Modern Dimension Reduction Techniques

Examining t-SNE and UMAP

Daniel Ralston

Dec 21, 2020

1 Introduction

Often it is useful to visualize a data set that has more than three attributes. We wish to reduce the number of dimensions in a data set, normally so that it has less than or equal to three attributes so it can be visualized. While there are many different algorithms that can complete such a task, for much of the past decade, the t-SNE algorithm (2008) has been the leading algorithm in nonlinear[1] dimension reduction. More recently, the Uniform Manifold Approximation Projection (UMAP) algorithm (2018) has proven to outperform t-SNE and has quickly gained popularity.[14] In this report, we begin by discussing the mathematics behind t-SNE and UMAP, in sections 2 and 3 respectively. While each paper uses very different terminology, throughout section 3 we make analogies between the two to display their similarities. In section 4, we compare the two algorithms and highlight UMAP's excellent performance. In section 5, we detail our results of experimenting with the algorithms in Python. Finally, we conclude with section 6. Sections 2 and 3 make the paper slightly front loaded, because I was interested in the inner workings of each algorithm, and why each was presented so differently. Still, all parts of the project were very enjoyable!

1.1 Notation

Throughout this paper, we will use the following notation (with a few exceptions as noted). Some of this notation is borrowed from the t-SNE and UMAP papers. [12][15]

- D is a given data set
- s is the number of entries (or the size) of D
- n is the ambient dimension (the number of attributes) of D
- d is the embedding dimension, or the number of dimensions our data D is being reduced to (we will use $\mathbb{R}^d$ to describe this embedding space)
- x_i represents the ith data point (or the ith entry) in D
- y_i is the point in the d dimensional embedding, corresponding high dimensional data point x_i
- G_H is a mathematical graph that represents the high dimensional data from D (further described in the UMAP section)
- G_L is a mathematical graph that represents the low dimensional embedding of the high dimensional data from D (also further described in the UMAP section)

[1] The method uses nonlinear techniques to describe distance.

2 t-SNE

Proposed by Laurens van der Maaten and Geoffrey Hinton, t-SNE builds on the stochastic neighbor embedding (SNE) method which was introduced by Hinton and Roweis in 2002.[9] We will discuss this portion of algorithm, before discussing the specifics which distinguish t-SNE from its previous counterparts. Furthermore, we present the algorithm from a probabilistic perspective, as it was outlined in the original paper.[15] However, as discussed in the Analysis and Comparison section, t-SNE (like UMAP) can also be understood as a graph-based algorithm. [4]

2.1 Overview

t-SNE attempts to maintain local structure of the data using the stochastic neighbor embedding process. The first step in the t-SNE algorithm is to convert the Euclidean distance between all points x_i in D to probabilities that represent the probability of x_i choosing its nearest neighbor, i.e. if the Euclidean distance between two points x_i and x_j is high, the probability that x_i chooses x_j as its nearest neighbor should be low. Next, the algorithm initializes the embedding space by distributing each point y_i around the origin according to a Student-t probability distribution – hence the t in t-SNE. Just like the in the high dimensional space, t-SNE then calculates the Euclidean distance between all points y_i and converts each distance to a probability. Finally, Maaten and Hinton use an objective (or cost) function that measures the difference between the probability of the closeness between every possible pair points x_i and x_j and their low dimensional counterparts y_i and y_j. The algorithm minimizes this objective function. In practice, this involves rearranging the low dimensional points so that for every pair of points the probability that y_i and y_j are close reflects the probability that x_i and x_j are close in D. This creates a visualization which preserves neighboring points.

2.2 Similarity Scores

The authors begin by defining the *similarity* of two data points x_i and x_j as $p_{j|i}$, which is the conditional probability that for a given data point x_i, x_i chooses x_j as it's neighbor. Mathematically defined, we have

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

To break down this definition, first note that the Euclidean distance between the two points is given by $||x_j - x_i||$. Next, consider the numerator of $p_{j|i}$ which looks "almost"[2] like a Gaussian centered at a mean of x_i. Intuitively, the numerator centers this "almost" Gaussian distribution over a given point x_i, which non-linearizes the Euclidean distance from x_i to every other data point.[3] Because we want to interpret each $p_{j|i}$ as a probability, the denominator normalizes each term so that

$$\sum_{j \neq i} p_{j|i} = 1.$$

Notice that the variance σ_i is dependent on each data point x_i. Rather than choosing a fixed variance as a parameter for all points to non linearize the Euclidean distance between points, this dependence arises

[2] "Almost", because the term is missing the scalar $\frac{1}{\sigma\sqrt{2\pi}}$. The authors ignore this constant because the normalization of $p_{j|i}$ cancels constant terms:

$$p_{j|i} = \frac{exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} exp(-||x_i - x_k||^2/2\sigma_i^2)} = \frac{\frac{1}{\sigma\sqrt{2\pi}} \cdot exp(-||x_i - x_j||^2/2\sigma_i^2)}{\sum_{k \neq i} \frac{1}{\sigma\sqrt{2\pi}} \cdot exp(-||x_i - x_k||^2/2\sigma_i^2)}.$$

because the distribution of points in D is unlikely to be uniform. A data set may have both dense and disperse clusters, and since variance is related to the number of points that fall within a certain distance of the mean, a model employing a constant variance may overcrowd its projection of dense clusters and fail to project a disperse cluster as a cluster. Allowing variance to change depending on our choice of x_i is what allows the algorithm to project local structure of the data – if variance was constant, our projection would be a global interpretation of the density of our data set.

