MONASH University

MONASH BUSINESS SCHOOL

ISSN 1440-771X

Department of Econometrics and Business Statistics

http://monash.edu/business/ebs/research/publications

Manifold learning with approximate nearest neighbors

Fan Cheng, Rob J Hyndman, Anastasios Panagiotelis

August 2021

Working Paper 03/2021

Manifold learning with approximate nearest neighbors

Fan Cheng
Monash University
Email: Fan.Cheng@monash.edu
Corresponding author

Rob J Hyndman
Monash University
Email: Rob.Hyndman@monash.edu

Anastasios Panagiotelis
University of Sydney
Email: Anastasios.Panagiotelis@sydney.edu.au

31 August 2021

JEL classification: C55, C65, C80

Manifold learning with approximate nearest neighbors

Abstract

Analyzing high-dimensional with manifold learning algorithms often requires searching for the nearest neighbors of all observations. This presents a computational bottleneck when data size is large or when observations lie in more general metric spaces, such as statistical manifolds. We resolve this problem by proposing a broad range of approximate nearest neighbor (ANN) methods to be used within manifold learning. The novelty of our evaluation of ANN algorithms is the manifold learning setting, with algorithms compared on the basis of embedding accuracy. A second novel contribution is to use ANN for statistical manifolds by exploiting the connection between Hellinger/Total variation distance for discrete distributions and the L2/L1 norm. A thorough empirical investigation of the benchmark MNIST dataset shows that ANN algorithms substantially improve computational time with little to no loss in the accuracy of manifold learning embedding. The result is robust to different manifold learning algorithms, different approximate nearest neighbor algorithms, and different measures of embedding accuracy. The proposed method is applied to learning statistical manifolds of electricity usage. This application demonstrates how underlying structures in high dimensional data, including anomalies, can be visualized and identified, in a way that is scalable to large datasets.

Keywords: statistical manifold, dimension reduction, anomaly detection, k-d trees, Hellinger distance, smart meter data

1 Introduction

Modern data often comprise a large number of high-dimensional observations in a possibly non-Euclidean space. Dimension reduction can be an important tool for exploring and analyzing such data for tasks such as clustering, classification, visualization, and anomaly detection. If high-dimensional data are assumed to lie on a lower-dimensional manifold, then manifold learning algorithms (see Cayton (2005), Lee & Verleysen (2007) and Izenman (2012) for reviews) can be used to extract a low dimensional representation of the data. Applications of manifold learning algorithms include analyzing cell characteristics in clinical cell cytometry (Carter et al. 2009), classification and visualization in hyperspectral image analysis (Lunga et al. 2014), image reconstruction during an MRI acquisition (Zhu et al. 2018), proteomic signaling network analysis in cancer (Banerjee, Akbani & Baladandayuthapani 2019), and detecting anomalous probability distributions of household electricity usage (Hyndman, Liu & Pinson 2018). The last of these examples pose a number of challenges that motivate two novel contributions proposed in this manuscript. The first is to consider the use of a wide range of contemporary approximate nearest neighbor methods, as opposed to exact nearest neighbors within manifold learning algorithms, and to thoroughly investigate the effect of this approximation on the accuracy of manifold learning. The second is to incorporate approximate nearest neighbor algorithms into manifold learning for statistical manifolds - that is a manifold whose elements are probability distributions. This is achieved by finding discrete approximations to each distribution and then exploiting the link between Hellinger (or Total Variation) distance and the L2 (or L1) norm.

Linear methods for dimension reduction date at least as far back as the development of Principal Components Analysis, while a number of non-linear methods were developed in the 1960s (Shepard 1962a,b; Kruskal 1964a,b). A further flourishing in the development of non-linear dimension reduction techniques gained traction after ISOMAP (Tenenbaum, Silva & Langford 2000) and Local Linear Embedding (LLE) (Roweis & Saul 2000) were introduced in the same issue of *Science*. These methods assume data lie on a manifold and are hence known as manifold learning techniques. Some algorithms developed in the early 2000s include, but are not limited to, Laplacian eigenmaps (Belkin & Niyogi 2003), Hessian LLE (Donoho & Grimes 2003), local tangent space alignment (Zhang & Zha 2003), diffusion maps (Nadler et al. 2006; Coifman & Lafon 2006), semi-definite embedding (Weinberger & Saul 2006). More contemporary algorithms including t-SNE (Van der Maaten & Hinton 2008) and UMAP(McInnes, Healy & Melville 2018), have been shown to be well-suited to the visualization of many real-world datasets. While these algorithms all differ, most contain a common step which is to compute a graph of K-nearest neighbors. These include ISOMAP, LLE, Laplacian

Eigenmaps, Hessian LLE, t-SNE, and UMAP, which are the six manifold learning algorithms we consider in this paper.

Computing K nearest neighbors can be a significant computational bottleneck especially when the number of observations is large. A naive way to find the nearest neighbors is to compute pairwise distances between all observations - an $O(N^2)$ operation. For some metrics including Euclidean and Manhattan distance, more efficient solutions that find the K nearest neighbors in $O(Nlog(N))$ time are available, for example, k-d trees (Bentley 1975). Some software implementations of manifold learning techniques, including the R package *dimRed* (Kraemer, Reichstein & Mahecha 2018) exploit such algorithms. For applications with an extremely large number of observations, faster approximate versions of k-d trees can be used for nearest neighbor search (Van Der Maaten 2014). McQueen et al. (2016) provides one such implementation through the Python package, *megaman*, which exploits the *FLANN* (Muja & Lowe 2009) package for fast neighbor searching[1]. However, to the best of our knowledge, no work has been done on evaluating the effect of using approximate nearest neighbors on the accuracy of embeddings produces by manifold learning algorithms. Furthermore, more recent research on approximate nearest neighbor search has seen alternative and arguably more efficient alternatives to k-d trees emerge, including Annoy (Spotify 2016) and Hierarchical Navigable Small Worlds (HNSW) (Malkov & Yashunin 2020). Consequently, the first main novel contribution in this paper is to thoroughly evaluate the impact of using approximate nearest neighbor search within manifold learning and to compare a broad range of approximate nearest neighbors in doing so.

We evaluate the impact of using approximate nearest neighbor search in manifold learning in two main ways. First, it is important to establish that using ANN algorithms improves, in a substantive manner, the speed of manifold learning algorithms. Via a thorough empirical study using the benchmark MNIST dataset, we establish that in practice, up to a four-fold improvement in computational time can be achieved when using ANN, relative to exact nearest neighbors even if k-d trees are used for the exact solution. The improvements in computational speed are greatest for the Annoy algorithm and for Laplacian Eigenmaps or UMAP. Second, it is important to establish that using ANN algorithms does not lead to a substantial deterioration in the accuracy of embeddings produced by manifold learning algorithms. UMAP (McInnes, Healy & Melville 2018) and variations of t-SNE (Van Der Maaten 2014; Tang et al. 2016) try to accelerate the nearest neighbor searching process using a tree-based method or NN-Descent algorithm (Dong, Moses & Li 2011), but the evaluations are limited to the accuracy of the neighborhood graph. Instead, the low-dimensional embeddings, which is the final output of manifold learning, tend to be tricky to explain and the

[1] approximate nearest neighbors using k-d trees can be implemented in R using the C++ ANN library (Mount & Arya 2010) is wrapped in the R package *RANN* (Mount et al. 2019)

topological information can be inaccurate. Therefore, we use multiple embedding quality measures in Section 2.4 for the evaluation of manifold learning accuracy. Again using the benchmark MNIST data, we find that using ANN instead of exact nearest neighbors leads to an almost negligible reduction in embedding accuracy exceeding 5% in only a small number of cases. The impact of using ANN on embedding accuracy is much smaller than the impact of choosing a different manifold learning algorithm. These results are robust to different choices of ANN algorithm and to a range of measures of embedding accuracy.

The second main contribution of this paper is to find a way to combine efficient exact and approximate nearest neighbor algorithms with manifold learning of statistical manifolds (for a general overview of statistical manifolds see Amari (2016)). Previously, Lee, Abbott & Araman (2007) parameterize discrete probability mass functions as points on a hypersphere but do not carry out nonlinear dimension reduction, concluding that *"it is unrealistic in terms of speed to use algorithms with a complexity of $O(N^2)$ when N is large"*. Recognizing that the approach of Lee, Abbott & Araman (2007) may fail when the statistical manifold in question lies on a submanifold of the hypersphere, Carter et al. (2009) propose manifold learning using the Fisher information metric. however, their method for computing this metric can be computationally expensive. Finally, Hyndman, Liu & Pinson (2018) compute the Jensen Shannon distance between two density estimates and then applied Laplacian Eigenmaps using the very same dataset we consider in Section 4. In all three of these papers, all N^2 pairwise distances are computed making them infeasible for applications with over a few thousand observations.

To overcome this issue we propose the following approach. First, in a similar fashion to Lee, Abbott & Araman (2007), we consider discrete-domain distributions. These are approximated for each element of the statistical manifold and the values of the probability mass functions (or transformations thereof) are stacked into vectors. We then propose using the Hellinger distance or Total Variation distance between these discrete approximations as the input metric for manifold learning. By exploiting a connection between the Hellinger and Total Variation distance and the L2 and L1 norm respectively, we are able to find nearest neighbors using k-d trees which has a computational complexity of $O(Nlog(N))$ rather than $O(N^2)$. In principle, even further speed up may be achieved with approximate nearest neighbor algorithms. To the best of our knowledge, this approach has not been previously proposed in the literature and represents the second main novel contribution of our paper. While we acknowledge the shortcomings of Hellinger and Total Variation metric compared to the Fisher information metric as used by Carter et al. (2009), we note that Hellinger and Total Variation metric approximate Fisher information metric well locally, i.e. when the distance between two probability distributions is small. This is precisely the situation likely to be encountered when N is large and the statistical manifold is more densely sampled.

We demonstrate the potential of our method with an application to smart meter data where the main objective of the analysis is visualization and anomaly detection. We show that manifold learning algorithms can be implemented quickly using k-d trees. The exact version of k-d trees can even be faster than ANN when the number of grid points used in the discrete approximation is high. For a fixed manifold learning algorithm, the effect of using ANN on the accuracy of the embedding and the identified anomalies is minor. Low dimensional visualizations identify structure in the data, whether that be in the time of week during which electricity is used, or in the way that anomalous households from far away regions of the embedding are very different from one another.

The rest of the paper is organized as follows. Section 2 serves as a primer defining the algorithms and measures used throughout the paper in detail and with consistent notation. It is composed of three subsections; the first deals with different manifold learning algorithms, the second with approximate nearest neighbor methods, and the third with embedding quality measures. Readers familiar with one or more of these topics may comfortably skip the corresponding subsections. Section 3 applies different techniques for manifold learning with ANN to the benchmark MNIST dataset and evaluates the results with respect to computational time and embedding accuracy. Section 4 contains the application to visualizing and identifying anomalies in household electricity usage using Irish smart meter data. In this section, we provide further justification for the use of ANN with manifold learning, including in the case where the distance between observations is either the Hellinger of Total Variation metric between two distributions of electricity usage. We provide some discussion and conclusions in Section 5.

2 Manifold learning with approximate nearest neighbors

2.1 Notation

Manifold learning finds a d-dimensional representation of data that lie on a manifold $\mathcal{M}$ embedded in a p-dimensional ambient space with $d \ll p$. We will denote the original data (or 'input' points) as $x_i, i = 1, \ldots, N$, where $x_i \in \mathbb{R}^p$, while the low-dimensional representation (or 'output' points) will be denoted as $y_i, i = 1, \ldots, N$, where $y_i \in \mathbb{R}^d$. Where two subscripts are used (e.g. x_{ih} or y_{ih}), the second subscript refers to the h^{th} coordinate or dimension of the data. Pairwise distances between input points x_i and x_j are denoted δ_{ij} while pairwise distances between output points y_i and y_j are denoted d_{ij}, where $i, j = 1, \ldots, N$. Unless otherwise stated, these distances are assumed to be Euclidean. Also important is the K-ary neighborhoods (or K nearest neighbors) of x_i denoted $U_K(i)$ and defined as a set of points j such that x_j is one of the K or less closest points to x_i.

2.2 Manifold learning algorithms

We now briefly describe the manifold learning algorithms used in the remainder of the paper. One feature shared by all algorithms discussed below is that they require K nearest neighbors to be found for all input points. In all implementations of manifold learning algorithms in Sections 3 and 4, K is set to 20 for comparison across different methods. Other parameters of the manifold learning algorithms, such as the intrinsic dimension of the manifold (Denti et al. 2021), are also essential to the complexity of the data and the embeddings. For visualization purposes, the dimension d is set as 2 in this paper.

ISOMAP

ISOMAP (Tenenbaum, Silva & Langford 2000), short for isometric feature mapping, was one of the first algorithms introduced for manifold learning as a non-linear extension to classical Multidimensional Scaling (MDS) and is described in Algorithm 1. Classical MDS uses the spectral decomposition to find an embedding such that the interpoint distances of the input points δ_{ij} and interpoint distances of the output points d_{ij} are similar. ISOMAP replaces δ_{ij} with estimates of the geodesic distance between x_i and x_j along the manifold. The geodesic distances are approximated by constructing a nearest neighbor graph and then finding the shortest path between two points along this graph using Dijkstra's method (Dijkstra 1959) or Floyd's method (Floyd 1962).

LLE

Local Linear Embedding [LLE; Roweis & Saul (2000)] aims to preserve the local properties of the data and is suited to embedding non-convex manifolds. Details of LLE are provided in Algorithm 2. First, each input point x_i is approximated as a linear combination of its K-nearest neighbors, $x_j : j \in U_K(i)$, with w_{ij} denoting the weight on x_j used to approximate x_i. In the next step, output points are found such that each y_i is well approximated by a linear combination of $y_j : j \in U_K(i)$ and with the weights w_{ij} identical to those computed in the previous step.

Analytical solutions exist both for finding the weights and then given the weights to find the output points. The latter involves an eigendecomposition of a matrix that is sparse due to the fact that only K nearest neighbors are used to compute the weights.

Laplacian Eigenmaps

Laplacian Eigenmaps (Belkin & Niyogi 2003) is based on finding a mapping f from the manifold to $\mathbb{R}^d$ such that the average squared gradient $\int_{x \in \mathcal{M}} ||\nabla f(x)||^2$ is minimized. The rationale is that input points close to one another on the manifold should be mapped to output points that are close to one another, hence f should be as flat as possible. Minimizing the average squared gradient of f corresponds to finding eigenfunctions of the Laplace-Beltrami operator. In practice, eigenvectors of

the graph Laplacian are found where the graph Laplacian serves as a discrete approximation to the Laplace-Beltrami operator. The graph Laplacian is computed from the K-nearest neighbors graph in a way that is described in Algorithm 3.

Hessian LLE

Hessian LLE [HLLE; Donoho & Grimes (2003)] works with the functional $\mathcal{H}(f) = \int_{x \in \mathcal{M}} ||H(f(x))||_F$ which is the average Frobenius norm of the Hessian of a function f where $f : \mathcal{M} \to \mathbb{R}^d$. The Hessian is defined using orthogonal coordinates on the tangent space of $\mathcal{M}$ which are approximated by taking a singular value decomposition of the K-nearest neighbors around each point. If there exists a mapping from the manifold $\mathcal{M}$ to $\mathbb{R}^d$ that is locally isometric, then the null space of $\mathcal{H}(f)$ is spanned by the original coordinates. The functional $\mathcal{H}(f)$ can be estimated, and spectral methods are then used to find the null space and hence recover the isometric coordinates. This is outlined in Algorithm 4.

t-SNE

t-Distributed Stochastic Neighbor Embedding [t-SNE; Van der Maaten & Hinton (2008)] is a dimension reduction technique well suited for the visualization of high-dimensional data in scatterplots. It works by minimizing the Kullback-Leibler divergence $KL(P||Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}$ between two distributions, p_{ij} and q_{ij}, respectively giving transition probabilities between points in the input and outpoint space. To ensure transitions to nearby points are more probable, the transition probabilities p_{ij} are proportional to a Gaussian kernel evaluated at the Euclidean distance δ_{ij}. To better capture the local data structure and solve the crowding problem(Van der Maaten & Hinton 2008) for nearby points, q_{ij} are chosen to be proportional to d_{ij} evaluated at a normalized Student-t kernel. The bandwidth of the Gaussian kernel, σ_i is determined by the *perplexity* which is set to 30 by default. An analytical form of the gradient is used in the Gradient Descent optimization of Kullback-Leibler divergence with details described in Algorithm 5. Variants of the Barnes-Hut algorithm can also be used to approximate the gradient of the object function (Van Der Maaten 2014).

