a speedup due to optimizations internal to Eigen.

Listing B.2: Rowwise approach

```
Eigen::MatrixXd D(X.rows(), Y.rows());
for (int i = 0; i < Y.rows(); i++)
    D.col(i) = (X.rowwise() - Y.row(i)).rowwise().squaredNorm().transpose();
```

The third variant only uses internal iteration and is the most complex. We applied the binomial theorem $(a-b)^2 = a^2 - 2ab + b^2$. This has the presumptive advantage that we can offload a lot of the computational effort onto Eigen's matrix-matrix multiplication, which should result into a considerable speedup.

Listing B.3: Binomial theorem approach

```
Eigen::MatrixXd D(X.rows(), Y.rows());
```

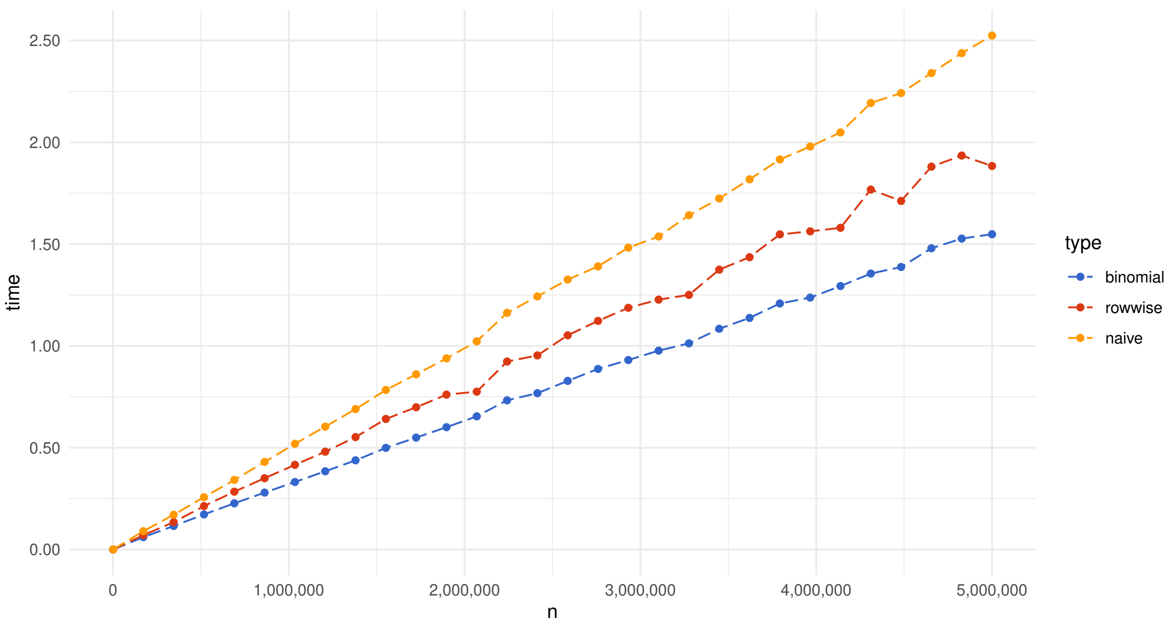


Figure B.1: Comparison in runtime between 3 approaches to computing all-pairs squared Euclidean distances

```
D = ((X * Y.transpose() * -2).colwise() + X.rowwise().squaredNorm()).rowwise() +
    ↪ Y.rowwise().squaredNorm().transpose();
```

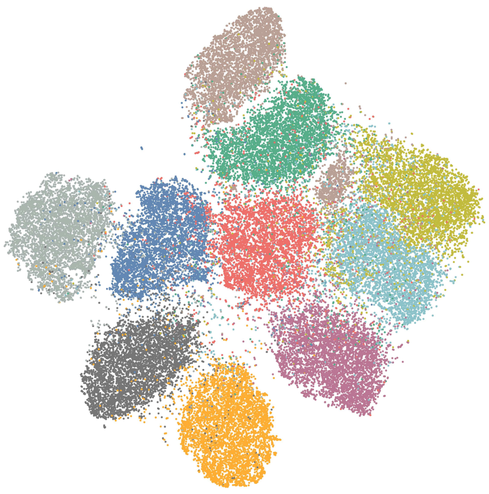
Figure B.1 shows the runtime comparison between the three aforementioned approaches. We ran 10 repeats for every problem size n. The matrices X and Y were of dimensions $n \times 2$ and 50×2, which corresponds to the use case in our method. It shows that the third variant performs well and the gaps between the methods keep widening as n increases. We thus used the binomial form in our implementation.