2.3 Variance and Perplexity

For each data point x_i, choosing such a variance σ_i is complex. Maaten and Hinton introduce the parameter *perplexity* to the algorithm (denoted as $Perp$) and provide both intuitive and mathematical definitions. Intuitively, the perplexity is defined as "the smooth measure of the effective number of neighbors[3]." Mathematically,

$$Perp(P_i) = 2^{\sum_j (p_{j|i}) \log_2(1/p_{j|i})}$$

where P_i is the probability distribution derived from each $p_{j|i}$ for a fixed i. We will attempt to show the connection between the intuitive and mathematical definitions of perplexity, although a full mathematical explanation is well beyond my understanding of information theory. Still, perplexity is the most important parameter in the t-SNE algorithm, and its underpinnings deserve description. The term $\sum_j (p_{j|i}) \log_2(1/p_{j|i})$ is the Shannon entropy of the probability distribution P_i. The term $\log_2(1/p_{j|i})$ is the self-information of the similarity $p_{j|i}$, which can be interpreted as the number of bits (hence the $\log_2$) needed to encode all similarities $p_{j|i}$ for a fixed i. Multiplying this value by $p_{j|i}$ and summing over j comes from the expected value formula. Putting this all together, the Shannon entropy can be interpreted as the expected value of the self-information of a given similarity.[17] We raise 2 to this power to convert back to digits. As the expected amount of information increases for a given i, this implies the density of the points surrounding the point x_i must increase since the self information of each similarity is inversely proportional to each similarity $p_{j|i}$. In the implementation of t-SNE, after a user specifies a value for this perplexity parameter, at each point x_i a binary search[4] is performed to find the optimal value of σ_i which produces this value of perplexity.

2.4 Symmetrization

There are a few important notes before proceeding. First, note that because our variance depends on the ith point, given that D is unlikely to be uniformly distributed, it is therefore unlikely that $p_{j|i} = p_{i|j}$. While this is not necessarily a problem, in order to improve minimization of cost function, the Maaten and Hinton set

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

the average of the two similarities. This average is described as symetricising the probabilities.[5] Again, in order to interpret p_{ij} as a probability, we normalize each term p_{ij} so that we have $\sum_{j \neq i} p_{ij} = 1$.

[3] "smooth" here means that while nearest number of neighbors is normally a positive integer, perplexity can be any positive real number [normally ranging from 1 to 50]

[4] While this may seem like an expensive calculation that must be done at each x_i, this is actually a minor computational expense.[14]

[5] This is the first difference (admittedly an unmotivated difference) between t-SNE and the SNE algorithm proposed by Hinton and Roweis in 2002.

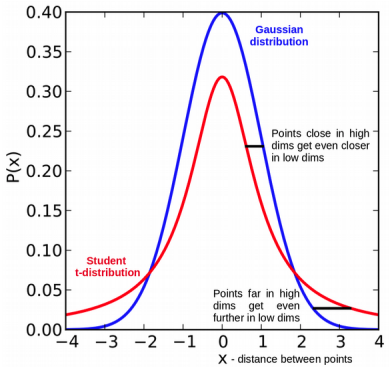


Figure 1: The "heavier tails" means that larger distances are represented probabilistically higher.[13]

2.5 Initializing the Lower Dimensional Embedding

The algorithm must project each point x_i to its corresponding representation y_i in the d dimensional embedding. Each point y_i is initialized from randomly sampling a Student-t probability distribution[6] centered at the origin in $\mathbb{R}^d$. The reason for this is to further space out points in order to combat the crowding problem. The crowding problem refers to the fact that in higher dimensions, space expands. For example, a sphere with radius r has a volume that scales with r^m where m is the dimension. Thus, when projecting into lower dimensions, data points can easily crowd, making it difficult to distinguish clusters. The authors chose a Student-t distribution because it reflects a slightly more linear relationship between probability and positive distance from its mean. In other words, for a given similarity score, we desire that points y_i and y_j are further apart than on a Gaussian distribution curve. This amounts to what the authors describe as "heavier tails", as shown in Figure 1. The Student-t distribution reflects this property.