UMAP

UMAP [Uniform Manifold Approximation and Projection; McInnes, Healy & Melville (2018)] preserves global structure is motivated by Riemannian geometry and algebraic topology with a large computational advantage over t-SNE. Rather than minimizing interpoint transition probabilities as in t-SNE, UMAP minimizes the error between two topological representations from the input space and embedding space. The error is measured by the cross entropy C of two local fuzzy simplicial sets built as weighted nearest neighborhood graph, with edge weights representing the likelihood that two points are connected. Geometrically, a simplex is a K-dimensional object built from connecting $K + 1$ points. UMAP captures topological structure by treating the data as

a set of simplices and combining them in a specific way described in Algorithm 6. In order to connect with nearby points, UMAP extends a radius outwards at each point until these overlap. The desired separation between nearby embedded points is controlled by the parameter *min_dist*. Finally, after initializing the embedding using the weighted graph Laplacian, stochastic gradient descent is used to find the most similar graph in lower dimensions. Further details of the algorithm and hyperparameters can be found in Section 4 of McInnes, Healy & Melville (2018).

2.3 Approximate nearest neighbor searching

As seen in the previous section, many manifold learning algorithms begin by finding the graph of nearest neighbors. A naive way to find this graph is to calculate all pairwise distances between each pair of the input data points. This has a complexity of $O(N^2)$ for N observations, which is not efficient when the data set is large. Although faster algorithms exist for exact nearest neighbors, for very large N we propose the use of approximate nearest neighbor algorithms. Whereas an exact nearest neighbor algorithm will, for a given query point x_q, return the point x_{q^*} such that $\delta_{qq^*} \leq \delta_{qj}$ for all $j \neq q^*$, an approximate nearest neighbor algorithm returns a point $x_{q^\dagger}$ such that $\delta_{qq^\dagger} \leq (1+\varepsilon)\delta_{qq^*}$ for some tolerance level $\varepsilon > 0$ (Arya et al. 1998). This is illustrated in Figure 1. This definition can be generalized to K-nearest neighbors. Since our objective is to find the K nearest neighbor graph, our query points are the points in the original sample. So the nearest neighbor of each x_q will be x_q itself. This is disregarded and we search for $K+1$ nearest neighbors.

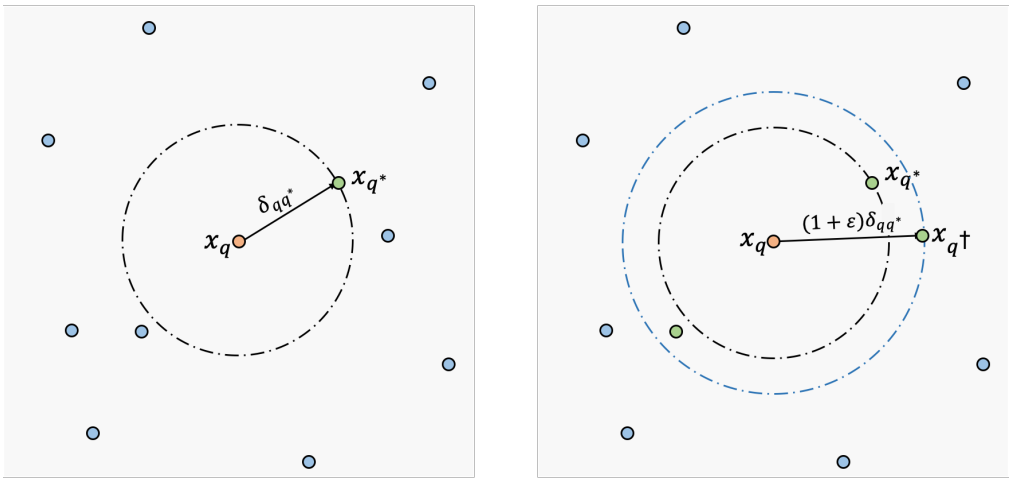

Figure 1: *Illustration of approximate nearest neighbors (right) compared to exact nearest neighbors (left). The left subplot shows the distance from the true nearest neighbor point x_{q^*} to the query point x_q is δ_{qq^*}, while the right subplot shows that the $(1+\varepsilon)$ -approximate nearest neighbors (green points) lie within $(1+\varepsilon)\delta_{qq^*}$ radius from x_q.*

Aumüller, Bernhardsson & Faithfull (2020) developed a benchmark tool[2] and evaluated a number of contemporary approximate nearest neighbor searching methods. Among these methods, Annoy and HNSW are the most competitive ones, so we consider these algorithms in our study. Furthermore,

[2]available at http://ann-benchmarks.com/.

since Annoy is a tree-based method and HNSW is a graph-based method, there is one algorithm from each large class of ANN algorithms. Finally, we will also use k-d trees since this is one of the most widely used algorithms for (approximate) nearest neighbor search. In addition, k-d trees also allow for the special case of exact nearest neighbor search through appropriate selection of the tuning parameters, thus easily allowing us to isolate the effect of using approximate nearest neighbors in manifold learning algorithms. We now briefly describe the approximate nearest neighbor algorithms used in our evaluation, namely, k-d trees, Annoy, and HNSW.

k-d trees

The k-d tree (Bentley 1975), is a binary tree structure that can be exploited for nearest neighbor search. Each node is associated with a partition of p-dimensional space $\mathcal{P}_g$ (hereafter the node and partition will be treated interchangeably). A node has two children — a left child node and a right child node — formed by splitting $\mathcal{P}_g$ by an axis-orthogonal hyper-plane. By a variant of k-d trees proposed by Friedman, Bentley & Finkel (1977), the splitting dimension l_g^* for the splitting hyperplane is chosen to maximize spread. The splitting value c_g^* is chosen as the median along the splitting dimension of all points in $\mathcal{P}_g$. The process of building a k-d tree is shown in Algorithm 7. Note that while Algorithm 7 will construct a tree with terminal nodes that include no more than one data point, variants of k-d trees allow for multiple points in the terminal nodes.

Once a k-d tree is constructed, nearest neighbors can be searched in time proportional to $O(\log(N))$. Nearest neighbor search begins at the root node and determines whether the query point belongs to the partition of the left child node or right child node. The search moves to the child node to which it belongs and this process is iterated until a terminal node $\mathcal{P}_T$ is reached. The distance from the query point to $x_i \in \mathcal{P}_T$ is set to the current shortest distance δ^* and x_i to the current nearest neighbor. The search algorithm then unwinds back through the tree. At each parent node, there will be one sibling node that has been searched and one sibling node yet to be searched. Denote the latter as $\mathcal{P}_{g^*}$. The tightest bounding box containing all $x_i : x_i \in \mathcal{P}_{g^*}$ is computed as well as a hyper-sphere of radius δ^* around the query point. If the hyper-sphere and bounding box do not intersect, then the nearest neighbor cannot lie in $\mathcal{P}_{g^*}$ and the branch starting from this node can be disregarded. Otherwise, this branch must also be searched. Pseudocode for a recursive procedure used to find the nearest neighbor x_{q^*} of a query point x_q is given in Algorithm 8. This search is initialized by evaluating this procedure at the root node and initializing the current closest distance at $\delta^* = \infty$.

The algorithm described above will find exact nearest neighbors with $O(\log(N))$ operations on average. However, if approximate rather than exact nearest neighbors are desired, further speed up can be achieved by setting the radius of the hypersphere at Line 11 and Line 18 of Algorithm 8 to $\delta^*/(1+\varepsilon)$. This increases the number of branches of the tree eliminated from the search. In

subsequent sections, we tune ε to trade off between speed and accuracy. In all cases below where exact nearest neighbors are computed, we employ k-d trees with $\varepsilon = 0$. We also note that while Algorithm 8 finds the nearest neighbor, it can be generalized to finding K nearest neighbors by replacing the current smallest distance and current closest node with the current K-th smallest distance and current K-th closest node respectively.

Annoy

Annoy [Approximate Nearest Neighbors Oh Yeah; Spotify (2016)] is a C++ library with Python bindings that implements an alternative tree-base algorithm for nearest neighbor search. While sharing some similarities, Annoy differs from k-d trees in two main ways. First, rather than using a single tree, the Annoy algorithm constructs a forest of randomly constructed trees that can be searched in parallel. Second, while Annoy builds trees by constructing splitting hyperplanes, unlike k-d trees, these hyperplanes are not axis-orthogonal. Instead, for each split, Annoy chooses two points at random and sets the splitting hyperplane to be equidistant from these two points. Each node (including terminal nodes) is associated with a partition containing multiple (at most κ) points. One such tree is shown in Figure 2.

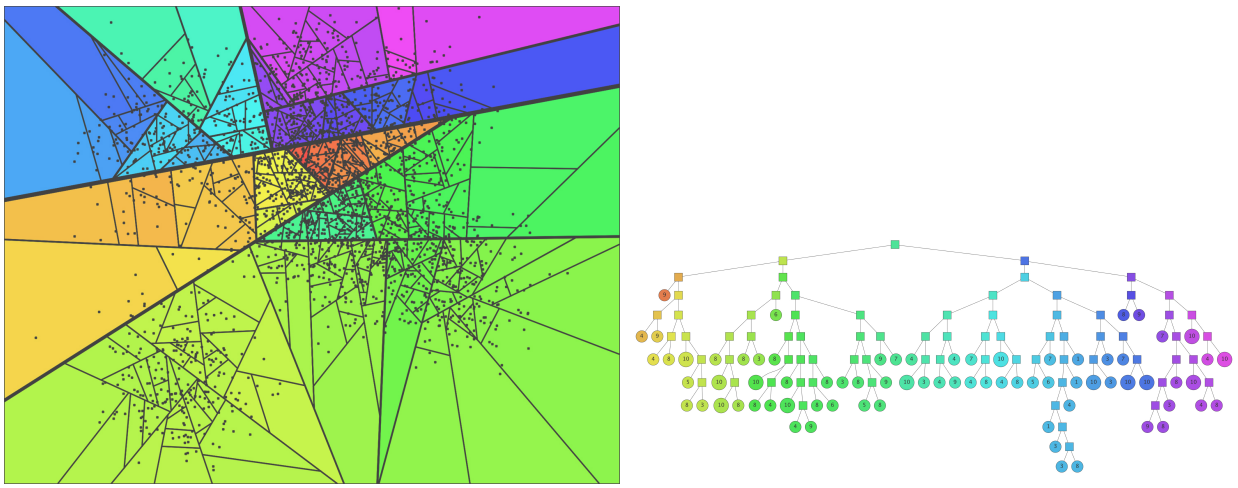

Figure 2: *The splitting process of Annoy (on the left) and the corresponding binary tree (on the right) reproduced from Bernhardsson (2015).*

Annoy searches for nearest neighbors on all trees in the forest in parallel. The search for nearest neighbor candidates on a single tree does share some similarities with nearest neighbor search on k-d trees. In particular, the search begins from the root node and iteratively moves to the child node associated with a partition containing the query point. Once a terminal node is reached, points within this node are considered as nearest neighbor candidates. However, Annoy does not unwind each tree in the same fashion as k-d trees. Instead, Annoy maintains a priority queue (shared across all trees of the forest) of the splitting hyperplanes closest to the query point. If the distance from the query point to a splitting hyperplane is less than some threshold η, then the opposite side of

the hyperplane is also searched. Increasing the threshold throughout allows more points to be considered as nearest neighbor candidates.

Once the union of candidate points across all trees contains *search_k* points, searching the forest ends. Distances are computed to each point in the candidate sets and approximate nearest neighbors found. The accuracy-performance trade-off is controlled by two parameters, the number of trees, *n_trees*, and the size of the candidate set, *search_k*. In both cases, larger values will improve the accuracy of nearest neighbor search at the cost of slower performance. Choosing a larger value of *n_trees* also increases the demand for storage. A full description of the Annoy is provided in Algorithm 9 and 10.

Hierarchical Navigable Small World graphs (HNSW)

Hierarchical Navigable Small World (Malkov & Yashunin 2020) is a graph-based approximate nearest neighbor searching method. Before providing an overview of HNSW, it is instructive to describe how a graph can be greedily traversed to search for the approximate nearest neighbor of a query point. This will be done using Figure 3 as an example where each node of the graph corresponds to a data point and the query point is colored red. Throughout we assume that the edges of the graph connect points that are (approximately) close to one another and searching proceeds via the principle that a *neighbor's neighbor is likely to be a neighbor*. First, a random entry point is selected (suppose this is J). The distance between the query point and entry point is computed and the current nearest point is set to the entry point. Next, the distances between the query point and the current nearest point's neighbors (G, I, and K) are computed. If at least one neighboring point is closer to the query point than the current nearest point, then the current nearest point is updated. In Figure 3 this will be K. The algorithm continues and terminates when there are no unvisited neighboring points that are closer to the query point than the current nearest point. In Figure 3 this will be at point L. A general description of this greedy search is provided in Algorithm 11.

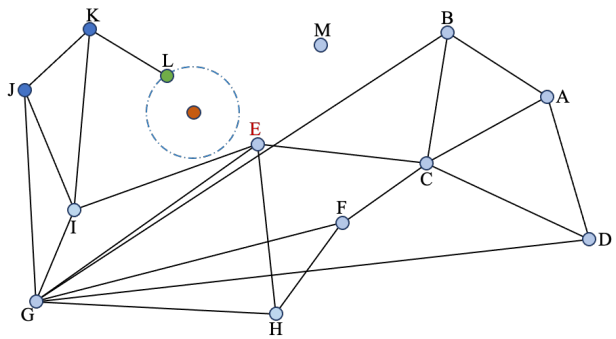

Figure 3: *An example of naive nearest neighbor searching in HNSW. For the red query point, the approximate nearest neighbor is L if the entry point is J. For entry point E or M, the greedy search would fail at approximate nearest neighbor E since the true nearest neighbor is L.*

This greedy search does have some shortcomings, in particular, it will fail for some entry points (for instance E or M in Figure 3) and can be slow when the number of data points is large. To overcome the second issue, HNSW constructs a hierarchy of graphs, with each increasing layer involving fewer and fewer points. This is depicted in Figure 4. Greedy search on the more sparse levels of the hierarchy acts as an expressway mechanism to quickly reach a region of space where nearest neighbors are likely to be found. In Figure 4, the algorithm begins at the randomly chosen red point on Layer 2 (x_e) and by greedy search reaches the blue point on Layer 2 (x_b). This point becomes the entry point for a greedy search on Layer 1 of the graph. This process is iterated until an approximate nearest neighbor is found on the dense graph at Layer 0 (the green point x_a in Figure 4).

The only issue remaining is the construction of the graphs at each level of the hierarchy. This process follows a similar idea to approximate nearest neighbor search of a query point only with each data point sequentially inserted into the graph as a query point. Once approximate nearest neighbors are found for a data point, *n_links* edges are added to the graph between the data point and its approximate nearest neighbors, where *n_links* can be chosen to trade off between speed and accuracy. A maximum layer L^* is randomly chosen for each point such that the point is only inserted into graphs corresponding to Layer L^* and below. Generating L^* from a distribution with exponentially decaying tails ensures that the high layers are sparse. Some pruning of unnecessary edges in each graph is also recommended with details provided in Malkov & Yashunin (2020).

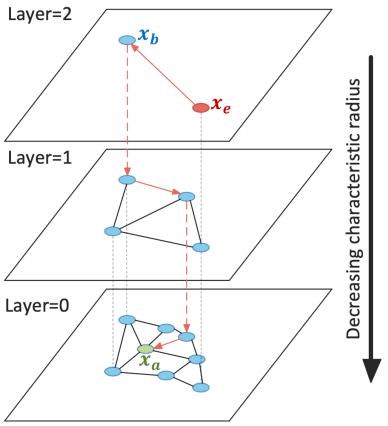

Figure 4: *The hierarchical structure built from HNSW reproduced from Malkov & Yashunin (2020). The red entry point x_e is randomly chosen on the top layer. The greedy search follows the red arrow until an approximate nearest neighbor x_a is found (shown green).*

2.4 Quality measures for manifold learning embedding

To examine the effect of using approximate nearest neighbor algorithms in manifold learning, measures of the quality of a low-dimensional representation must be defined. In the ANN benchmark tool built by Aumüller, Bernhardsson & Faithfull (2020), recall rate is used to measure the accuracy of the approximate nearest neighbor searching methods. Recall rate is defined as the

proportion of observations correctly classified as nearest neighbors compared to the true ones. However, even with a large proportion of true nearest neighbors, the local structure in the manifold learning embeddings might still be inaccurate. Therefore, the quality measures of the embeddings are essential.