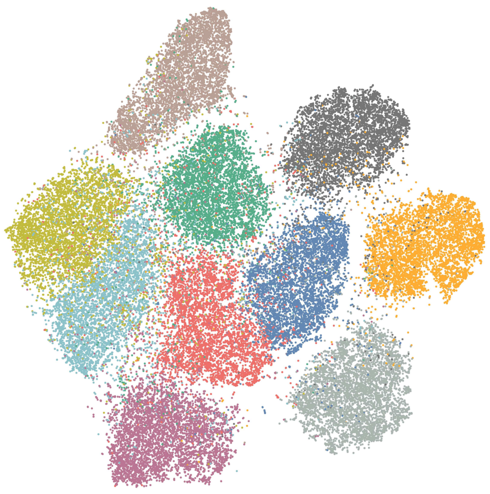
Appendix C

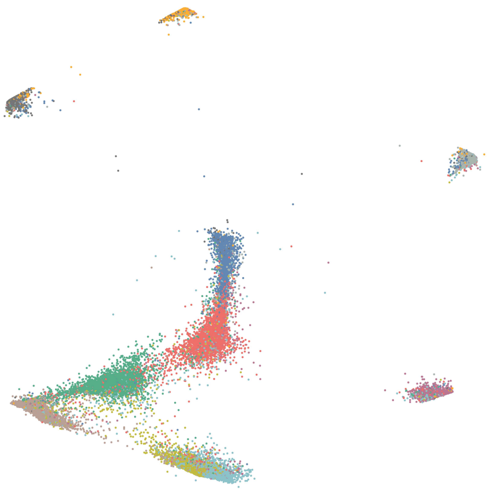
Selected Embedding Comparison

On the next pages, we present the embeddings of 6 selected datasets—cifar, com-amazon, fashion MNIST, pubmed, hollywood and wiki-topcats—found by the 4 methods compared in our work.

As can be seen, our method tends to compute embeddings that appear somewhat distorted and not as spherical especially when compared to FI*t*-SNE and Barnes-Hut *t*-SNE. This may be due to the large exaggeration values used or due to the approximation error. It is notable, though, that the distortion is also present in UMAP's embeddings. Overall, all embeddings appear sensible. Do note that the labels for com-amazon and wiki-topcats were generated using graph clustering and indeed match the embeddings well.


(a) Barnes-Hut *t*-SNE


(b) FI*t*-SNE


(c) *kt*-SNE

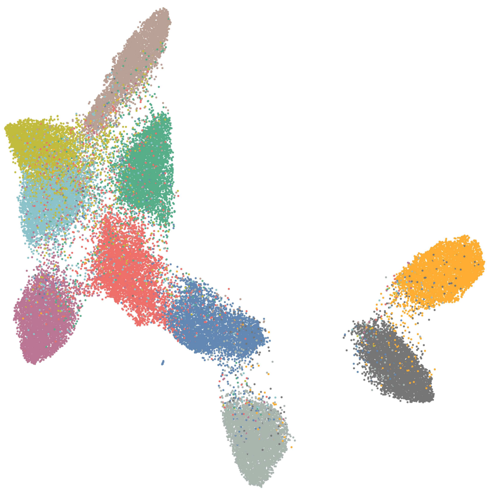
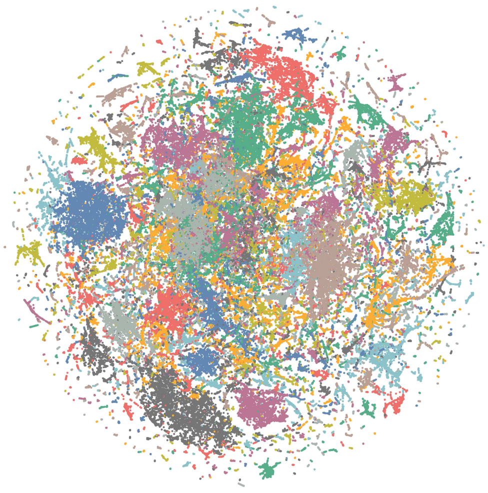
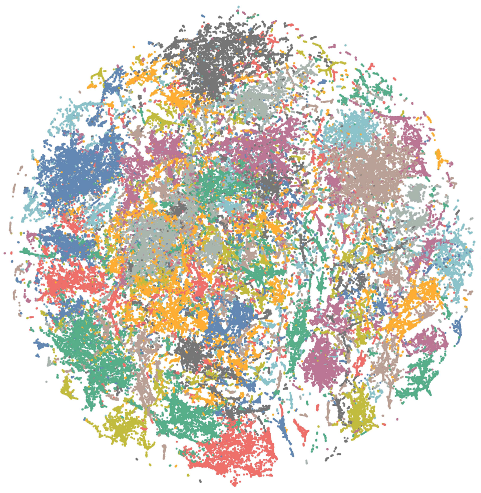
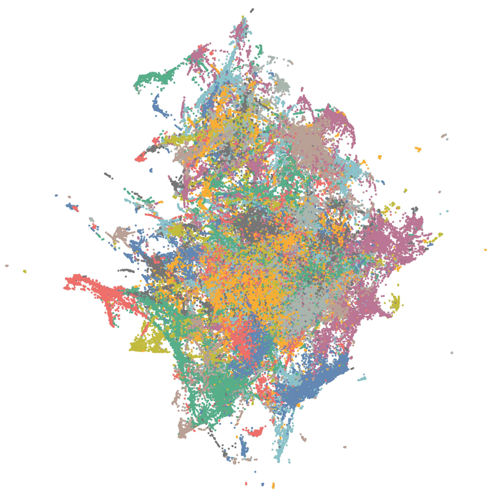