Mathematically, distance between low dimensional points y_i is analogous to the high dimensional case. Consider defining the similarity $q_{j|i}$ between $y_i, y_j \in \mathbb{R}^d$ as

$$q_{j|i} = \frac{\frac{1}{1+||y_i - y_j||^2}}{\sum_{k \neq l} \frac{1}{1+||y_k - y_l||^2}}.$$

In this case the term $\frac{1}{1+||y_i-y_j||^2}$ corresponds to non linearizing the Euclidean distance from y_i to all other y_j by centering a Student-t distribution with a variance of one over y_i. The denominator, as in the high dimensional case, normalizes each similarity score so that we may interpret all $q_{j|i}$ with the probability density function Q_i. Additionally, because variance is constant, no symmetrization is needed, and thus $q_{j|i} = q_{i|j} = q_{ij}$.

2.6 Creating the Visualization

We have determined two conditional probabilities (similarities) from non-linearizing distance. Consider $H = \{P_i | 1 \leqslant i \leqslant s\}$, the set of all similarities p_{ij}, and $L = \{P_i | 1 \leqslant i \leqslant s\}$, the set of all similarities q_{ij}. Ideally, for all unordered pairs i, j (with $1 \leqslant i \leqslant s$, $1 \leqslant j \leqslant s$), we want we want $p_{ji} \approx q_{ji}$. A cost function that captures the difference between two such sets is the KL-Divergence (KL) function. From here on out, updating all points y_i to minimize this objective function is a standard machine learning task, and the authors employ the gradient descent method.[7] Because KL-Divergence and gradient descent methods

[6] This is the second of the two main differences between t-SNE and the SNE model, which used a Gaussian distribution.

[7] This method calculates the gradient vector ∇KL at a given point on the cost function, and then takes steps in the direction of $-\nabla KL$.

are canonical in machine learning, we will exclude those topics from our discussion. Still, a brief comparison of KL-Divergence and gradient descent methods with the methods UMAP uses will be made in the Analysis and Comparison section.

2.7 Comments

The first step in this algorithm is to convert Euclidean distance into a probability. Throughout our class, we haven't talked about how one could represent distance with probability. In part, this is because most probability density functions are nonlinear, meaning that the distance between points will become highly nonlinear, making calculations difficult. Really, we would rather have our algorithm simply be an isometry that could perfectly preserve distance between points. As I understand, converting distance into a probability attempts to replicate an isometry in this case, since it allows for a map between the two spaces that can be optimized in order to somewhat preserve distance.

3 UMAP

While very similar to t-SNE, the Uniform Manifold Approximation and Projection algorithm (UMAP) is explained as a graph-based approach, and is motivated with theoretically advanced mathematical machinery. We will explain it in the context the paper presents the material, but we will also make comparisons to t-SNE in order to display the two algorithms' similarities.

3.1 Overview

Relying on some sophisticated manifold theory, the authors of the UMAP paper begin by noting that all high dimensional data points x_i can be thought of as points on the surface of some $n-1$ manifold in $\mathbb{R}^n$. With this frame of thinking, the objective of the algorithm is to represent this high dimensional manifold as accurately as possible in $\mathbb{R}^d$. The best approximation of our manifold would occur if our points were uniformly distributed, or in other words, the union of unit balls B around each data point x_i would provide a cover C of the manifold[8]. Unfortunately, real world data is unlikely to be uniformly distributed. In order to solve this problem, the authors define a Riemannian metric on the manifold, which changes at each point x_i. With this varying way of measuring distance, we can assume the data uniformly distributed. In other words, because with a global metric the distance between points in D is probably not the same, in order to make the distance between points the same, the notion of distance must vary from point to point. This varying metric serves the same purpose of letting σ_i vary for each point x_i in t-SNE. Next, the cover C of our data set can be understood combinatorially as a edge-weighted undirected graph, where each edge corresponds to a probabilistic measure derived from our varying Riemannian metric; edges are analogous to the similarities $p_{j|i}$ in t-SNE. Finally, a similar low dimensional graph is assembled analogously to initializing the q_{ji} terms in t-SNE. Cross entropy assigned as the cost function for UMAP, and stochastic gradient descent is used to minimize the difference between the high and low dimensional graph representations.

[8]Consider that we would get a better measurement of the topography of Maine if we got our data from a lattice of lat/long coordinates (relatively uniformly distributed), rather than as measured at every house – we would have barely any data for northeastern Maine

3.2 Geometrical Foundations

The first step in the algorithm is to calculate a Riemannian metric at each point x_i which represents the high dimensional manifold as uniformly distributed. In the original paper, Lemma 1 gives a precise definition of the Riemannian geodesic between two points p, q on a manifold:

$$\frac{1}{r} d_{\mathbb{R}^n}(p, q),$$

where $d_{\mathbb{R}^n}(p, q)$ is the original metric in the space, and r is the radius of the ball B_q centered at q. Additionally, B_q must have volume

$$\frac{\pi^{n/2}}{\Gamma(0.5n + 1)}.$$

This volumetric requirement is motivated by theoretic in Riemannian Geometry.[4] Notice that because n is the dimension of the data set D, this volume is constant. This is exactly where our assumption that our data is uniformly distributed is influencing our metric for the data, since we want a ball around each point to contain the same number amount of data points.[9] Not only is this a complicated description of the geodesic between two points, but we are also introduced to a new notion of a metric, one that is localized to each point x_i.