There are a number of criteria that we consider, all of which measure the extent to which the nearest neighbor structure of the output points resembles the nearest neighbor structure of the input points. That is, all criteria measure the extent to which the topology of the manifold is preserved while the final criterion we consider (the Procrustes measure), additionally measures the extent to which the local geometry is preserved. For ease of comparison, we adjust all quality measures so that higher values of a measure indicate a higher quality embedding.

Recall that $U_K(i)$ was defined as the set of points j such that x_j is one of the K-nearest neighbors of x_i. We now also define $V_K(i)$ as the nearest neighborhood of observation i in the output space, i.e. a set of points j such that y_j is one of the K-nearest neighbors of y_i. We define the neighborhood ranking of x_j with respect to x_i as $\rho_{ij} = |\{\ell : \delta_{i\ell} < \delta_{ij}\}|$. For example, if x_j is the nearest neighbor of x_i, then $\rho_{ij} = 1$ since only $\delta_{ii} < \delta_{ij}$, where we assume without loss of generality that all input points are distinct. In case of tied distances, $\rho_{ij} = \left|\left\{\ell : \delta_{i\ell} < \delta_{ij} \text{ or } (\delta_{i\ell} = \delta_{ij} \text{ and } \ell < j)\right\}\right|$. The neighborhood rankings of output points, denoted r_{ij}, are similarly defined using d_{ij} in place of δ_{ij}. The value of K used to compute quality measures need not be the same as that used in the manifold learning algorithms, however, in all our simulations, we also set $K = 20$ for the purpose of computing quality measures.

Local Continuity Meta-Criterion (LCMC)

Chen & Buja (2009) proposed the local continuity criterion (LCMC) defined as

$$\text{LCMC}(K) = \frac{1}{NK}\sum_{i=1}^{N}\Big(\big|U_K(i)\bigcap V_K(i)\big| - \frac{K^2}{N-1}\Big).$$

The LCMC computes the average size of the overlap between the K-nearest neighborhood in the output space and K-nearest neighborhood in the input space. The value of $\text{LCMC}(K)$, is bounded between zero and one with values closer to one indicating a larger overlap between nearest neighborhoods and therefore better quality embedding.

Trustworthiness & Continuity (T&C)

Venna & Kaski (2006) defined two quality measures, trustworthiness and continuity, that respectively distinguish two types of errors. For the first type of error, y_j is among the K-nearest neighbors of y_i (i.e. observations close in output space) but x_j is not among the K-nearest neighbors of x_i (i.e. observations not close in the input space). Using all such points for each i, the trustworthiness

of the embedding can be calculated as

$$M_T(K) = 1 - \frac{2}{G_K} \sum_{i=1}^{N} \sum_{\substack{j \in V_K(i) \\ j \notin U_K(i)}} (\rho_{ij} - K),$$

where the normalizing factor $G_K = \begin{cases} NK(2N - 3K - 1) & \text{if } K < N/2, \\ N(N-K)(N-K-1) & \text{if } K \geq N/2. \end{cases}$

This is bounded between zero and one, with values closer to one indicating a higher-quality representation. In contrast to LCMC, Trustworthiness uses information on the rankings of interpoint distances. In particular, points that are incorrectly included as nearest neighbors in the output space are penalized more when they are further away in the input space (ρ_{ij} is high).

For the second type of error, x_j is among the K nearest neighbors of x_i (i.e. observations close in input space) but y_j is not among the K-nearest neighbors of y_i (i.e. observations not close in the output space). Using these points, the continuity is defined as

$$M_C(K) = 1 - \frac{2}{G_K} \sum_{i=1}^{N} \sum_{\substack{j \in U_K(i) \\ j \notin V_K(i)}} (r_{ij} - K).$$

This is also normalized between zero and one and higher values indicate a higher-quality representation. Errors made for points further away in the output space (r_{ij}) are penalized to a greater extent.

Mean Relative Rank Errors (MRREs)

Lee & Verleysen (2008) developed two measures known as mean relative rank errors, based on similar principles as Trustworthiness and Continuity. These are defined as

$$W_n(K) = \frac{1}{H_K} \sum_{i=1}^{N} \sum_{j \in U_K(i)} \frac{|\rho_{ij} - r_{ij}|}{\rho_{ij}},$$
$$W_v(K) = \frac{1}{H_k} \sum_{i=1}^{n} \sum_{j \in V_k(i)} \frac{|\rho_{ij} - r_{ij}|}{r_{ij}},$$

where H_K is the normalizing factor defined as $H_K = n \sum_{i=1}^{K} |N - 2i + 1|/i$.

The set of observations j that lie in the K-nearest neighborhood of i in both the input and output space do not affect the Trustworthiness and Continuity measures. In contrast, Mean Relative Rank Errors will penalize such points, particularly when the ranking of the distance between x_i and x_j differs substantially from the ranking of the distance between y_i and y_j. For ease of comparison to

other quality measures, we will report $1 - W_n(K)$ and $1 - W_v(K)$ so that larger values indicate a better quality embedding.

Co-ranking Matrix ($Q_{NX}(K)$)

The co-ranking matrix criterion is defined as

$$Q_{NX}(K) = \frac{1}{KN} \sum_{k=1}^{K} \sum_{\ell=1}^{K} q_{k\ell},$$

where $q_{k\ell} = |\{(i,j) : \rho_{ij} = k \text{ and } r_{ij} = \ell\}|$ is the element in row k and column ℓ of the co-ranking matrix. For instance, the element in the first row and column of Q, q_{11}, denotes the number of pairs of observations for which the second observation is the nearest neighbor of the first observation in both the input and output space. The range of the criterion is $Q_{NX}(K) \in [0,1]$, where 1 indicates a more accurate representation. It is also worth noting that the T&C, MRREs, and LCMC can be expressed in terms of the co-ranking matrix with details provided by Lee & Verleysen (2008).

Procrustes measure ($R(X,Y)$)

As a final criterion for measuring the quality of a low-dimensional representation, we consider the Procrustes measure (Goldberg & Ritov 2009). The idea rests on the definition of a manifold as a being locally isomorphic to Euclidean space. More concretely, the Procrustes measure considers observations in the neighborhood of x_i, i.e observations $x_j : j \in U_K(i)$. The Procrustes rotation finds a $d \times p$ matrix A and a d-vector b that projects x_j onto a d-dimensional subspace. The aim is for the projected (and translated) points $Ax_j + b$ to be close to the corresponding output points y_j for $j \in U_K(i)$. This measure of closeness is the Procrustes statistic which for observation i, is defined as

$$G_i = \inf_{\{A,b:A'A=I\}} \sum_{j \in U_K(i)} \|x_j - Ay_j - b\|^2.$$

An overall measure of quality is found given by

$$G = \frac{1}{n} \sum_{i=1}^{N} \frac{G_i}{\sum_{j \in U_K(i)} \|x_j\|^2},$$

where the denominator is a normalizing factor that ensures G will lie between zero and one. An advantage of this measure is that it will favor embeddings that preserve the geometric properties (e.g. distances, angles) of the manifold. Since larger values of G indicate a worse representation, to ease comparison with other measures we report $1 - G$.

3 Experiments

3.1 MNIST dataset

The MNIST database [Modified National Institute of Standards and Technology database; LeCun, Cortes & Burges (2010)] is a commonly used benchmark dataset for dimension reduction techniques, consisting of 60,000 grayscale images of handwritten digits in a training set, and 10,000 grayscale images in a test set. It was constructed from a larger database called NIST. In the original NIST dataset, images in the training data were hand-written by American Census Bureau employees (Special Database 3, SD-3), while images in the test data were handwritten by high school students (Special Database 1, SD-1). LeCun, Cortes & Burges (2010) normalized the size of the NIST images and centered them in a $28 \times 28 = 784$ pixel field. Then they mixed the samples from both SD-3 and SD-1 to form the MNIST dataset, where the training set contains 30,000 images from SD-3 and 30,000 images from SD-1, while the test set is composed of 5,000 images from SD-3 and 5,000 images from SD-1.

To examine manifold learning using ANN techniques, we use the MNIST test set of $N = 10,000$ observations, with a dimension equal to the number of pixels ($p = 784$). The embedding dimension for manifold learning is set as $d = 2$ and the number of nearest neighbors is set as $K = 20$. We apply different combinations of manifold learning algorithms and ANN methods to the data, recording the computational time and calculating the embedding quality measures discussed in Section 2.4. All experiments were run in parallel on a high-performance computing cluster with 2.70GHz Xeon-Gold-6150 CPUs.

3.2 Experimental results

Choosing different values of the tuning parameters for approximate nearest neighbor algorithms allows for a trade-off between computational time and accuracy. For k-d trees, we consider values of ε ranging from 0 to 5 in increments of 0.1. Note that setting $\varepsilon = 0$ allows us to compute exact nearest neighbors. For Annoy, we set values of *n_trees* ranging from 2 to 100 in increments of 2 and fix *search_k* as 500. For HNSW, we set *n_links* to range from 2 to 200 incremented by 2. For each set of parameter values, we record the computational time taken to find the nearest neighbor graph as well as the recall rate .

These results are summarized in Figure 5, where each point represents the recall rate and computational time for an ANN algorithm with a specific set of tuning parameters. A vertical line indicating the computation time for exact nearest neighbors is also added to the plot as a baseline for the ANN methods. A similar line is also added to subsequent figures to compare the accuracy and computation time. As expected, each ANN method shows a trade-off between computational

time and accuracy with lower computational time associated with lower recall rates. If a curve lies entirely to the top and left of another curve, this suggests that one ANN method dominates the other with respect to the proportion of true nearest neighbors found. Figure 5 shows that the best performing ANN method is Annoy, followed by HNSW followed by k-d trees. This is in line with the finding of Aumüller, Bernhardsson & Faithfull (2020) who show that Annoy and HNSW both outperform k-d trees.

While the experiment gives a clear ranking of ANN methods according to recall rate, this ranking may not hold when the ANN methods are used as the first step of a manifold learning algorithm, and accuracy is defined according to quality measures of the resulting embedding. For the remainder of this section, we construct similar plots to Figure 5, but with different measures of the quality of a manifold learning embedding rather than recall rate.

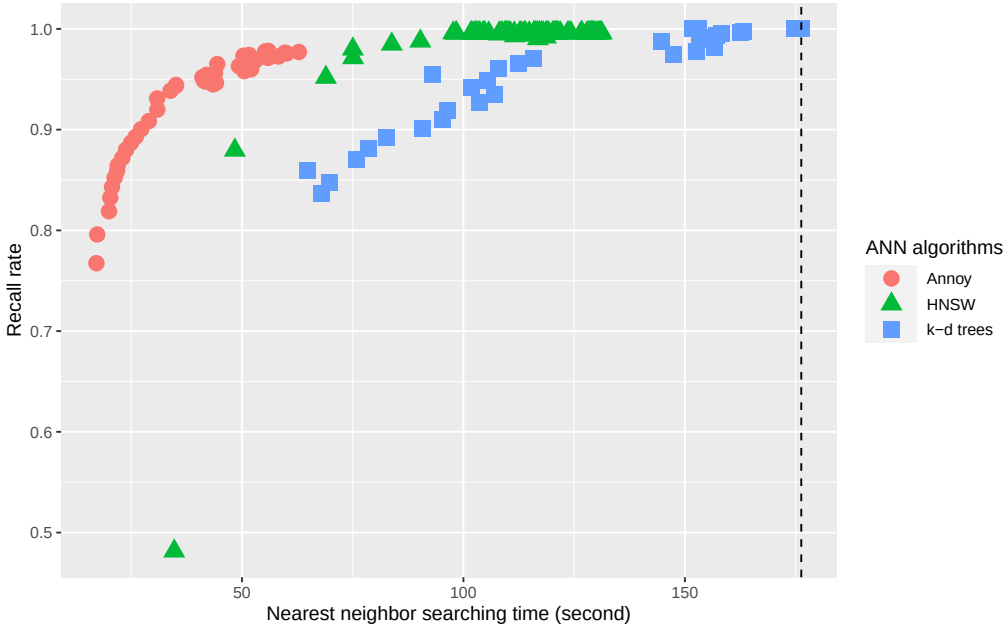

Figure 5: *The comparison plot of recall rate for three ANN methods, k-d trees, Annoy, and HNSW. The points show the change of computation time (second) against the recall rate for different ANN parameter values. Points that are higher in recall rate and less in computation time (topleft) are relatively better. The black vertical line indicates the computation time for finding exact nearest neighbors.*

Figure 6 shows the performance of different combinations of ANN and manifold learning algorithms according to the Trustworthiness measure. Once again ANN methods with a curve closer to the top left of the plot are to be preferred. Each panel refers to a different manifold learning algorithm method, namely ISOMAP, LLE, Laplacian Eigenmaps, Hessian LLE, t-SNE, and UMAP. Since the aim of the paper is not to find optimal manifold learning methods but to introduce ANN in them, we should focus on comparing across different ANN methods in subsequent figures, i.e. comparing points in different shapes and colors.

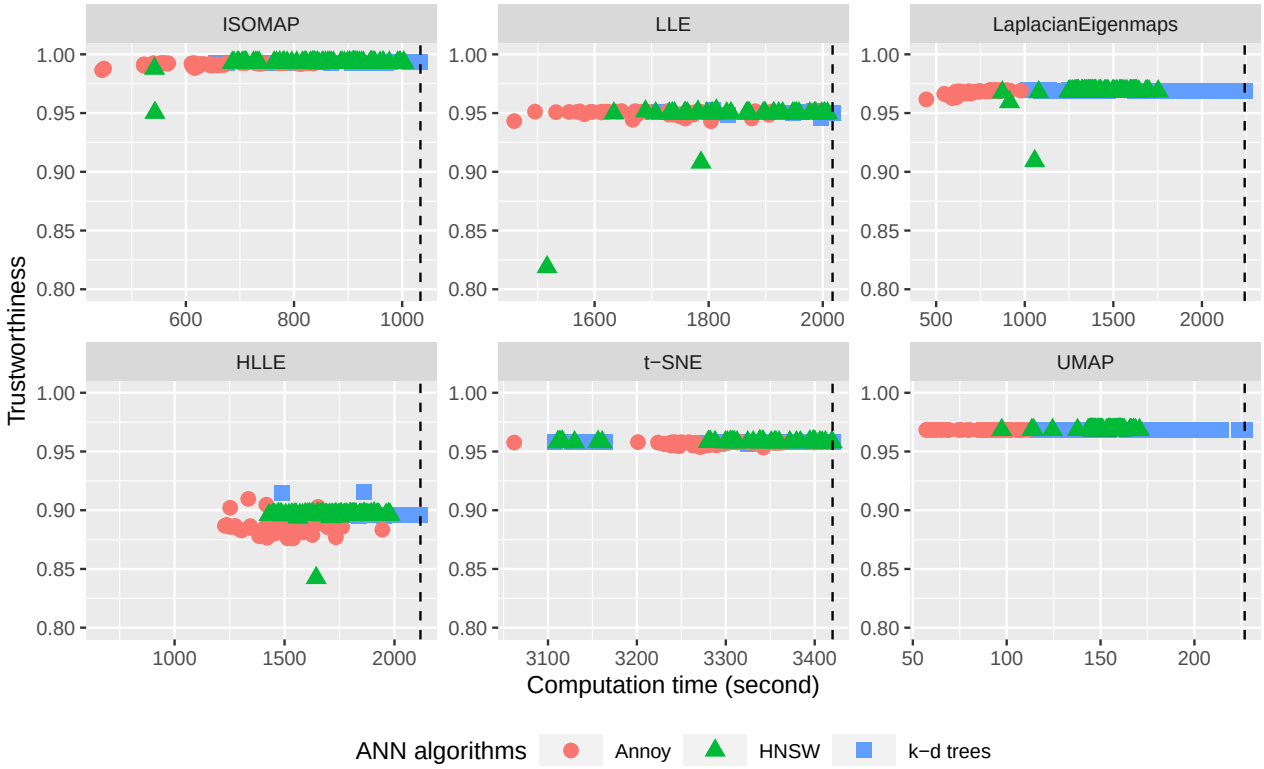

Figure 6: *Trustworthiness against computation time for Annoy, HNSW and k-d trees in six manifold learning methods: ISOMAP, LLE, Laplacian Eigenmaps, Hessian LLE, t-SNE, and UMAP, with each point representing a different parameter value in ANN algorithms. Points that are higher in Trustworthiness and less in computation time are relatively better.*

Figure 6 provides a number of insights. First, the use of approximate nearest neighbors can reduce the computational time taken to carry out manifold learning. For example, using k-d trees with ISOMAP allows computational time to be reduced from around 1,030 seconds to 650 seconds. This reduction in computational time is greatest when Annoy is used, a result consistent with Figure 5. The speed-up is also particularly noticeable for Laplacian Eigenmaps where Annoy can achieve a roughly four-fold improvement in computational time compared to exact nearest neighbors. This result can be explained by the fact that finding the nearest neighbors graph represents more of a computational bottleneck for Laplacian Eigenmaps compared to other algorithms. Second, in a result that stands in contrast to Figure 5, the improvement in computational time does not come at the cost of substantially lower Trustworthiness. In fact, the effect of using a different manifold learning algorithm seems to have a much greater impact on Trustworthiness than the use of an approximate nearest neighbor algorithm. For example, the Trustworthiness is almost always between 0.98 and 1 when ISOMAP is used (the best performing algorithm for this particular dataset), while it is almost always between 0.87 to 0.92 for Hessian LLE (the worst-performing algorithm for this particular dataset). As a minor caveat to the conclusion that ANN has a negligible impact on embedding accuracy, we note that HNSW has a small number of very inaccurate embeddings that lead to a substantially lower Trustworthiness. This may be explained by the propensity of the

greedy search upon which HNSW is built to return nearest neighbors that are quite far from the true nearest neighbor.