(d) UMAP

Figure C.1: CIFAR


(a) Barnes-Hut *t*-SNE


(b) FI*t*-SNE


(c) *kt*-SNE

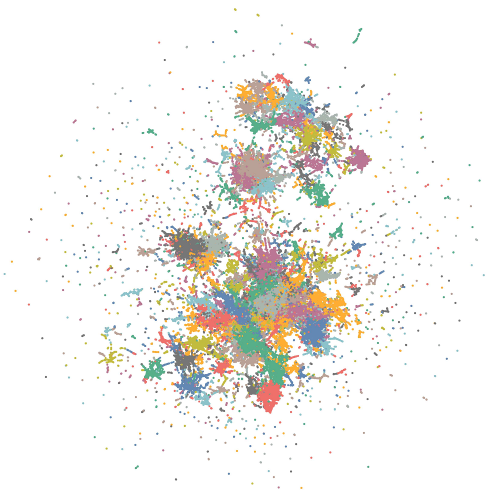
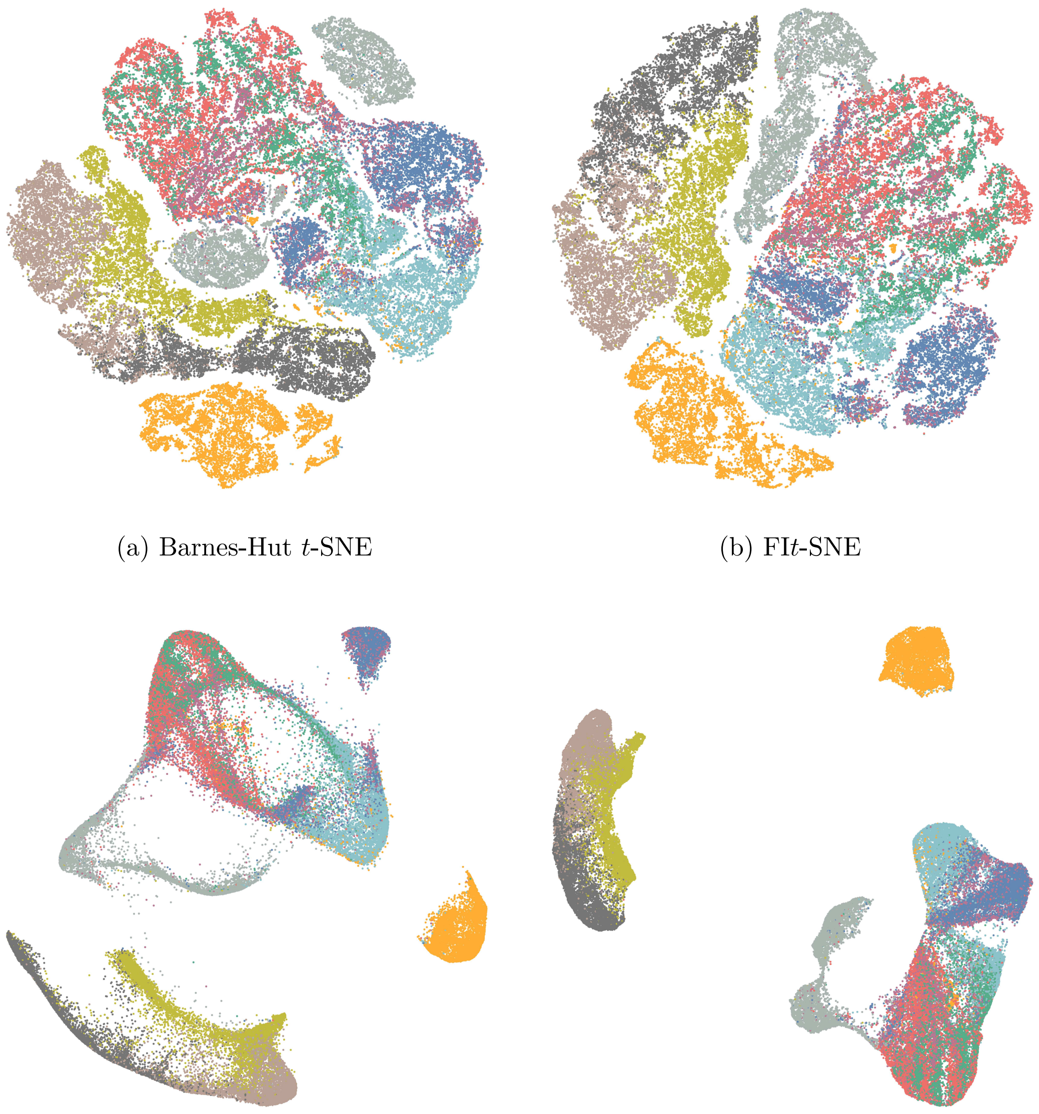