3.3 From Manifolds to Graphs

Representing all such discrete metric spaces probabilistically as a fuzzy simplicial structure is the critical link between the theory of UMAP and its implementation. Interestingly, our varying definition of distance is enough to capture the shape of the manifold. To convert between our cover of the data set and a graph-based approach which can be implemented by a computer, the authors take the nerve of the covering. If C is a covering made up of open balls B_i (where i is the index of each high dimensional data points), then the nerve of C is given by

$$N(C) = \{J | J \subset I \text{ and } \bigcap_{j \in J} U_j \neq \varnothing\}.[18]$$

In words, this means that the nerve is a collection of sets of indices, where each set J in $N(C)$ corresponds to the points around j for which the cover intersects with J. Thus, each element in the nerve corresponds to points which have covers which are connected. So, if say $\{1, 2, 3\} \in N(C)$, this would imply that the open covers of points x_1, x_2, and x_3 all intersect. If we simply connect all of these points, this becomes a simplicial[10] object, specifically a Cech complex. Roughly speaking, the Nerve Theorem states that the nerve of a cover is a sufficiently accurate representation of the manifold D lives on. Although the Cech complex is guaranteed to accurately approximate our data, computing the higher dimensional simplices becomes computationally cumbersome. For better computational speed, the authors use the Vietoris-Rips complex, which just constructs a 1- simplex if two balls intersect – if two balls don't intersect then each point is already a 0 simplex, a point.[6] This creates a fully connected graph G_H between points whose open balls intersect. The authors of UMAP justify the round-about approach to creating a graph by arguing their explanation provides motivation at each step, and gives a foundational understanding for why the specific graph was constructed.[11] Other benefits of this approach are discussed in the Analysis and Conclusion section.

[9] Rather than using volumes in Riemannian geometry, most graph based algorithms create a nearest neighbors k hyper parameter. As detailed in the UMAP documentation, k serves as a way of controlling the number of points given within a ball, thereby altering the radius of the ball. If much of the data set was encompassed in each ball around a given point, the discrete Riemannian metrics defined on each point will be representative largely of the global structure of the data; the opposite is true if only a few points exist within each ball. While the theory of defining a Riemannian metric does not account for the parameter k, the UMAP implementation does.

[10] A k dimensional simplex is called a k-simplex, which is formed by taking the convex hull of $k + 1$ independent points, as shown in Figure 2.

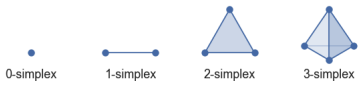


Figure 2: A point, line, triangle, and tetrahedron respectively. [11]

3.4 Graph Edges and Symmetrization

Consider an edge in G_H between points x_i and x_j. Because the definition of distance is highly localized to each point, as we are further on this edge away from x_i we are less confident of the accuracy of the metric defined at x_i. To represent such decay in confidence, the authors now define the weight from point x_i to point x_j as

$$w(x_i, x_j) = \exp(\frac{-\max(0, \phi(x_i, x_j) - \rho_i}{\gamma_i}),$$

where the function ϕ is the ambient metric in the high dimensional data. γ_i and ρ_i are derived from the Riemannian metric local to the point x_i. γ_i ensures that the value of each weight is greater than 0 and less than or equal to 1. ρ_i is the distance measured from x_i to its nearest neighbor using the metric ϕ. Subtracting this value ensures that the weight from x_i to its nearest neighbor is 1, since $exp(0) = 1$. In other words we are confident[11] that the metric at x_i still is locally relevant at its nearest neighbor point. This property is motivated by theoretical constraints, and is referred to as the property of "local connectivity," a surprisingly consequential characteristic that will be further discussed in later sections. The weight between two graph vertices x_i and x_j is directly analogous to the similarity $p_{j|i}$ between the two data points x_i and x_j.[12] This somewhat unusual function is derived from the description of a fuzzy topological representation.

Because γ_i and ρ_i are dependent on our choice of x_i, it is not always true that $w(x_i, x_j) = w(x_j, x_i)$, a property we desire.[13] In t-SNE, this symmetrization was done by taking the average of $p_{j|i}$ and $p_{i|j}$. UMAP uses a different method. In the UMAP graph analogy, w_{ji} represents a weighted edge from points x_i to x_j, symmetrization is equivalent to ensuring that only one edge exists between points. UMAP symmetrization is accomplished with

$$w_{ij} = w_{ji} = w(x_i, x_j) + w(x_j, x_i) - w(x_i, x_j) \cdot w(x_j, x_i).$$

If we consider that the edges are weights, this equation can be understood as the probability that either edge exists, but not both.[14]

3.5 Embedding, the Cost Function, and Optimization

To create a d dimensional embedding, the algorithm creates a weighted graph G_L in our low dimensional space using a similar procedure in the high dimensional case, with one key difference. In our projection we require that our results be globally interpretable, so our weight function for our d dimensional graph is defined with a constant metric not dependent on a given data point x_i – in practice this metric is normally Euclidean. Using math, the low dimensional weight function v is given by

$$v_{ij} = v(y_i, y_j) = \frac{1}{(1 + a(y_i - y_j)^{2b}}.[13]$$

[11] Although these weights are not probabilities, because their values are all between 0 and 1, an edge weight of 1 can be thought of as similar to a chance of 100%.