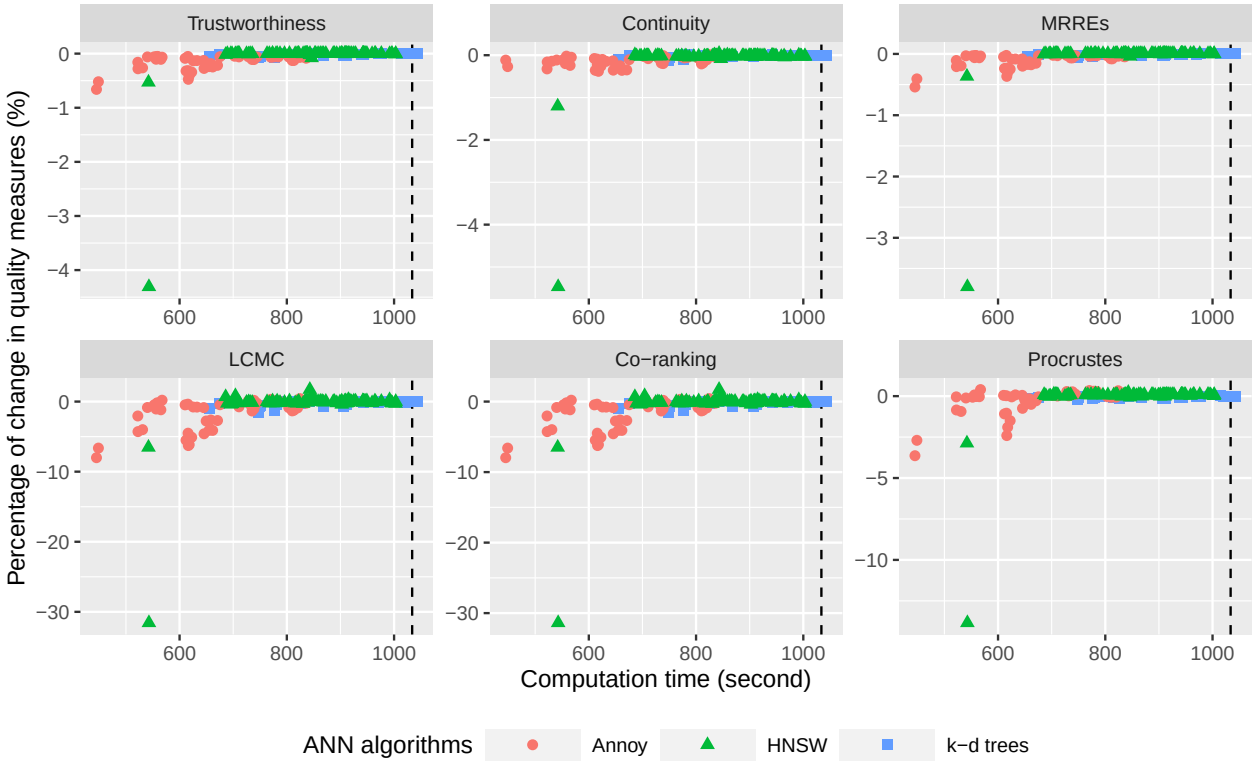

Figure 7: *Comparison of ISOMAP embedding quality measures against computation time for three ANN methods, Annoy, HNSW, and k-d trees. The percentage change compared to true nearest neighbors of six quality measures are shown in the order of Trustworthiness, Continuity, MRREs, LCMC, Co-ranking matrix, and Procrustes measure.*

The results from Figure 6 are, for the most part, robust to the use of metrics used to evaluate embedding quality. For instance, Figure 7 shows results of percentage change of all quality measures discussed in Section 2.4 for ISOMAP. For all measures, the computational time of the algorithm can be cut in half with an associated reduction in the accuracy of less than 10% (with a few exceptions due to the instability of HNSW). Similar conclusions can be drawn for LLE, Laplacian Eigenmaps, t-SNE, and UMAP so these figures[3] are omitted for brevity. On the other hand, Figure 8 does show that for Hessian LLE, HNSW can achieve a higher level of accuracy for some measures (Continuity, LCMC, Co-ranking, and Procrustes), even compared to exact nearest neighbors. While this result is quite counter-intuitive, it should be noted that Hessian LLE performs considerably worse than ISOMAP (in general ISOMAP achieves an LCMC of above 0.3 for ISOMAP, the corresponding figure never exceeds 0.2 for Hessian LLE). In general, with a wide range of different ANN parameter values, the combination of ANN methods with all six manifold learning methods has shown an obvious reduction in computation time with an accuracy loss of less than 10%.

[3]available at https://github.com/ffancheng/paper-mlann/tree/public/paper/figures/public.

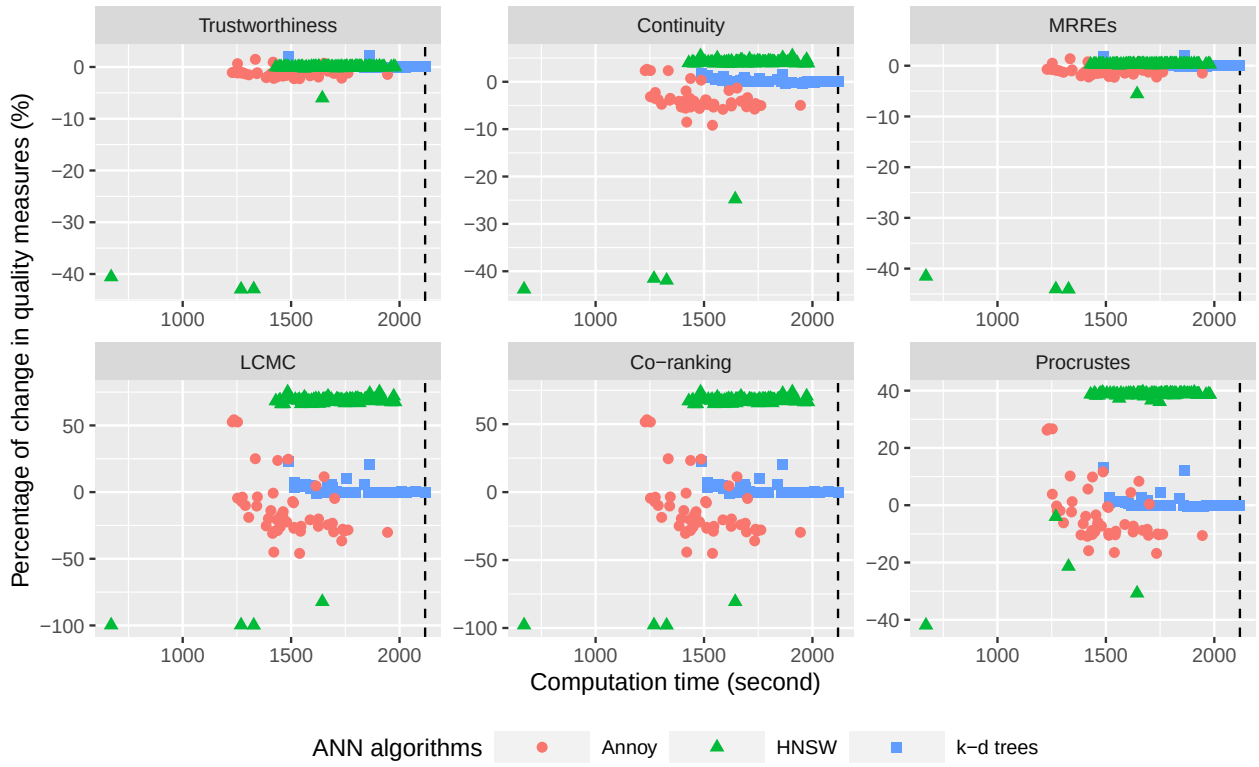

Figure 8: *Comparison of percentage change in six Hessian LLE embedding quality measures against computation time for three ANN methods, Annoy, HNSW and k-d trees.*

Overall, the results from this benchmark study do seem to suggest that approximate nearest neighbors are suitable for use in manifold learning algorithms. In particular, we would recommend Annoy for the greatest computational speed up and warn that care should be taken if using HNSW due to the instability of this algorithm in a small number of cases. When using Annoy, the improvement in computational time comes at the cost of at most a small reduction in the accuracy. This result is robust to the use of different manifold learning algorithms, different measures of embedding accuracy, and different choices of tuning parameters for approximate nearest neighbor algorithms.

4 Application to smart meter data

4.1 Irish Smart meter dataset

Next, we consider smart-meter data for residential and non-profiled meter consumers, collected in the *CER Smart Metering Project - Electricity Customer Behaviour Trial, 2009-2010* in Ireland (Commission for Energy Regulation (CER) 2012). Electricity smart meters record consumption, on a near real-time basis, at the level of individual commercial and residential properties. The CER dataset[4] does not include energy for heating systems since it is either metered separately, or households use a different source of energy, such as oil or gas. In this study, the installed cooling systems are also not reported.

[4]accessed via the Irish Social Science Data Archive - www.ucd.ie/issda.

We use measurements of half-hourly electricity consumption gathered from 3,639 residential consumers over 535 consecutive days. Every meter provides electricity consumption between 14 July 2009 and 31 December 2010. Demand data from two smart meters (ID 1003 and 1539) are shown in Figure 9 as time-series plots. It is obvious that these meters have relatively different patterns. Meter 1539 (bottom of Figure 9) has a period of around 150 days with lower (approximately half) the electricity usage of remaining days and is otherwise relatively stable. In contrast, Meter 1003 (top of Figure 9) exhibits regular spikes on weekends.

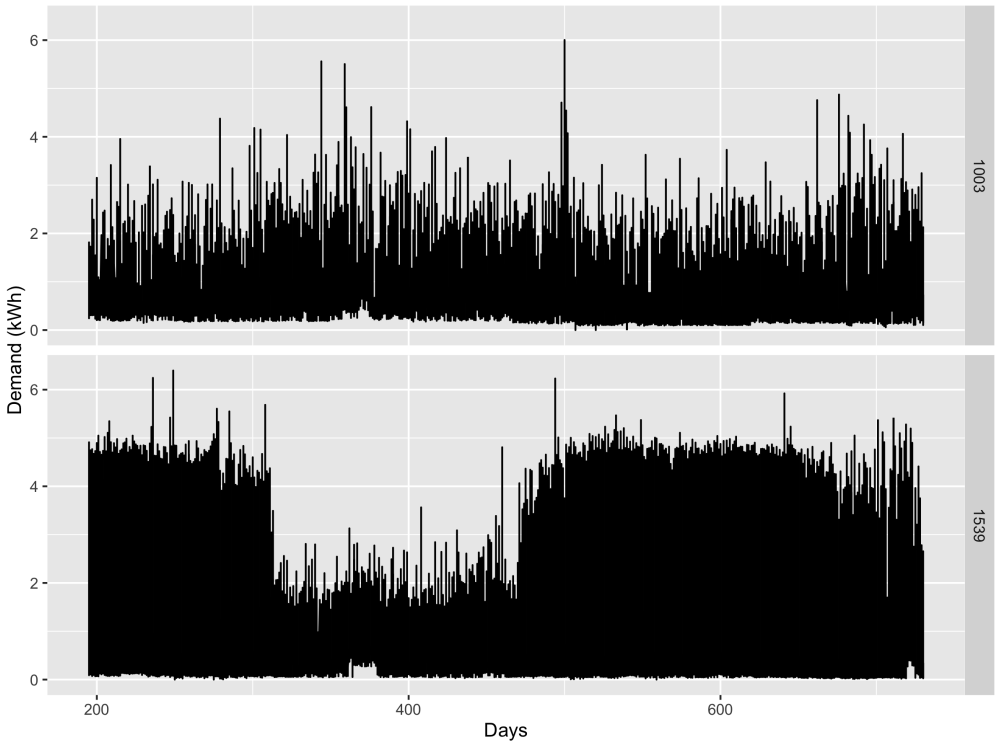

Figure 9: *Two smart-meter demand examples, ID 1003 and ID 1539, from the Irish smart meter data set.*

For electricity demand data, one particular problem of interest is to visualize and identify households or periods of the week with anomalous usage patterns. For this reason, the object of interest in comparing households or periods of the week is the distribution of electricity demand rather than the raw data itself (Hyndman, Liu & Pinson 2018). An additional advantage of looking at distributions rather than raw data is that it provides a convenient mechanism for handling missing data which can be a significant problem in smart meter data applications. In the next section, we first describe how discrete approximations to the distributions of interest are computed. We then describe how the connection between popular metrics on the space of probability distributions (in particular the Hellinger and Total Variation distance) and the L_1 and L_2 norms of a vector containing probabilities allows us to apply manifold learning techniques on statistical manifolds.

4.2 Data processing

Estimating empirical distributions

Let $d_{i,t}$ denote the electricity demand for observation i and for time period t (subsequently we will see that i can either index the household, time of the week, or both, while t may index the week or half-hour period). The objective is to approximate the distribution of electricity demand for observation i, F_i, over time. The first step is to group all data together and construct an evenly spaced grid $\kappa_0, \kappa_1, \ldots, \kappa_G$ of size $G = 200$ with $\kappa_0 = \min_{i,t}\{d_{i,t}\}$ and $\kappa_G = \max_{i,t}\{d_{i,t}\}$ as endpoints. This provides 200 bins which can be used to construct a discrete distribution approximation to F_i. This is found by computing $\pi_{i,g} = (1/T)\sum_t I(\kappa_{g-1} < d_{i,t} < \kappa_g)$ where T is the total number of time periods. The resulting vector π_i represents a probability mass function over the discrete bins, i.e. the input dimension is $p = 200$.

As an alternative, a kernel density estimate could be used. However, since the data contain a large number of zeros, a discrete distribution is more appropriate than a smooth continuous distribution. Also, the data are heavily skewed meaning that standard kernel densities may underestimate the tails of the distribution. Most importantly, an advantage of using a discrete distribution is that manifold learning algorithms with ANN can be directly applied to π_i or simple transformations of π_i in a way that either the Euclidean or Manhattan distance between such vectors will correspond to popular metrics for probability distributions.