(d) UMAP

Figure C.2: com-amazon



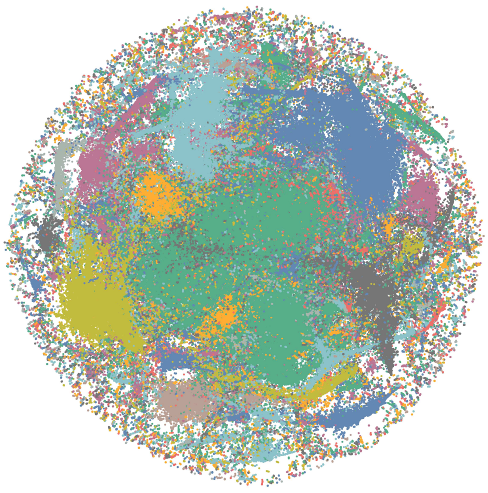
(a) Barnes-Hut t-SNE

(b) FIt-SNE

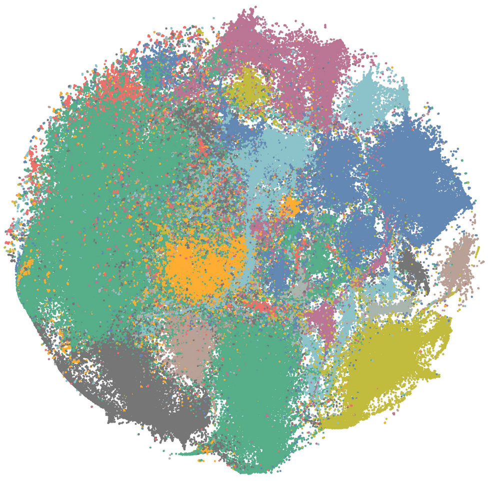
(c) kt-SNE

(d) UMAP

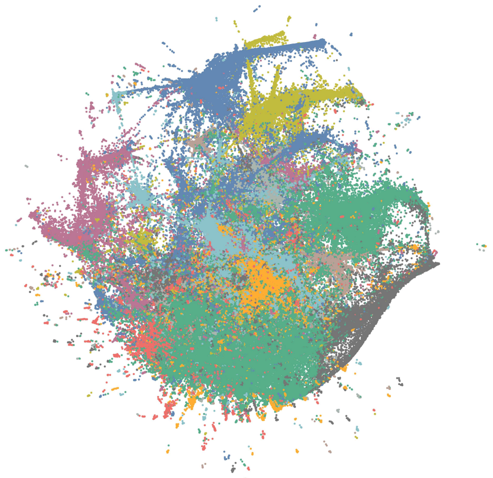
Figure C.3: Fashion MNIST