[12] This is exactly why Leland McInnes characterizes t-SNE as a graph based approach

[13] Currently, G_H is a directed graph, since there is a different weight associated with leaving and arriving at the vertex; we want an undirected graph, which is more computationally efficient.

[14] The probability of event A and event B both happening is given by $P(A) \cdot P(B)$, and the probability that event A or event B happens is $P(A) + P(B)$.

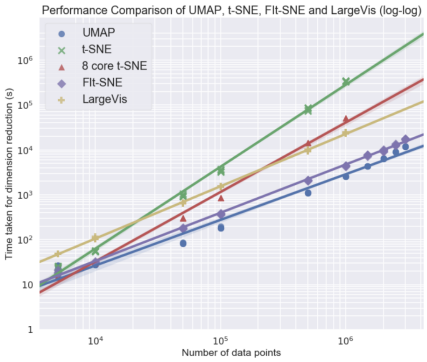


Figure 3: Standard UMAP implementation in dark blue, and standard t-SNE in light green.[12]

The constants $a, b \in \mathbb{R}$ are chosen so that each edge weight is greater than 0 and less than 1. The user's choice of the *minimum distance parameter*, which changes how close points are represented together in the low dimensional representation, also affects the choices of a and b. Notice because the metric is globally uniform, G_L is already undirected, since weights are naturally symmetric $v(y_i, y_j) = v(y_j, y_i)$. In the implementation of UMAP, a forced directed graph layout algorithm creates the low dimensional representation, the advantages of which are discussed in the Analysis and Comparison section. Finally, rather than KL-Divergence, the UMAP authors choose cross entropy (CE) as a cost function, which measures the difference between the corresponding weights in G_H and G_L. The authors employ a stochastic gradient descent method to minimize CE, which is very similar to the gradient descent method used in t-SNE, except that it is computationally faster by calculating the gradient using sub samples of the objective function. Like t-SNE, from here on out UMAP relies on standard machine learning techniques to efficiently update G_L to better represent G_H, in accordance with minimizing CE.

4 Analysis and Comparison

Although the UMAP algorithm undoubtedly provides better motivation, as was McInnis' aim, the implementation of the algorithm may be the source of its greatest strengths.[4] According to its authors, UMAP is preferable over t-SNE: first, UMAP is undoubtedly faster in every respect when compared to t-SNE; second, it better preserves the global structure of the underlying data.

4.1 UMAP's Faster Runtime

The most practical advantage of UMAP is it's significantly faster run time, which is favorable to t-SNE for large values of s, n, and d.

For a data set with over 10,000 entries, UMAP is noticeably faster than t-SNE. This is largely a weakness of t-SNE, rather than a strength UMAP, since UMAP is only just faster than comparable dimension reduction techniques as shown in Figure 3. Such inefficiency of t-SNE occurs from the normalization required in calculating each $p_{j|i}$, specifically the denominator term $\sum_{k \neq i} exp(-||x_k - x_i||^2/2\sigma_i^2)$. Thus, for a data set of size s, at every data point x_i, $s-1$ terms must be added together. Since there are s data points, this is roughly s^2 terms that must be added together in t-SNE. Because of UMAP's theoretical foundation, UMAP does not require this summation.

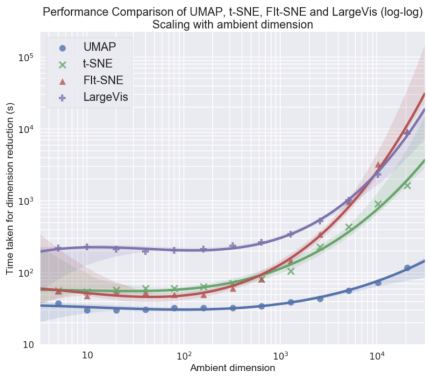


Figure 4: [12]

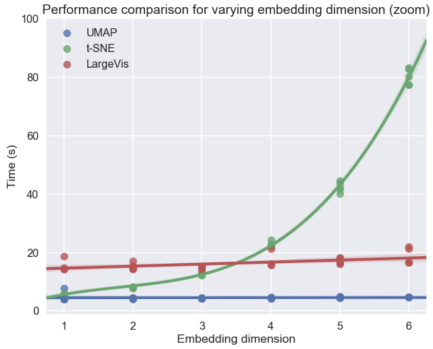


Figure 5: [12]

With respect to the dimension of the data set, UMAP is also much faster than competing nonlinear dimension reduction methods. Interestingly, UMAP's efficiency in this respect is mainly attributable to the assumption the manifold is always locally connected, which ensures that every vertex in the high dimensional graph has an edge leaving it. As it happens, graph-based optimization methods that are used in UMAPs implementation can perform faster on locally connected graphs. As we see in Figure 4, it is not that t-SNE is inefficient, rather UMAP is uniquely clever with this assumption.