Hellinger distance and Total Variation distance

For the manifold learning algorithms discussed in Section 2, it is often assumed that the manifold lies in a p-dimensional ambient space. Although the approximate nearest neighbor algorithms we consider can be applied to metric spaces that are not necessarily Euclidean, they generally require that observations can be characterized as vectors in $\mathbb{R}^p$. Therefore it may not immediately be clear how these algorithms can be applied to a statistical manifold, which brings the second contribution of the paper. The key advantage of estimating the electricity usage distributions as discrete over a domain of fixed bins is that it allows each distribution to be characterized as a vector. In this case, the Euclidean distance between $\sqrt{\pi_i}$ and $\sqrt{\pi_j}$ where square roots are taken element-wise is equal to the Hellinger distance between our approximations to F_i and F_j (up to a constant scale). Similarly, the Manhattan distance between π_i and π_j is equal to the Total Variation distance between $\hat{F}_i$ and $\hat{F}_j$. The Hellinger distance and Total Variation distance are popular metrics used on the space of probability distributions (Hellinger 1909; LeCam et al. 1973) and at least locally are good approximations to the Fisher Information metric used for statistical manifolds. By exploiting this connection between L_1 and L_2 norms of vector representations of probability mass functions, and

metrics on spaces of probability distributions, we make it possible to apply manifold learning algorithms to an application where the observations are probability distributions.

4.3 Manifold learning results for single household

To start with, we consider the case of a single household where each observation is a distribution corresponding to a single half-hour of the week so that each i corresponds to one of $48 \times 7 = 336$ day-time pairs and t corresponds to a week. In this case, $N = 336$ in the manifold learning. We choose meter ID 1003 for this purpose since it is a household with a fairly typical pattern of electricity usage. Figure 10 shows a nearest neighbors graphs with $K = 20$ for meter ID 1003. The left two panels are found using k-d trees with the left panel being exact ($\varepsilon = 0$) and the middle panel being approximate ($\varepsilon = 1$). We note that the ANN output with $\varepsilon = 1$ does not differ much from other values such as a ε of 0.5 or 2. We also use Annoy with $n\_trees = search\_k = 50$ to plot the corresponding nearest neighbor graph on the right panel of Figure 10. The three subplots appear quite similar to each other with a recall rate of 1 for the exact graph, 0.988 for k-d trees, and 0.992 for Annoy, suggesting that for the smart meter data, ANN techniques can be used within manifold learning to save computation time and memory without having too severe an impact on the quality of the low-dimensional representation.

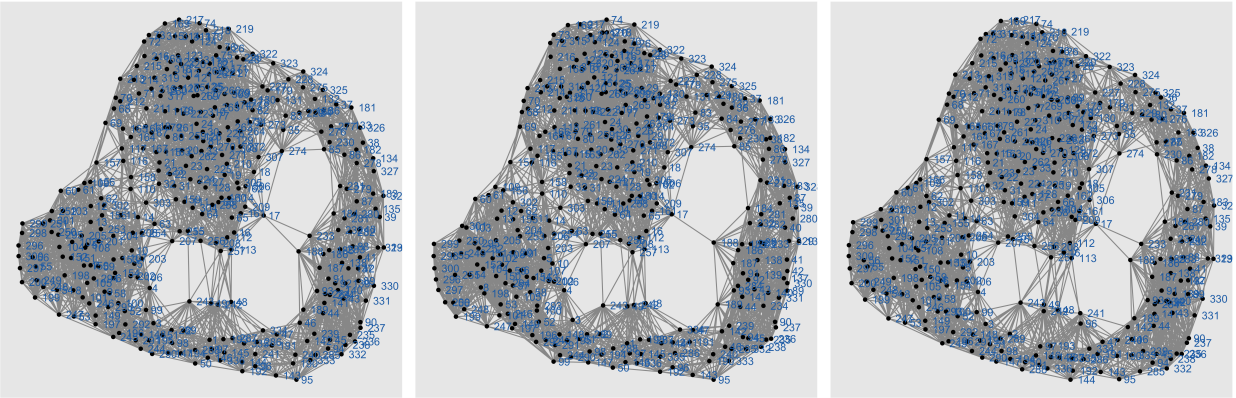

Figure 10: *Nearest neighborhood graphs for meter ID 1003 with $K = 20$. The left subplot is the exact nearest neighborhood graph, while the middle and right subplots are the approximate ones using k-d trees ($\varepsilon = 1$) and Annoy ($n\_trees = search\_k = 50$) respectively.*

Since a key objective is the visualization of anomalous times of the week, Figure 11 shows $d = 2$-dimensional representations of the data for meter ID 1003 obtained using, from left to right, ISOMAP, LLE, Laplacian Eigenmaps, Hessian LLE, t-SNE and UMAP. All cases in the top panels refer to results when exact nearest neighbors are used, while the middle and bottom panels refer to results where approximate nearest neighbors are used. The half-hour of the day that an observation belongs to is depicted using color. With the exception of LLE, the results demonstrate that similar times of day are grouped closely together. The cyclical pattern observed in the representation is indicative of the fact that low values for half-hour of the day (00:00, 00:30, 01:00) are temporally proximate to

Table 1: *Comparison of Trustworthiness measure using true nearest neighbors, k-d trees and Annoy for six manifold learning methods in meter ID 1003. ISOMAP with Annoy gives the highest Trustworthiness.*

	Isomap	LLE	Laplacian Eigenmaps	Hessian LLE	t-SNE	UMAP
Exact NN	**0.983**	0.821	0.937	0.964	**0.973**	0.976
ANN k-d trees	**0.983**	0.813	0.937	**0.966**	0.962	0.975
ANN Annoy	**0.983**	**0.869**	**0.938**	0.961	0.970	**0.978**

Table 2: *Comparison of computation time using true nearest neighbors, k-d trees, and Annoy for six manifold learning methods in meter ID 1003. Generally, k-d trees implementations for both exact and approximate NN are faster than Annoy.*

	Isomap	LLE	Laplacian Eigenmaps	Hessian LLE	t-SNE	UMAP
Exact NN	**1.724**	2.581	**1.900**	1.412	1.452	1.781
ANN k-d trees	1.914	**2.266**	**1.900**	**1.397**	**1.446**	**1.778**
ANN Annoy	1.929	2.633	1.926	1.438	1.520	1.832

high values for half-hour of the day (22:30, 23:00, 23:30) and are therefore similar. Figure 11 also exhibits three clusters roughly corresponding to three phases of a typical day, working during the day (08:00 – 17:00), recreation during the evening (17:00 – 00:00), and sleeping (00:00 – 08:00). The times of day were not explicitly used in constructing results, rather this structure was uncovered by the manifold learning algorithms themselves. This grouping is particularly prominent in Hessian LLE and to a lesser extent in Laplacian Eigenmaps. Encouragingly, when comparing row-wise, all patterns are equally prominent irrespective of whether exact or approximate nearest neighbors are used.

Table 1 also shows one of the embedding quality measures, Trustworthiness, for all the embeddings in Figure 11. By comparing the Trustworthiness, it can be shown that using ISOMAP gives the highest Trustworthiness of 0.983, while Annoy leads to slightly better trustworthiness than even exact nearest neighbors. This conclusion can also be drawn for almost all the other quality measures mentioned in Section 2.4. The running times are summarized in Table 2, with the run time for k-d trees generally shorter than Annoy. Note that we use the same k-d trees implementation for exact nearest neighbor searching, but with $\varepsilon = 0$. In general, the improvement from using approximate nearest neighbors relative to exact nearest neighbors is fairly minor in this small subset of the data. However, we reiterate that even exact nearest neighbors here use k-d trees, compared to other methods in the literature that require all N^2 pairwise distances to be computed and are thus infeasible.

One particular problem of interest in visualizing smart meter data is to identify anomalous times of the week. For this reason, Figure 12 displays the same two-dimensional representations from using ISOMAP and Annoy as Figure 11 but with colors now used to demonstrate points in areas of high

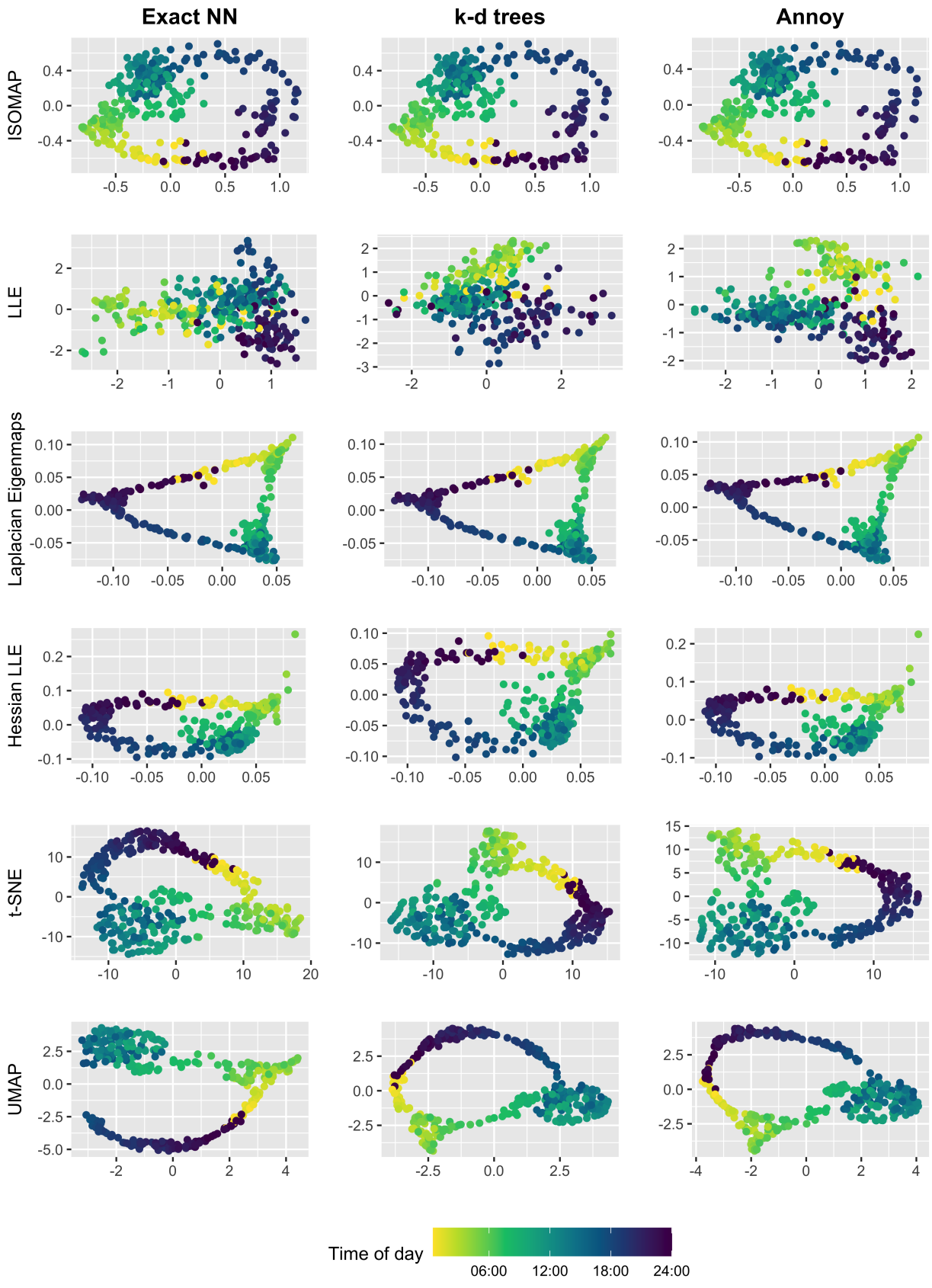

Figure 11: *Embeddings from six manifold learning methods (ISOMAP, LLE, Laplacian eigenmaps, Hessian LLE, t-SNE and UMAP) for one household from the smart meter data. Each subplot is a scatterplot of the 2-d embedding points, with the color representing half-hourly time of the day. Top-panel: exact nearest neighbors. Middle-panel: ANN with k-d trees. Bottom-panel: ANN with Annoy.*

density using the method of Hyndman (1996). We identify the anomalies as the observations with the top 10 lowest density values and these are shown in black points with the time of week index labeled in blue. While the anomalies identified by each manifold learning algorithm differ, when the same manifold learning algorithm, ISOMAP, is used, the use of approximate nearest neighbors has little impact on the identified anomalies.

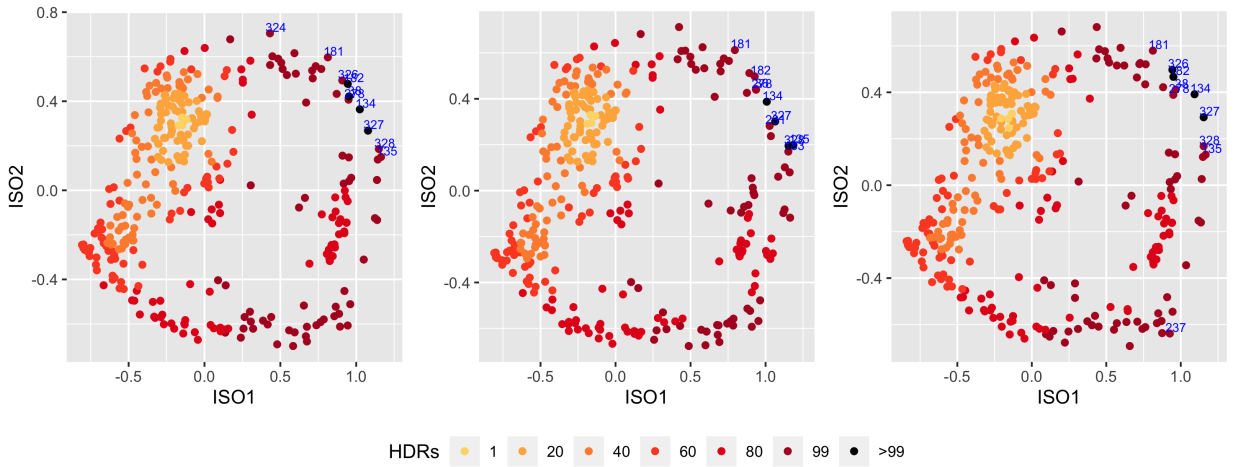

Figure 12: *Highest density region plot for one household (Meter 1003) from the ISOMAP embeddings using exact NN (left panel), k-d trees (middle panel) and Annoy (right panel). The different colours represent highest density regions with different coverage probabilities. The most typical points (the 1% HDR) are shown in yellow, while the top 10 most anomalous are shown in black with the time of week indexed in blue.*

We can gain further insight by comparing distributions corresponding to specific times of the week identified as anomalous and typical by Figure 12. In Figure 13, the left two panels in orange correspond to the typical periods 134 and 317 (corresponding to Monday 12 pm and Sunday 11 am respectively), while the right two panels in black display the anomalous, periods 24 and 310 (corresponding to Wednesday 7 am and Sunday 11 pm respectively). The typical periods show relatively lower electricity usage, and the distributions and are skewed right. In contrast, for anomalous periods, electricity demand is generally higher and distributions exhibit a thicker right tail. In particular, there is a more bimodal distribution on Wednesday at 7 pm, which may suggest certain patterns of behavior are undertaken by this specific household on some but not all weeks at this time (e.g. entertaining guests).