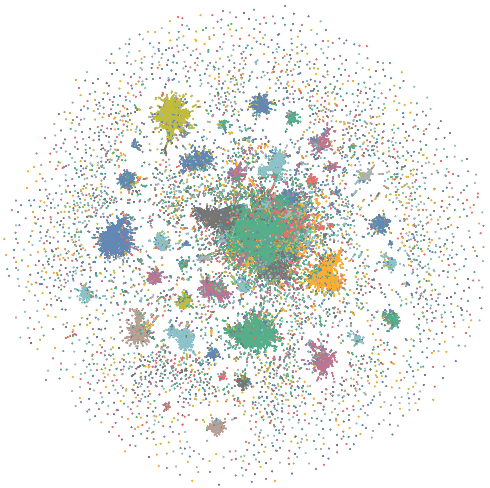
(a) Barnes-Hut *t*-SNE



(b) FI*t*-SNE

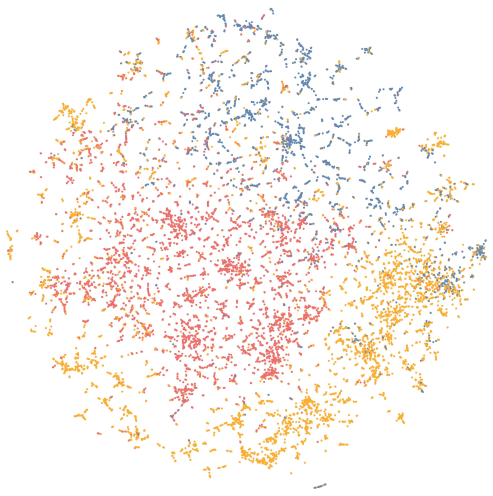


(c) *kt*-SNE

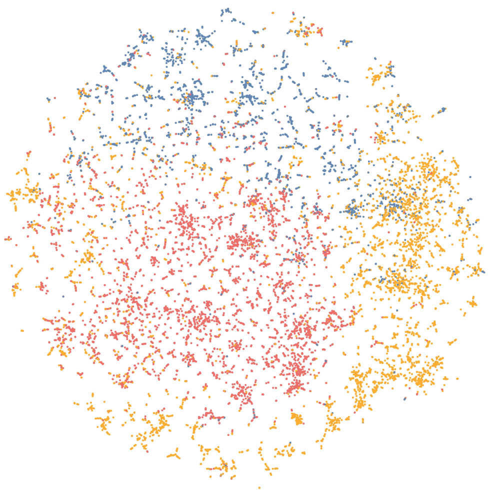


(d) UMAP

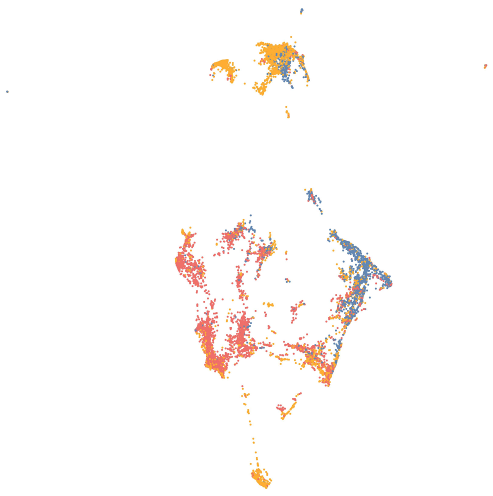
Figure C.4: hollywood