Finally, although UMAP and t-SNE are mainly used as visualization tools, if we consider embedding D in a dimension $d > 3$, t-SNE is exponentially slower than UMAP, as shown in Figure 5. This is due to the KL Divergence Cost function that t-SNE uses. KL Divergence can be thought of as a measure of the information gained by revising one's beliefs from the original probability distribution to the updated probability distribution. In practice, updating a probability distribution involves using Bayes Theorem, given as

$$P(model|data) = P(model) \cdot \frac{P(data|model)}{P(data)}.[16]$$

To calculate the normalization constant $P(data)$ requires using binary space partitioning trees, which scale exponentially as the embedding dimension gets larger. Such trees are the cause of t-SNE's inability to reduce the data to higher dimensions. This is why McInnes et al. claim that UMAP may be a good dimension reduction tool, which could be helpful in a machine learning pipeline.[12]

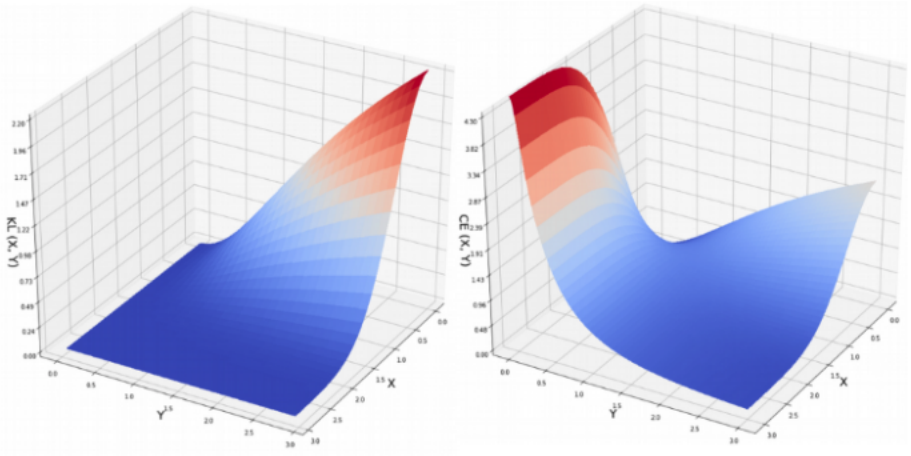


Figure 6: KL-Divergence KL on the right, and cross entropy CE on the left. X represents the distance between points in a higher dimension, and Y represents the distance between the corresponding points in a lower dimension. Notice that when X is large and Y is small, CE has a much higher penalty than KL. [14]

4.2 Does UMAP Actually Preserve Global Structure Better than t-SNE?

The authors of UMAP mention that their algorithm "preserves more of the global structure [than t-SNE does]." About a year ago (Dec. 19), researchers Kobak and Linderman published the preprint titled "UMAP does not preserve global structure any better than t-SNE when using the same initialization."[10] They claim that the reason why some say UMAP preserves the global structure better is because of the way it initializes points in the low dimensional space, which is not original to UMAP. The default initialization for UMAP uses a technique called Laplacian eigenmaps, whereas the default initialization for t-SNE is random about a Student-t distribution. Kobak and Linderman show that when points are randomly initialized in the low dimensional space with UMAP and t-SNE, each projection poorly represents the global structure of the data. Moreover, when low dimensional points are initialized in UMAP with Laplacian eigenmaps and in t-SNE with a PCA based method, both algorithms produce similar and improved results. Although the most recent version of the UMAP paper acknowledges Koback and Linderman, the UMAP authors still argue that different cost functions used plays an important role. Consider the plots shown in Figure 6, which demonstrates that the t-SNE cost function does not provide a large penalty when large distances are embedded as smaller distances. As opposed to UMAP, in t-SNE there is little incentive for the model to correct for large distances that are represented as small in $\mathbb{R}^d$, and so global distances are not preserved to the same degree as UMAP.

5 Experimentation

5.1 Python Implementation

The California Housing Prices data set[2] was used to implement the model in Python 3.8.[15] This data contains 20,640 entries and has ten attributes.[16] Therefore, we are assured that our data set, while not being unmanageable with a laptop processor, will sufficiently test each algorithm to expose their differences.