4.4 Comparison between households

To compare all 3,639 households in the Irish smart meter data we consider two ways to compute distances between observations. The first is to compute the density of each household's energy usage while ignoring the time of the week with $N = 3,639$; using the notation of Section 4.2, i corresponds to the household and t corresponds to the half-hour period. However, since the time of week is highly informative in electricity consumption data, an alternative would be to

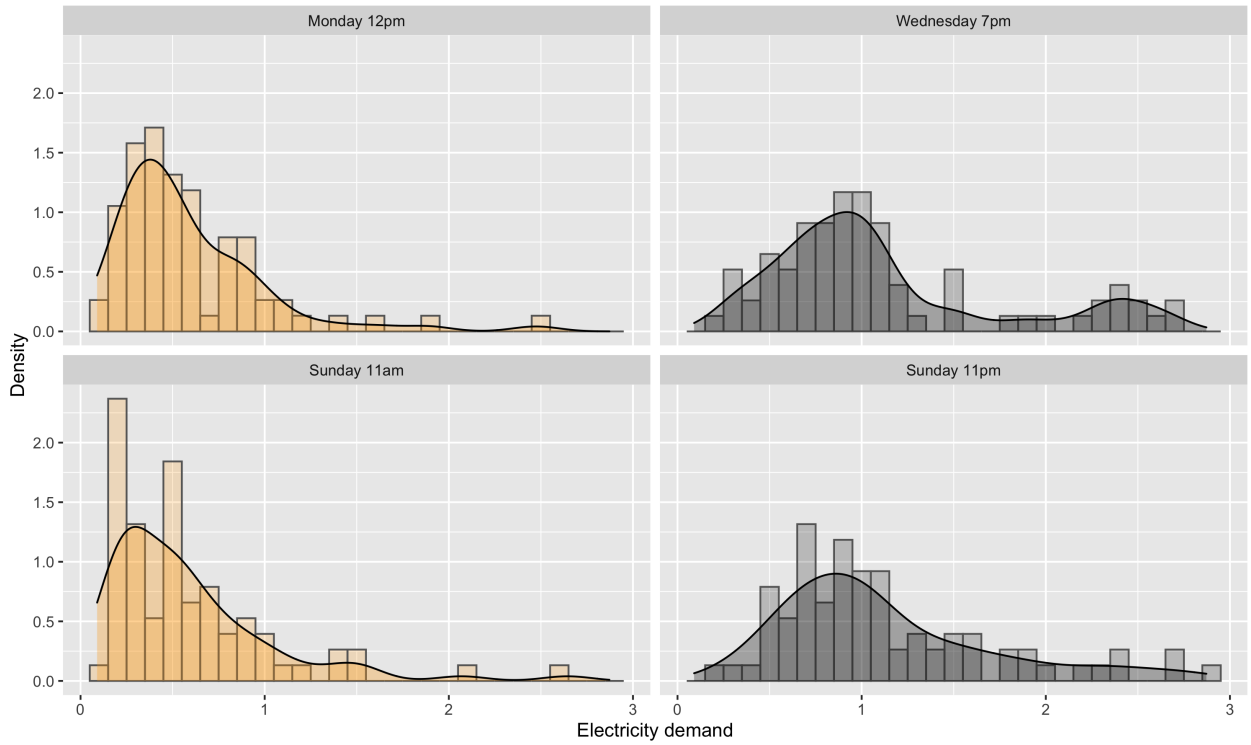

Figure 13: *Electricity demand distribution plots for four time of week periods of meter ID 1003. The most typical periods, Monday 12pm and Sunday 11am, are shown in orange (left panels), while the most anomalous periods, Wednesday 7pm and Sunday 11pm, are shown in black (right panels).*

compute densities such that the index i corresponds to a household/time-of-week pair and index t corresponds to the week, i.e. $N = 3,639 * 336 = 1,222,704$. In this case, we use $F_{j,h}$ to denote the distribution corresponding to household j and time of week h, and $\pi_{j,k}$ to denote a vector whose entries give values of a probability mass function of a discrete approximation to $F_{j,h}$. The distance between household j and household k can then be computed as

$$\delta_{j,k} = \sum_{h=1}^{336} d(F_{j,h}, F_{k,h}), \tag{1}$$

where $d(.,.)$ is the total variation metric between two discrete distributions. This metric resembles that used by Hyndman, Liu & Pinson (2018) who use a sum of Jenson-Shannon divergences rather than the total variation metric. However, we propose the use of total variation metric with an important advantage because the $\delta_{j,k}$ is equivalent to a Manhattan distance between the stacked vectors $\pi_j = (\pi'_{j,1}, \ldots, \pi'_{j,336})'$ and $\pi_k = (\pi'_{k,1}, \ldots, \pi'_{k,336})'$. Unlike Hyndman, Liu & Pinson (2018), we do not have to compute all pairwise distances (an $O(n^2)$ operation) before computing nearest neighbors; instead, we can compute exact nearest neighbors using k-d trees or, in an even faster time, approximate nearest neighbors.

Similar to the single household case in Section 4.3, exact nearest neighbors are implemented by setting $\varepsilon = 0$ in k-d trees and approximate nearest neighbor is implemented using k-d trees with

Table 3: *Comparison of Trustworthiness measure for all household embeddings using true nearest neighbors, k-d trees, and Annoy for six manifold learning methods. ISOMAP and UMAP give the highest Trustworthiness while Hessian LLE gives the lowest.*

	Isomap	LLE	Laplacian Eigenmaps	Hessian LLE	t-SNE	UMAP
Exact NN	**0.985**	**0.952**	**0.966**	**0.948**	**0.975**	**0.987**
ANN k-d trees	**0.985**	0.917	0.965	0.943	**0.975**	**0.987**
ANN Annoy	**0.985**	**0.952**	**0.966**	0.943	**0.975**	0.986

$\varepsilon = 1$ and Annoy with $n\_trees = 50$. The distance metric between households is defined as in Equation (1). The recall rates for these three methods are 1, 0.892, 0.994 respectively. For each scenario, the same six manifold learning algorithms are used as in Section 4.3. The highest density region plots of the embeddings are shown in Figure 14 with exact nearest neighbors on the top panel, ANN with k-d trees on the middle panel, and ANN with Annoy on the bottom panel. Also, the trustworthiness measure is computed for all combinations of manifold learning and nearest neighbor searching algorithms with results summarized in Table 3, and the corresponding computation time is reported in Table 4.

ISOMAP and t-SNE are highly robust to the use of approximate nearest neighbors with the trustworthiness measures equivalent across all nearest neighbor algorithms to the third decimal place. Once again, ISOMAP, together with UMAP, yields fairly accurate embedding according to the trustworthiness metric. Results are similar if other accuracy metrics are used. The households identified as anomalous do not change for ISOMAP, Laplacian Eigenmaps, t-SNE, and UMAP when approximate nearest neighbors are used. For the LLE and Hessian LLE, the trustworthiness and the identified anomalies do change across different ANN methods, however, the differences are minor. The Hessian LLE embedding has the overall lowest trustworthiness measure which may explain why it appears to exhibit some degeneracy in Figure 14. An alternative explanation for the poor performance of Hessian LLE is that Hessian LLE is predicated on the existence of an isometric d-dimensional embedding, and for this specific dataset such an isometry may not exist for $d = 2$. As was the case in Section 4.3, whether the focus is upon the identification of anomalies or the trustworthiness measure, the impact of using different approximate nearest neighbor algorithms is insubstantial when compared to the choice of the manifold learning algorithm. However, in contrast to the earlier results, Table 4 shows that a computational speedup is mostly observed when an approximate version of k-d trees is used and not for Annoy. This may be due to the fact that the discrete approximation is to a multivariate density and hence has a higher dimensionality. Annoy is generally recommended for data with a moderately high dimension of a few hundred to a few thousand.

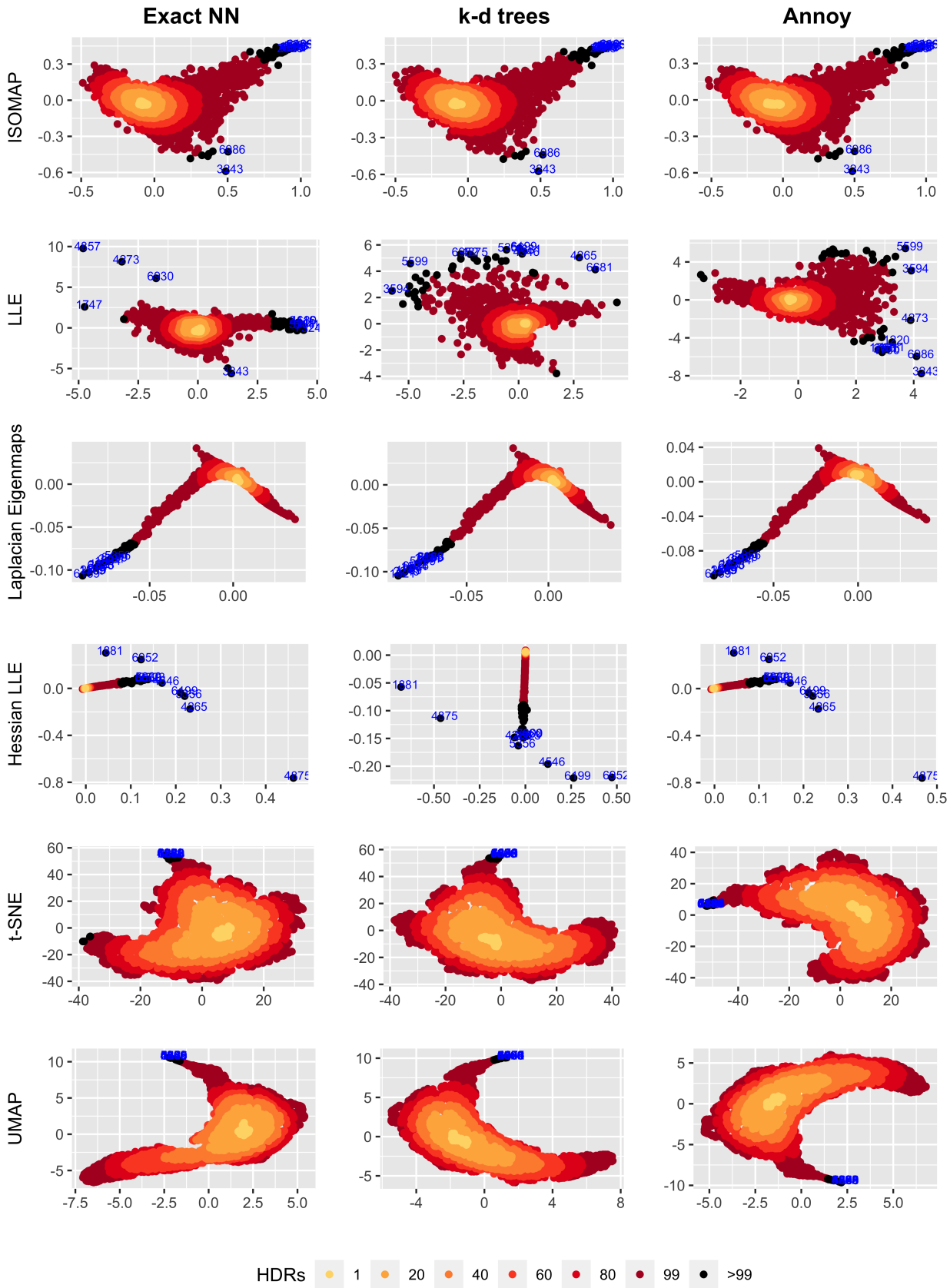

Figure 14: *Highest density region plots from six manifold learning methods for all 3,639 households, with each point representing the distribution of one household. Top row: exact nearest neighbors. Middle row: ANN with k-d trees. Bottom row: ANN with Annoy.*

Table 4: *Comparison of computation time for all household embeddings using true nearest neighbors, k-d trees, and Annoy for six manifold learning methods. Laplacian Eigenmaps with k-d trees saves the most computation time while Annoy is the slowest.*

	Isomap	LLE	Laplacian Eigenmaps	Hessian LLE	t-SNE	UMAP
Exact NN	**11.350**	22.109	3.086	5.964	535.705	4.184
ANN k-d trees	11.433	**18.820**	**2.781**	5.171	**508.336**	**4.114**
ANN Annoy	16.164	24.239	7.756	**5.128**	512.593	4.970

Further insights are gained by comparing the electricity demand distributions of typical and anomalous households. For example, in the ISOMAP highest density region plot, the meter IDs 3243 and 4669, are both detected as anomalies. However, they lie far away from one other in the embedding plot, suggesting that while both are anomalous, their distributions are quite different. We can compare these to one typical household, namely the household ID 1321). In Figure 15, the electricity usage data of these three households are plotted over all trial days, while the corresponding quantile region plots of electricity demand by half-hour and day of the week are given in Figure 16. For all three households, the time of the week patterns are revealed in the top panel of Figure 16. When compared to the typical household 1321, the electricity demand of the anomalous households is much lower. The highest density region boxplots show that the distributions of the two anomalous households, 3243 and 4669 are quite different from each other, which is consistent with their far apart locations in the embedding plots. For household 3243, in most of the days, the electricity demand is below 1 kWh per day indicating that there are not many electrical applicants in this household or that people are away for most of the trial days, which also explains some spikes in certain days. The situation is quite different for household 4669 where for most of the time the electricity usage is below 2 kWh per day and there are certain periods with almost no usage. This shows that this household might have vacations for holidays and leave only the necessary electrical applicants on while not at home. These findings show that the manifold learning with approximate nearest neighbors works well in finding anomalous households in the smart meter data despite the large volume of data.

5 Conclusions

In this paper, we propose the use of a broad range of approximate nearest neighbors in manifold learning algorithms to attain a lower-dimensional embedding of the high-dimensional data. By applying different combinations of manifold learning algorithms (ISOMAP, LLE, Laplacian Eigenmaps, Hessian LLE, t-SNE, and UMAP) and approximate nearest neighbor methods (k-d trees, Annoy, and HNSW) to the benchmark MNIST data of handwritten digits, we show that the use of

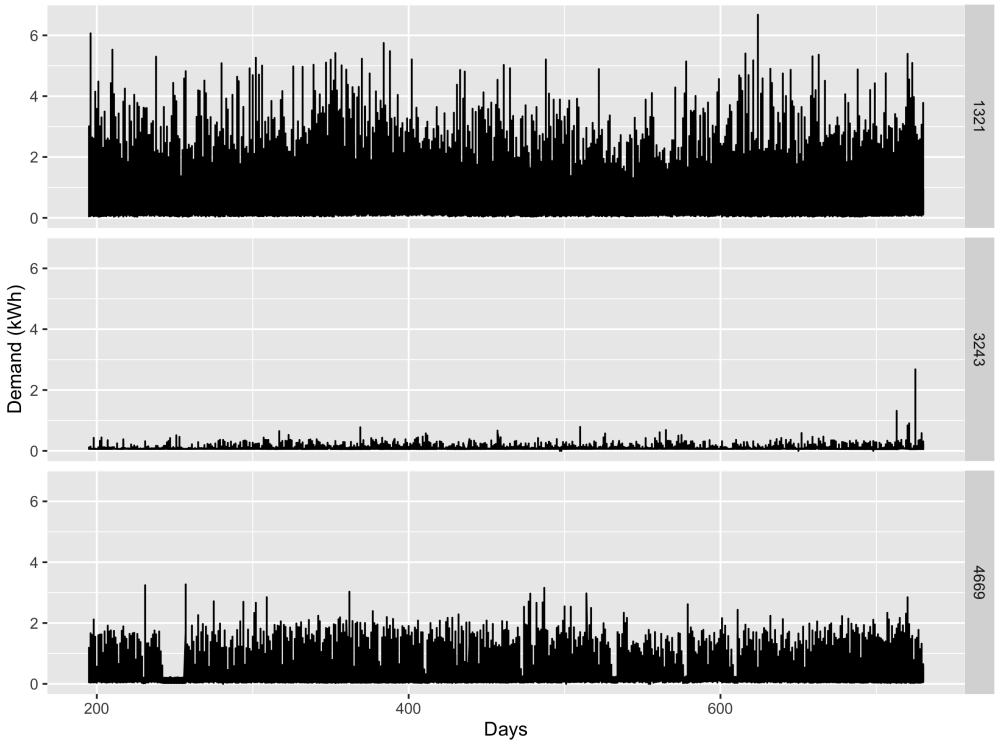

Figure 15: *Electricity demand plots of all 535 days for one typical household 1321 and two anomalies, 3243 and 4699.*