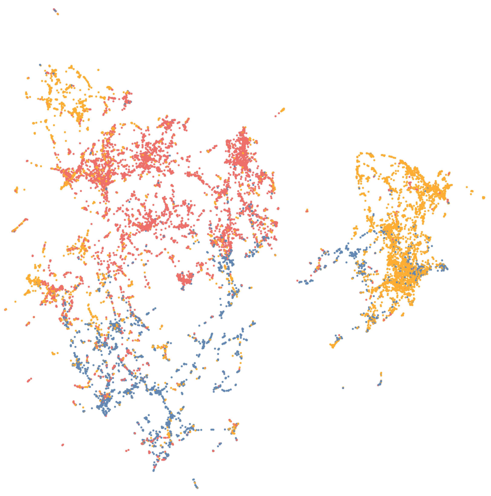
(a) Barnes-Hut *t*-SNE



(b) FI*t*-SNE

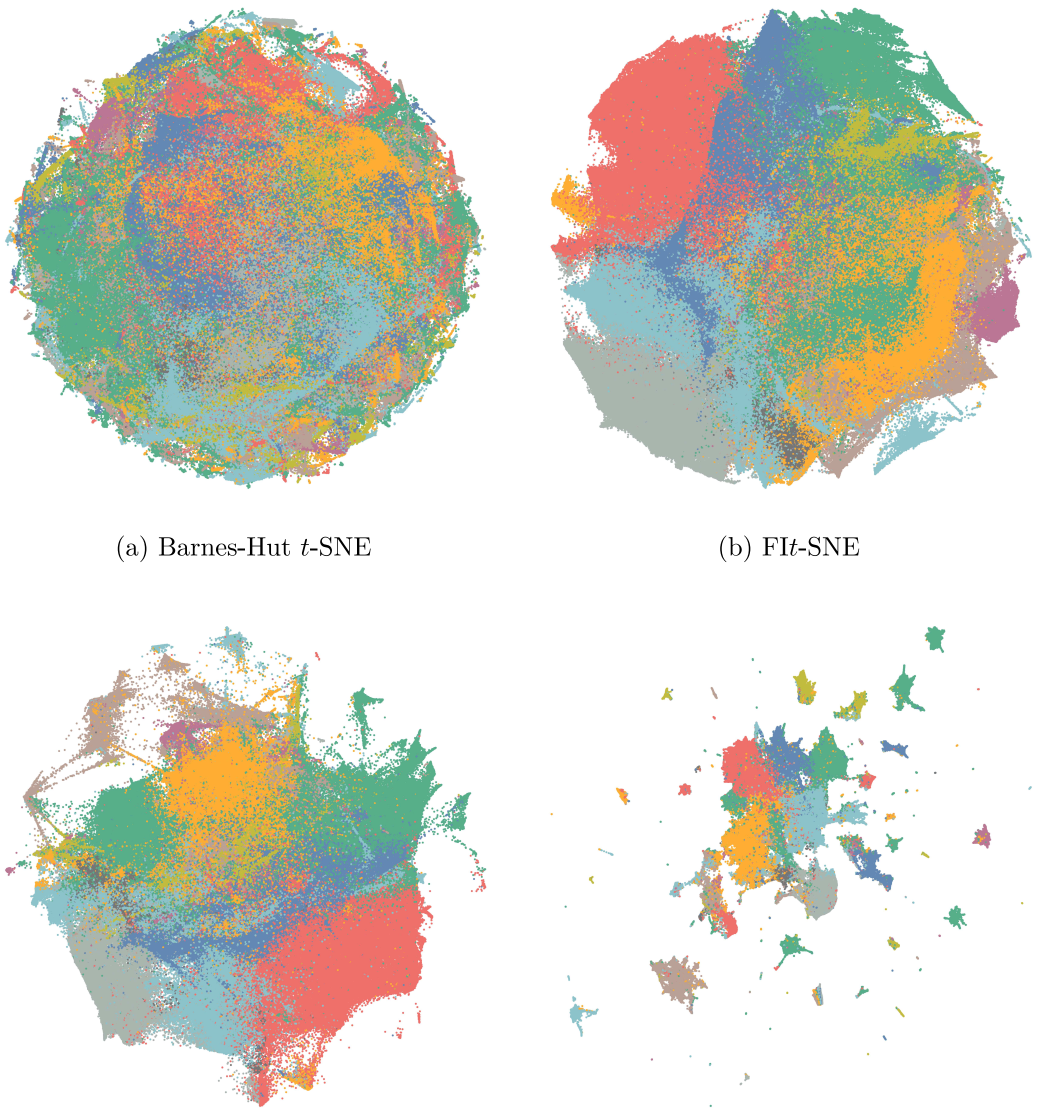


(c) *kt*-SNE



(d) UMAP

Figure C.5: pubmed



(a) Barnes-Hut t-SNE

(b) FIt-SNE

(c) kt-SNE

(d) UMAP

Figure C.6: wiki-topcats