[15]The UMAP implementation used is from the UMAP Python 3 library [11]. The t-SNE implementation used is from the sklearn.manifold library.[1]

[16]longitude, latitude, housing median age, total rooms, total bedrooms, population, households, median income, median house value, ocean proximity

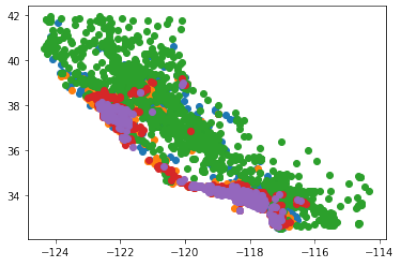
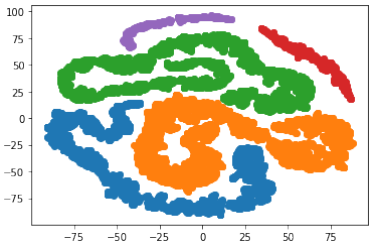
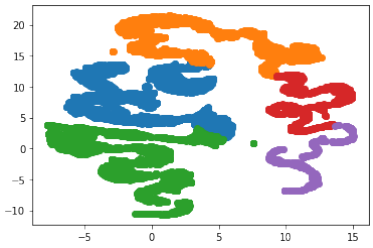


Figure 7



(a) UMAP ($nearestneighbors = 10$)



(b) t-SNE ($perplexity = 8$)

Figure 8: Output of the two algorithms with a comparably low measurement of the number of nearest neighbors. [12][15] With a free Google Colab account, UMAP took 24 seconds to run, whereas t-SNE took 36 seconds.

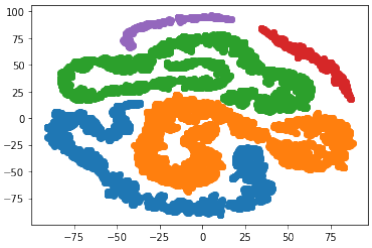
First, we use the sklearn implementation of k-Means to cluster the data into 5 clusters – this number of clusters was chosen from the elbow method.[8] Next, we ran both UMAP and t-SNE, varying the nearest neighbors and minimum distance parameters, as well as the perplexity parameter for t-SNE.

UMAP's visulization displays a lack of clustering in the data set. Although the specific shapes are not interpretable, from Figures 8 and 9, we conclude that this data set represents a range of data points that begin to morph into each other. We see from Figure 7 that data points in the cities of LA and San Francisco (purple, red, orange, and blue) are all adjacent to each other, but also that housing data corresponding to the blue properties are relatively different from properties represented by purple points. t-SNE, with perplexity values that are relatively similar to the corresponding nearest neighbor plots[6], in some respects better captures the clustering of the data set by isolated the purple data points. Still, in many places t-SNE also reflects the lack of clustering in the data set. In this respect, t-SNE is worse than UMAP because it fails to completely unravel the data, as shown in Figures 8 and 9. In this respect, because the t-SNE visualization is more "jumbled together" and less separated than the UMAP visualization, we conclude that this implementation of UMAP better preserves the data's global structure.

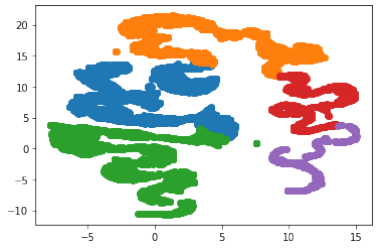
5.2 Implications of Local Connectivity in UMAP

In the UMAP algorithm, the assumption that G_H is locally connected (every vertex has an edge leaving it), leads to a faster runtime with respect to data set dimensions. In practice this ensures that data points form clusters, since we are essentially requiring that every point has a nearby point; no point can be completely isolated from the others.

To better understand the implications of this assumption, consider the least-possible-clustered data set, a lattice. For visualization purposes, we will consider using UMAP to reduce a 2-dimensional lattice to a 1-dimensional line, as shown in Figures 10, 11, and 12. Ideally, UMAP would capture the lattice structure



(a) UMAP ($nearestneighbors = 300$)



(b) t-SNE ($perplexity = 50$)

Figure 9: Output of the two algorithms with a high measurement of the number of nearest neighbors (at the top of each parameter's respective range as specified by each paper). [12][15] With a free Google Colab account, UMAP took 1 minute and 13 seconds to run, whereas t-SNE took 3 minutes and 11 seconds.

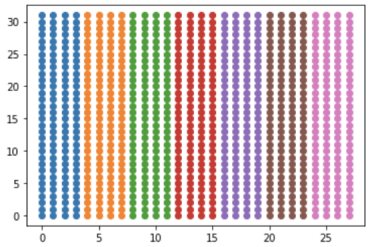


Figure 10: A lattice of 1024 points, columns of width 8 colored differently to visualize where the points land.

with points that are mostly evenly spaced on a line. Instead, the algorithm tends to cluster the data into groupings, which persists with different values of the nearest neighbor parameter. This makes sense, since although the nearest neighbors parameter does account for how much of the global structure of the data should be preserved, the problem here lies in capturing the local structure of the data. Also, changing the minimum distance parameter, only affects the scale of the projection, and does not change the clustering in our projection.

In the real world, we don't expect our data to resemble the lattice structure; we expect some clustering in data sets, since data analysis is focused on finding relationships within a data set. Still, examining the projection of a lattice structure reveals that UMAP struggles to capture isolation in a data set, meaning that UMAP is not a good tool to project data with many outliers or anomalies.