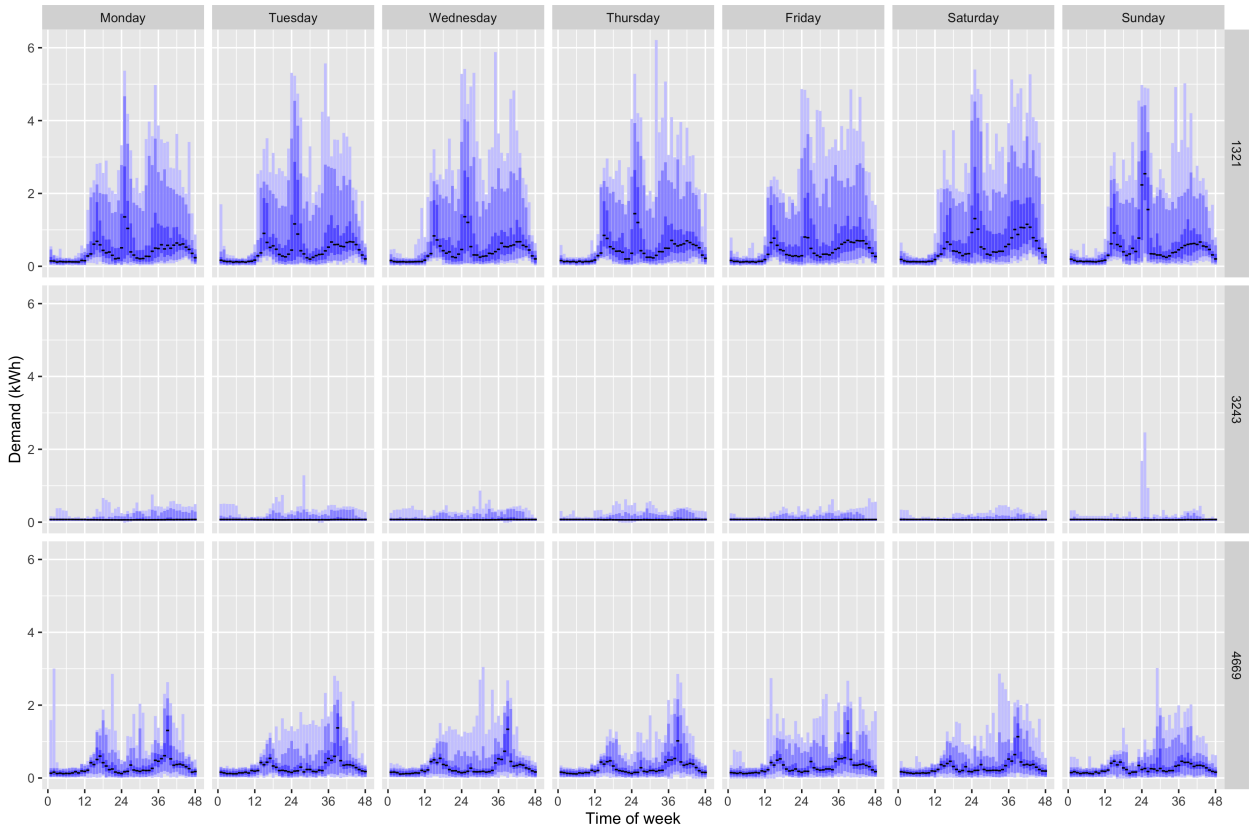

Figure 16: *Quantile region plots of electricity demand against the time of week for one typical household 1321 and two anomalies, 3243 and 4699. The quantile regions displayed in the plot are 99.0%, 95.0%, 75%, and 50%.*

ANN methods significantly reduces computational time while maintaining a high level of embedding quality across a range of accuracy measures (LCMC, Trustworthiness & Continuity, MRREs, Co-ranking matrix, and Procrustes measure). Annoy gives the greatest reduction in computation time — almost four-fold when applied to Laplacian Eigenmaps. While HNSW generally achieves a high recall rate, it can return very inaccurate results in a small number of instances which in turn leads to poor embedding quality. Consequently, we recommend using Annoy or k-d trees together with manifold learning algorithms. The comparison framework could also be extended to other manifold learning algorithms with neighborhood graph construction or approximate nearest neighbor searching methods.

We also explore the electricity usage patterns of different time periods and households in the Irish smart meter dataset. In this case, the elements of the manifold in question are probability distributions. Here we propose a solution that exploits the connection between the Hellinger and Total Variation Metric used to describe the distance between discrete probability distributions and the L2 and L1 norm where the vectors are values of probability mass functions. In doing so, rather than compute all pairwise Hellinger or Total Variation distances, k-d trees and Annoy can be used to reduce computational time. Once again, we show that ANN techniques can be used within manifold learning to save computation time and memory without having a severe impact on the quality of the low-dimensional representations. For the particular dataset, ISOMAP or UMAP together with k-d trees gives the best tradeoff between embedding quality and computational time, while Annoy breaks down for one example that is particularly high-dimensional. Using the highest density region plots, we show how the techniques developed can successfully identify both typical and anomalous times of week and households.

There are several open questions to be explored. The first involves the selection of tuning parameters for the approximate nearest neighbor algorithm. The optimal choice may depend on a number of considerations, for instance, if manifold learning is used to detect anomalies in an online fashion, then the tuning parameters may be constrained in a way that approximate nearest neighbors are found within a certain time frame (Talagala et al. 2020). Alternatively, tuning parameters may be selected so that a maximal level of embedding quality is achieved, where embedding quality is measured using one of the metrics we discussed in Section 2.4.

Finally, there remain a number of open issues in manifold learning including the choice of intrinsic dimension d (Denti et al. 2021) and the choice of manifold learning algorithm itself. In the case of the former, we present results in Section 4 based only on $d = 2$ due to our emphasis on visualizations and the ease of using scatterplots. However, in principle, larger values of d could be used and if visualization is required, the embedding dimensions could be summarized using principal

components or using projection pursuit methods with random tours (Cook et al. 1995; Laa, Cook & Valencia 2020). Regarding the choice of the manifold learning algorithm, we have seen that this has a much bigger impact on accuracy measures than the use of approximate nearest neighbors. This does bring into focus the need for further work on the selection of manifold learning algorithms. However, it is an encouraging result for the use of ANN which across a wide range of algorithms improves the computational speed of manifold learning.

Acknowledgment

This research was supported in part by the Monash eResearch Centre and eSolutions-Research Support Services through the use of the MonARCH HPC Cluster. We would also like to acknowledge the support of the *Australian Research Council (ARC) Centre of Excellence for Mathematical and Statistical Frontiers (ACEMS)* for this project.

Supplementary Materials

The GitHub repository, `https://github.com/ffancheng/paper-mlann`, contains all materials required to reproduce this article. The code and data files are also available online in the same repository.

References

Amari, SI (Feb. 2016). *Information Geometry and Its Applications*. en. Springer.

Arya, S, DM Mount, NS Netanyahu, R Silverman & AY Wu (Nov. 1998). An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM* **45**(6), 891–923.

Aumüller, M, E Bernhardsson & A Faithfull (Jan. 2020). ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* **87**, 101374.

Banerjee, S, R Akbani & V Baladandayuthapani (2019). Spectral clustering via sparse graph structure learning with application to proteomic signaling networks in cancer. *Computational Statistics & Data Analysis* **132**, 46–69.

Belkin, M & P Niyogi (June 2003). Laplacian Eigenmaps for Dimensionality Reduction and Data Representation. *Neural Comput.* **15**(6), 1373–1396.

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

Bernhardsson, E (2015). *ANN presentation*. https://github.com/erikbern/ann-presentation. Accessed on 2020-09-24.

Carter, KM, R Raich, WG Finn & AO Hero 3rd (Nov. 2009). FINE: fisher information nonparametric embedding. en. *IEEE Trans. Pattern Anal. Mach. Intell.* **31**(11), 2093–2098.

Cayton, L (2005). Algorithms for manifold learning. *Univ. of California at San Diego Tech. Rep* **12**(1-17), 1.

Chen, L & A Buja (Mar. 2009). Local Multidimensional Scaling for Nonlinear Dimension Reduction, Graph Drawing, and Proximity Analysis. *J. Am. Stat. Assoc.* **104**(485), 209–219.

Coifman, RR & S Lafon (July 2006). Diffusion maps. *Appl. Comput. Harmon. Anal.* **21**(1), 5–30.

Commission for Energy Regulation (CER) (2012). *CER Smart Metering Project - Electricity Customer Behaviour Trial, 2009-2010 [dataset]*. SN: 0012-00.

Cook, D, A Buja, J Cabrera & C Hurley (Sept. 1995). Grand Tour and Projection Pursuit. *J. Comput. Graph. Stat.* **4**(3), 155–172.

Denti, F, D Doimo, A Laio & A Mira (Apr. 2021). Distributional Results for Model-Based Intrinsic Dimension Estimators. arXiv: 2104.13832 [stat.ME].

Dijkstra, EW (Dec. 1959). A note on two problems in connexion with graphs. *Numer. Math.* **1**(1), 269–271.

Dong, W, C Moses & K Li (Mar. 2011). Efficient k-nearest neighbor graph construction for generic similarity measures. In: *Proceedings of the 20th international conference on World wide web*. WWW '11. Hyderabad, India: Association for Computing Machinery, pp.577–586.

Donoho, DL & C Grimes (May 2003). Hessian eigenmaps: locally linear embedding techniques for high-dimensional data. en. *Proc. Natl. Acad. Sci. U. S. A.* **100**(10), 5591–5596.

Floyd, RW (June 1962). Algorithm 97: Shortest path. *Commun. ACM* **5**(6), 345.

Friedman, JH, JL Bentley & RA Finkel (Sept. 1977). An Algorithm for Finding Best Matches in Logarithmic Expected Time. *ACM Trans. Math. Softw.* **3**(3), 209–226.

Goldberg, Y & Y Ritov (Oct. 2009). Local procrustes for manifold embedding: a measure of embedding quality and embedding algorithms. *Mach. Learn.* **77**(1), 1–25.

Hellinger, E (1909). Neue Begründung der Theorie quadratischer Formen von unendlichvielen Veränderlichen. *J. Reine Angew. Math.* (136), 210–271.

Hyndman, RJ, X Liu & P Pinson (2018). Visualizing big energy data: Solutions for this crucial component of data analysis. *IEEE Power Energ. Mag.*

Hyndman, RJ (1996). Computing and Graphing Highest Density Regions. *Am. Stat.* **50**(2), 120–126.

Izenman, AJ (Sept. 2012). Introduction to manifold learning. *WIREs Comp Stat* **4**(5), 439–446.

Kraemer, G, M Reichstein & MD Mahecha (2018). dimRed and coRanking—Unifying dimensionality reduction in R. *R J.* **10**(1), 342–358.

Kruskal, JB (Mar. 1964a). Multidimensional scaling by optimizing goodness of fit to a nonmetric hypothesis. *Psychometrika* **29**(1), 1–27.

Kruskal, JB (June 1964b). Nonmetric multidimensional scaling: A numerical method. en. *Psychometrika* **29**(2), 115–129.

Laa, U, D Cook & G Valencia (July 2020). A Slice Tour for Finding Hollowness in High-Dimensional Data. *J. Comput. Graph. Stat.* **29**(3), 681–687.

LeCam, L et al. (1973). Convergence of estimates under dimensionality restrictions. *Ann. Stat.* **1**(1), 38–53.

LeCun, Y, C Cortes & C Burges (2010). *MNIST handwritten digit database*.

Lee, J & M Verleysen (2008). Quality assessment of nonlinear dimensionality reduction based on K-ary neighborhoods. In: *Proceedings of the Workshop on New Challenges for Feature Selection in Data Mining and Knowledge Discovery at ECML/PKDD 2008*. Ed. by Y Saeys, H Liu, I Inza, L Wehenkel & Y Van de Pee. Vol. 4. Proceedings of Machine Learning Research. Antwerp, Belgium: PMLR, pp.21–35.

Lee, JA & M Verleysen (Oct. 2007). *Nonlinear Dimensionality Reduction*. en. Springer Science & Business Media.

Lee, SM, AL Abbott & PA Araman (2007). Dimensionality reduction and clustering on statistical manifolds. In: *2007 IEEE Conference on Computer Vision and Pattern Recognition*. ieeexplore.ieee.org, pp.1–7.

Lunga, D, S Prasad, MM Crawford & O Ersoy (Jan. 2014). Manifold-Learning-Based Feature Extraction for Classification of Hyperspectral Data: A Review of Advances in Manifold Learning. *IEEE Signal Process. Mag.* **31**(1), 55–66.

Malkov, YA & DA Yashunin (Apr. 2020). Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. en. *IEEE Trans. Pattern Anal. Mach. Intell.* **42**(4), 824–836.

McInnes, L, J Healy & J Melville (Feb. 2018). UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. arXiv: 1802.03426 [stat.ML].

McQueen, J, M Meilă, J VanderPlas & Z Zhang (2016). Megaman: Scalable Manifold Learning in Python. *J. Mach. Learn. Res.* **17**(148), 1–5.

Mount, D & S Arya (2010). *ANN: A Library for Approximate Nearest Neighbor Searching*. http://www.cs.umd.edu/~mount/ANN. Version 1.1.3. Accessed on 2020-09-24.

Mount, D, S Arya, SE Kemp, G Jefferis & K Mülle (2019). *Fast Nearest Neighbour Search (Wraps ANN Library) Using L2 Metric*. https://github.com/jefferislab/RANN. Accessed on 2020-09-24.

Muja, M & DG Lowe (2009). Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)* **2**(331-340), 2.

Nadler, B, S Lafon, I Kevrekidis & RR Coifman (2006). "Diffusion Maps, Spectral Clustering and Eigenfunctions of Fokker-Planck Operators". In: *Advances in Neural Information Processing Systems 18*. Ed. by Y Weiss, B Schölkopf & JC Platt. MIT Press, pp.955–962.

Roweis, ST & LK Saul (Dec. 2000). Nonlinear dimensionality reduction by locally linear embedding. en. *Science* **290**(5500), 2323–2326.

Shepard, RN (June 1962a). The analysis of proximities: Multidimensional scaling with an unknown distance function. I. *Psychometrika* **27**(2), 125–140.

Shepard, RN (Sept. 1962b). The analysis of proximities: Multidimensional scaling with an unknown distance function. II. *Psychometrika* **27**(3), 219–246.

Spotify (2016). *Annoy*. https://github.com/github/open-source-survey. Accessed on 2020-09-24.

Talagala, PD, RJ Hyndman, K Smith-Miles, S Kandanaarachchi & MA Muñoz (Jan. 2020). Anomaly Detection in Streaming Nonstationary Temporal Data. *J. Comput. Graph. Stat.* **29**(1), 13–27.

Tang, J, J Liu, M Zhang & Q Mei (Apr. 2016). Visualizing Large-scale and High-dimensional Data. In: *Proceedings of the 25th International Conference on World Wide Web*. WWW '16. Montréal, Québec, Canada: International World Wide Web Conferences Steering Committee, pp.287–297.

Tenenbaum, JB, V de Silva & JC Langford (Dec. 2000). A global geometric framework for nonlinear dimensionality reduction. en. *Science* **290**(5500), 2319–2323.

Van Der Maaten, L (2014). Accelerating t-SNE using tree-based algorithms. *Journal of Machine Learning Research* **15**(1), 3221–3245.

Van der Maaten, L & G Hinton (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research* **9**(Nov), 2579–2605.

Venna, J & S Kaski (July 2006). Local multidimensional scaling. en. *Neural Netw.* **19**(6-7), 889–899.

Weinberger, KQ & LK Saul (Oct. 2006). Unsupervised Learning of Image Manifolds by Semidefinite Programming. *International Journal of Computer Vision* **70**(1), 77–90.

Zhang, Z & H Zha (2003). Nonlinear Dimension Reduction via Local Tangent Space Alignment. In: *Intelligent Data Engineering and Automated Learning*. Springer Berlin Heidelberg, pp.477–481.

Zhu, B, JZ Liu, SF Cauley, BR Rosen & MS Rosen (Mar. 2018). Image reconstruction by domain-transform manifold learning. en. *Nature* **555**(7697), 487–492.

A Appendix

A.1 Pseudocode for manifold learning algorithms

This section provides the pseudocode for the four manifold learning algorithms in Section 2.2.

Algorithm 1: ISOMAP

Input : high-dimensional data x_i for all $i = 1, \ldots, N$
Output: low-dimensional embedding y_i for all $i = 1, \ldots, N$

1 Construct the K-nearest neighbor graph $\mathcal{G}$ for x_i where $i = 1, \ldots, N$;
2 Set edge weights to δ_{ij} for x_i and x_j connected by an edge in $\mathcal{G}$;
3 Estimate the geodesic distances (distances along a manifold) between all pairs of points as the shortest-path distances on the graph $\mathcal{G}$;
4 Using the estimates of the geodesic distances as inputs into classical MDS, obtain the output points y_i for $i = 1, \ldots, N$.

Algorithm 2: LLE

Input : high-dimensional data x_i for all $i = 1, \ldots, N$
Output: low-dimensional embedding y_i for all $i = 1, \ldots, N$

1 Construct the K-nearest neighbor graph $\mathcal{G}$ to find the K-ary neighborhood $U_K(i)$ of x_i for all $i = 1, \ldots, N$;
2 Find a representation of each input point x_i as a weighted and convex combination of its nearest neighbors by minimizing

$$\left\| x_i - \sum_{j \in U_K(i)} w_{ij} x_j \right\|^2$$

with respect to weights w_{ij};
3 Find a configuration that minimizes

$$\sum_i \left\| y_i - \sum_{j \in U_K(i)} w_{ij} y_j \right\|^2$$

with respect to $y_1, \ldots, y_n$ where w_{ij} are identical to those found in the previous step.

Algorithm 3: Laplacian Eigenmaps

Input : high-dimensional data x_i for all $i = 1, \ldots, N$
Output: low-dimensional embedding y_i for all $i = 1, \ldots, N$

1 Construct the K-nearest neighborhood graph $\mathcal{G}$ for input points x_i for all $i = 1, \ldots, N$;
2 For input points x_i and x_j that are connected by an edge on $\mathcal{G}$, compute weights as $w_{ij} = 1$. Alternatively, set weights using the heat kernel $w_{ij} = e^{-\|x_i - x_j\|^2 / 2\sigma^2}$. For input points x_i and x_j that are not connected by an edge on $\mathcal{G}$, set $w_{ij} = 0$;
3 Compute the graph Laplacian as $L := D - W$, where W has elements w_{ij} and D is the diagonal matrix with elements $D_{ii} = \sum_j w_{ij}$;
4 Solve the generalized eigendecomposition equation $Lv = \lambda Dv$, where λ and v are the eigenvalue and eigenvector, respectively. The output points are obtained as the eigenvectors corresponding to the smallest non-zero eigenvalues.

Algorithm 4: Hessian LLE

Input : high-dimensional data x_i for all $i = 1, \ldots, N$
Output: low-dimensional embedding y_i for all $i = 1, \ldots, N$

1: Construct the K-nearest neighborhood graph $\mathcal{G}$;
2: **for** $\ell = 1, \ldots, N$ **do**
3: Estimate the tangent space at x_ℓ by stacking all $x_j : j \in U_K(\ell)$ as rows in a $K \times p$ matrix and take a singular value decomposition. A basis for the tangent space is determined by the first d left singular vectors (i.e. a $K \times d$ matrix);
4: Form a $K \times d(d+1)/2$ matrix, Z, by augmenting the d left singular vectors from the previous step with a column of ones and with element-wise squares and cross-products of the d left singular vectors;
5: Conduct a Gram-Schmidt orthogonalization on Z and transpose the result;
6: Form the $d(d+1)/2 \times K$ matrix H^ℓ by picking out the rows of the matrix from the previous step that correspond to the squared terms and cross products. Pre-multiplying a vector composed of elements $f(x_j) : j \in U_K(\ell)$ by H^ℓ yields an estimate of the elements in the Hessian at x_ℓ;
7: **end for**
8: Compute an $N \times N$ discrete approximation of $\mathcal{H}(f)$ with entries

$$\mathcal{H}_{i,j} = \sum_\ell \sum_r (H^\ell_{r,c(i)} H^\ell_{r,c(j)}),$$

where $c(i)$ and $c(j)$ are the columns of H^ℓ corresponding to observation i and j, and r corresponds to an element of the Hessian;
9: Perform an eigendecomposition on the matrix from the previous step to approximate the null space of $\mathcal{H}$. This will have $d+1$ zero (or near zero) eigenvalues, the smallest of which corresponds to a constant f. The eigenvectors corresponding to the next d smallest eigenvalues yield the output points.

Algorithm 5: t-SNE

Input : high-dimensional data x_i for all $i = 1, \ldots, N$
Output : low-dimensional embedding y_i for all $i = 1, \ldots, N$
parameter : *perplexity*
optimization parameter: number of iterations T, learning rate η, momentum $\alpha(t)$

1: Compute the Gaussian conditional probability of distances for x_i and x_j as $p_{j|i}$ using

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

where the variance σ_i for each x_i optimized with the perplexity parameter by $2^{-\sum_j p_{j|i} \log_2 p_{j|i}} = \textit{perplexity}$, and set $p_{i|i} = 0$;

2: Set the probability as $p_{ij} = (p_{j|i} + p_{i|j})/(2N)$;

3: Initialize the optimization solution as $\mathcal{Y}^{(0)} = \{y_1, \ldots, y_N\}$ and $\mathcal{Y}^{(1)}$ sampled from normal distribution $\mathrm{N}(0, 10^{-4}\boldsymbol{I})$;

4: **for** $t = 1, \ldots, T$ **do**

5: Compute the Student t-distribution of distances between y_i and y_j as

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

6: Compute the gradient of the Kullback-Leibler divergence loss function for all $i = 1, \ldots, N$ as

$$\frac{\delta KL}{\delta y_i} = 4 \sum_j \left(p_{ij} - q_{ij}\right)\left(y_i - y_j\right)\left(1 + \left\|y_i - y_j\right\|^2\right)^{-1};$$

7: Update $\mathcal{Y}^{(t)} = \mathcal{Y}^{(t-1)} + \eta \frac{\delta KL}{\delta \mathcal{Y}} + \alpha(t)\left(\mathcal{Y}^{(t-1)} - \mathcal{Y}^{(t-2)}\right)$.

8: **end for**

Algorithm 6: UMAP

Input : high-dimensional data x_i for all $i = 1, \ldots, N$
Output : low-dimensional embedding y_i for all $i = 1, \ldots, N$
parameter: number of nearest neighbors K, minimum distance between embedded points *min_dist*, amount of optimization work *n_epochs*

1: Construct the K-nearest neighborhood graph to find the K-ary neighborhood $U_K(i)$ with distances δ_{ij}, and denote the distance to its nearest neighbor as ρ_i for each x_i;
2: For the neighborhood points in $U_K(i)$, apply a smooth approximator to δ_{ij} to search for σ_i such that $\sum_i^K e^{\frac{-\max(0,\delta_{ij}-\rho_i)}{\sigma_i}} = \log_2 K$;
3: Compute the exponential probability of distances between x_i and x_j as $p_{j|i}$ using

$$p_{i|j} = e^{\frac{-\max(0,\delta_{ij}-\rho_i)}{\sigma_i}};$$

4: Set $p_{ij} = p_{j|i} + p_{i|j} - p_{i|j}p_{j|i}$;
5: Define the distribution of distances between y_i and y_j as

$$q_{ij} = \left(1 + a\left(y_i - y_j\right)^{2b}\right)^{-1},$$

where a and b are found using the *min_dist* parameter such that

$$q_{ij} \approx \begin{cases} 1 & \text{if } y_i - y_j \leq min\_dist \\ e^{-\left(y_i - y_j\right) - min\_dist} & \text{if } y_i - y_j > min\_dist; \end{cases}$$

6: The cost function is constructed using a binary fuzzy set cross entropy (CE) as

$$CE(X, Y) = \sum_i \sum_j \left[p_{ij}(X) \log\left(\frac{p_{ij}(X)}{q_{ij}(Y)}\right) + \left(1 - p_{ij}(X)\right) \log\left(\frac{1 - p_{ij}(X)}{1 - q_{ij}(Y)}\right) \right]$$

7: Initialize the embedding coordinates y_i using the graph Laplacian with weights as p_{ij}, similar to Laplacian Eigenmaps in Algorithm 3;
8: Optimize y_i by minimizing the cross entropy using stochastic gradient descent with *n_epochs*.

A.2 Pseudocode for approximate nearest neighbor searching methods

The pseudocode for three approximate nearest neighbor searching methods, k-d trees, Annoy, and HNSW, are included in the following section.

The k-d trees method involves two steps: k-d tree construction in Algorithm 7 and recursive searching in Algorithm 8. Annoy also consists of two steps: preprocess in Algorithm 9 and query in Algorithm 10. Finally, a general HNSW greedy searching process is shown in Algorithm 11.

Algorithm 7: Constructing a k-d tree

Initialize the tree $depth = 0$ and the node index $g = 0$;
Initialize the root node $\mathcal{P}_g$ as $\mathbb{R}^p$;
Initialize the set of all nodes at a depth of zero as $\mathrm{C}_0 = \{\mathcal{P}_g\}$;
while $|\{x_i : x_i \in \mathcal{P}_h\}| > 1$ for some $\mathcal{P}_h \in \mathrm{C}_{depth}$ **do**
  Set $depth \leftarrow depth + 1$;
  Initialize $\mathrm{C}_{depth} = \varnothing$;
  for $\mathcal{P}_h \in \mathrm{C}_{depth-1}$ **do**
    Find splitting dimension $\ell_h^* = \underset{\ell}{\operatorname{argmax}} \max_{i,j \in \mathcal{P}_h} |x_{i\ell} - x_{j\ell}|$;
    Find splitting value $c_h^* = \left(x_{\ell_h^*}^{(v+1)} - x_{\ell_h^*}^{(v)}\right)/2$ where $x_\ell^{(r)}$ denotes r^{th} order statistic of $\{x_i : x_i \in \mathcal{P}_h\}$ along dimension ℓ;
    Set the partition for the left child node $\mathcal{P}_h^{left}$ as the set of points $\{z : z \in \mathcal{P}_h, z_{\ell^*} < c^*\}$ where z_{ℓ^*} is the value of z on the splitting dimension;
    Set the partition for the right child node $\mathcal{P}_h^{right}$ as the set of points $\{z : z \in \mathcal{P}_h, z_{\ell^*} \geq c^*\}$;
    Set $\mathcal{P}_{g+1} := \mathcal{P}_h^{left}$ and $\mathcal{P}_{g+2} := \mathcal{P}_h^{right}$;
    Update $\mathrm{C}_{depth} \leftarrow \mathrm{C}_{depth} \cup \{\mathcal{P}_{g+1}, \mathcal{P}_{g+2}\}$;
    Update $g \leftarrow g + 2$;
  end for
end while

Algorithm 8: Recursive procedure `search`($\mathcal{P}_g$) for nearest neighbor searching.

if $\mathcal{P}_g$ is a terminal node **then**
  Compute distance δ_{iq} from query x_q to $x_i \in \mathcal{P}_g$;
  if $\delta_{iq} < \delta^*$ **then**
    Set $\delta^* \leftarrow \delta$ and $x_{q^*} \leftarrow x_i$;
  end if
  return
else
  For splitting dimension l_g^* and splitting value c_g^*,
  if $x_{ql_g^*} < c_g^*$ **then**
    Evaluate `search`($\mathcal{P}_g^{left}$);
    Find hyper-sphere $\mathcal{S}$ of radius δ^* and tightest bounding box $\mathcal{B}$ of $\{x_i : x_i \in \mathcal{P}_g^{right}\}$;
    if $\mathcal{S}$ and $\mathcal{B}$ intersect **then**
      Evaluate `search`($\mathcal{P}_g^{right}$);
    end if
    return
  else
    Evaluate `search`($\mathcal{P}_g^{right}$);
    Find hyper-sphere $\mathcal{S}$ of radius δ^* and tightest bounding box $\mathcal{B}$ of $\{x_i : x_i \in \mathcal{P}_g^{left}\}$;
    if $\mathcal{S}$ and $\mathcal{B}$ intersect **then**
      Evaluate `search`($\mathcal{P}_g^{left}$);
    end if
    return
  end if
end if

Algorithm 9: Annoy preprocess

Initialize the node index $g = 0$ and tree $depth = 0$;
Initialize each root node as $\mathcal{P}_{0,t} = \mathbb{R}^p$ for $t = 1, \ldots, n\_trees$;
Initialize $\mathrm{C}_{0,t} = \{\mathcal{P}_{0,t}\}$ for $t = 1, \ldots, n\_trees$;
for $t = 1, \ldots, n\_trees$ **do**
  while $|\{x_i : x_i \in \mathcal{P}_{h,t}\}| > \kappa$ for some $\mathcal{P}_{g,t} \in \mathrm{C}_{depth,t}$ **do**
    Set $depth \leftarrow depth + 1$
    Initialize $\mathrm{C}_{depth,t} = \varnothing$
    for $\mathcal{P}_{h,t} \in \mathrm{C}_{depth-1,t}$ **do**
      Randomly select two points $x_i, x_j \in \mathcal{P}_{h,t}$;
      Find the hyperplane $\mathcal{S}_{h,t}$ equidistant from x_i and x_j;
      Set the left child node partition $\mathcal{P}_{h,t}^{left}$ as all points $\{z : z \in \mathcal{P}_{h,t}, \delta_{zi} < \delta_{zj}\}$ where δ_{zi} and δ_{zj} are the distances from z to x_i and x_j respectively;
      Set the right child node partition $\mathcal{P}_{h,t}^{right}$ as the set of all points $\{z : z \in \mathrm{C}_g, \delta_{zi} \geq \delta_{zj}\}$;
      Update $\mathcal{P}_{g+1,t} := \mathcal{P}_{g,t}^{left}$ and $\mathcal{P}_{g+2,t} := \mathcal{P}_{g,t}^{right}$;
      Update $\mathrm{C}_{depth,t} \leftarrow \mathrm{C}_{depth,t} \cup \{\mathcal{P}_{g+1,t}, \mathcal{P}_{g+2,t}\}$
      Update $g \leftarrow g + 2$;
    end for
  end while
end for

Algorithm 10: Annoy query

Letting, $S_{0,t}$ be the hyperplane splitting the root node of tree t, find $\mathcal{P}_{0,t}^{q}$ and $\mathcal{P}_{0,t}^{-q}$ where the former is on the same side of $S_{0,t}$ as x_q latter is on the opposite side of $S_{0,t}$ from x_q;
Set $C_0 \leftarrow \{\mathcal{P}_{0,t}^{q}\}$ for $t = 1, \ldots, n\_trees$;
Set up a priority queue $\mathcal{Q}$ made up of pairs $(\mathcal{P}_g, \delta_g)$ where $\mathcal{P}_g$ is a partition of space (initialized at $\mathcal{P}_{0,t}^{-q}$ for all t) and δ_g are the distances from x_q to the hyperplane corresponding to each $\mathcal{P}_g$;
Set $depth \leftarrow 0$;
while alsot least one element of $C_{depth} \cup \mathcal{Q}_{depth}$ does not correspond to a leaf node **do**
  for $\mathcal{P}_g \in \mathcal{Q}$ **do**
    Find $\mathcal{P}_g^q$ and $\mathcal{P}_g^{-q}$ where these are defined in a similar fashion to Step 1;
    Replace $\mathcal{P}_g$with $\mathcal{P}_g^q$ in the priority queue;
    Add $\mathcal{P}_g^{-q}$ to the priority queue;
    if $|\mathcal{Q}| > maxsize\_queue$ **then**
      Remove $\mathcal{Q}_{maxsize\_queue}$ from $\mathcal{Q}$;
    end if
  end for
  Set $depth \leftarrow depth + 1$, $C_{depth} \leftarrow \varnothing$;
  for $\mathcal{P}_h \in C_{depth-1}$ **do**
    Find $\mathcal{P}_h^q$ and $\mathcal{P}_h^{-q}$ where these are defined in a similar fashion to Step 1;
    Set $C_{depth} \leftarrow C_{depth} \cup \mathcal{P}_h^q$;
    Add $\mathcal{P}_h^{-q}$ to $\mathcal{Q}$, removing elements in the same fashion as Steps 10-12 if needed;
  end for
end while
Set candidate set $\mathcal{K} \leftarrow \bigcup_{\mathcal{P}_h \in C_{depth}} x_i \in \mathcal{P}_h$;
Set $g \leftarrow 0$;
while $|\mathcal{K}| < search\_k$ **do**
  Set $g \leftarrow g + 1$;
  Set $\mathcal{K} \leftarrow \mathcal{K} \cup \{x_i : x_i \in \mathcal{P}_g\}$ where $\mathcal{P}_g$ is the g^{th} element of the priority queue;
end while
Search for nearest neighbors among the elements of $\mathcal{K}$ by brute force;

Algorithm 11: Hierarchical Navigable Small World graphs (HNSW)

For a query point x_q, randomly select an entry point x_e;
Initialize $x_b \leftarrow x_e$, $\delta_{qb} = 0$ and δ_{qj^*} as the distance between x_q and x_b;
while $\delta_{qb} < \delta_{qj^*}$ **do**
  Set $\delta_{qb} \leftarrow \delta_{qj^*}$;
  Update $\delta_{qj^*} = \min_{j \in N(b)} \delta_{qj}$ where $N(b)$ is a set of indices for points connected to x_b by exactly one edge. Set x_{j*} to the value of x_j that achieves this minimum;
  Set $x_b \leftarrow x_{j^*}$;
end while
return x_b