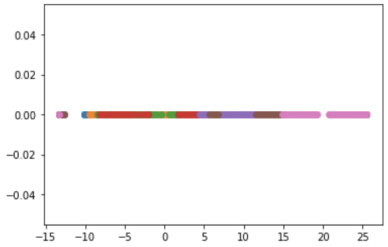


Figure 11: The projection with nearest neighbors parameter equal to 15 and the minimum distance parameter equal to $\frac{1}{2}$. Notice similar colored points are roughly placed together.

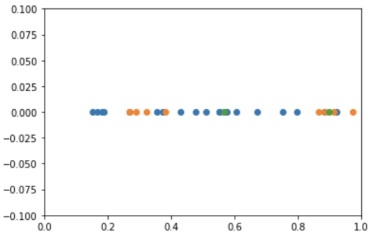


Figure 12: A zooming in of Figure 11 to display the clustering of small groups of points, which is due to the local connectivity assumption.

6 Conclusion

Perhaps the biggest take away from this analysis is the benefits of a theoretically strong mathematical foundation behind an algorithm. In many cases, the strengths of UMAP were motivated from theoretical constraints. From an analytical perspective, the deep mathematical underpinnings of the algorithm motivate its construction and provide a more thorough approach to visualizing high dimensional data.

6.1 Other Dimension Reduction Methods

As with most nonlinear dimension reduction techniques, both t-SNE and UMAP do not produce directly interpretable results.[17] Principal Component Analysis (PCA) is a linear reduction technique, which produces an interpretable visualization where the dimensions are the directions of greatest variance in the data. The drawback is that this method assumes the data can be linearly separated. A nonlinear technique that uses similar techniques to t-SNE and UMAP is Largevis. All three of these algorithms prioritize local structure over global structure, in order to best preserve clustering. As the authors of UMAP note, multidimensional scaling (MDS) seeks to better preserve the global scale of the data, and the PHATE algorithm is a hybrid approach that attempts to account for local structure while still employing MDS for global preservation. [12]

6.2 Future Exploration

With more time, there are many questions that could be explored. I think it would be interesting to further examine the inner workings of both UMAP and t-SNE. For example, it would be interesting to analyze how UMAP operates without the assumption that G_H is locally connected – perhaps the algorithm could work better on more disperse data sets without this assumption. In t-SNE, what type of visualizations would we archive if we let the variance σ be constant for all values of i; would we get a more globally accurate projection? Furthermore, I am interested in the idea of stripping these algorithms down to their bare essentials, even at the expense of losing some of the quality of our visualization. Could there be a simpler way of designing an algorithm similar to UMAP, that (even if its visualization was much lower quality) uses more basic mathematics? These questions could be analyzed by writing our own implementation of UMAP or t-SNE. For a beginner coder such as myself, this is a daunting task. Thankfully there are already Python constructions of UMAP and t-SNE from scratch.[7][5]

[17] shapes, Euclidean distances between points, etc

References

sklearn.manifold.tsne. https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html.

California housing prices. https://jmyao17.github.io/Kaggle/California_Housing_Prices.html, 1990. Accessed: 2020-12-15.

t-sne, clearly explained. https://www.youtube.com/watch?v=NEaUSP4YerM, 2017. Accessed: 2020-12-15.

Pca, t-sne, and umap: Modern approaches to dimension reduction. https://www.youtube.com/watch?v=YPJQydzTLwQ, 2018. Accessed: 2020-12-15.

Gabriel Beauplet. t-sne in python from scratch. https://github.com/beaupletga/t-SNE, 2018.

Andy Coenen and Adam Pearce. Understanding umap. Technical report, Google Pair, 2020.

Jack Downson. How to program umap from scratch. https://morioh.com/p/884822e1fbd6, 2019. Accessed: 2020-12-15.

Alind Gupta. Elbow method for optimal value of k in kmeans. https://www.geeksforgeeks.org/elbow-method-for-optimal-value-of-k-in-kmeans/, 2019. Accessed: 2020-12-15.

Geoffrey E Hinton and Sam Roweis. Stochastic neighbor embedding. In S. Becker, S. Thrun, and K. Obermayer, editors, *Advances in Neural Information Processing Systems*, volume 15, pages 857–864. MIT Press, 2003.

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

Leland McInnes. *Understanding UMAP*, 2018.

Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction. *arXiv*, 2020.

Nikolay Oskolkov. How exactly umap works. https://towardsdatascience.com/how-exactly-umap-works-13e3040e1668, 2019. Accessed: 2020-12-15.

Nikolay Oskolkov. Why umap is superior over t-sne. https://towardsdatascience.com/why-umap-is-superior-over-tsne-faa039c28e99, 2019. Accessed: 2020-12-15.

Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-sne, 2008.

Wikipedia. Bayes' theorem — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Bayes'%20theorem&oldid=994418363, 2020. [Online; accessed 20-December-2020].

Wikipedia. Entropy (information theory) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Entropy%20(information%20theory)&oldid=992799080, 2020. [Online; accessed 20-December-2020].

Wikipedia. Nerve of a covering — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Nerve%20of%20a%20covering&oldid=984128522, 2020. [Online; accessed 20-December-2020].