An Overview and Comparison of t-SNE and UMAP

Mustafa Aydogdu

12/20/2020

Contents

1 Introduction

In order to understand the properties of a dataset, we need to have a good visual representation of it. Just looking at the numbers in a data is not enough to understand the big picture. However, as for high dimensional datasets (4 or more dimensional), we are not able to visualise them directly.

This means that we will need to reduce a high dimensional dataset into lower dimensions in order to represent it geometrically. However, any attempt to reduce the dimensionality will result in loss of some information from the data in one way or another since it will manipulate the dimensions in some way and squeeze them into lower dimensions. But if we know what we want to learn about a dataset from its geometric representation, we can more easily produce a reduction algorithm suitable for our need.

In many fields that deal with high dimensional datasets, such as machine learning, neuroinformatics and bioinformatics, one of the most important features of a dataset is how the data clusters. So, a good reduction algorithm's job that wants to represent clusters is to place the similar data points close to each other and the dissimilar ones far away from each other in low dimensions.

In this paper, I am going to focus on two dimension reduction algorithms, t-SNE and UMAP. Even though t-SNE is an older technique than UMAP, I am going to discuss UMAP first because it is a more detailed algorithm that, in my view, encompasses t-SNE. So, explaining UMAP and then pointing to the steps that t-SNE lacks will make it easier for the reader to understand the differences between the two. While discussing the two methods, I will give an outline of their steps and provide some of the underlying mathematical proofs wherever necessary. Then, I will talk about the similarities and differences between the two. I will specifically mention that they are fundamentally similar algorithms even though UMAP might have advantages in terms of computational efficiency and preserving the global structure of a dataset.

Then, I am going to talk about the application of the two algorithms on a real life dataset about costs of living in various cities around the world. In this section, I will also give brief descriptions of the parameters used by the two algorithms.

2 An Overview of UMAP

UMAP algorithm consists of two main steps: (1) analysing the source data and creating a topological description of it, and (2) forming its low dimensional representation based on that topological description and optimising it. I will first try to explain how UMAP handles the source data and creates its topological description.

Real world datasets are often sparse and not well distributed, which can make it really hard to analyse the data. For example, if we want to understand which points are close to each other or, in other words, similar with each other, we can simply create a ball of fixed radius around each point, and say that the points that remain inside the ball are similar to the central point. Then, we would represent them as being in the same cluster in the low dimensional representation. But there is a problem with creating a ball of fixed radius: Some clusters in the high dimensional data can be dense whereas some can be sparse. If we make the radius too small, some points that actually should cluster together in the low dimensional representation may end up in different clusters because they were sparsely distributed in the source data. If we make the radius too large, there is the risk of bringing together some points that should not actually cluster [3] (see figure 1). So, finding the right radius length is impossible.

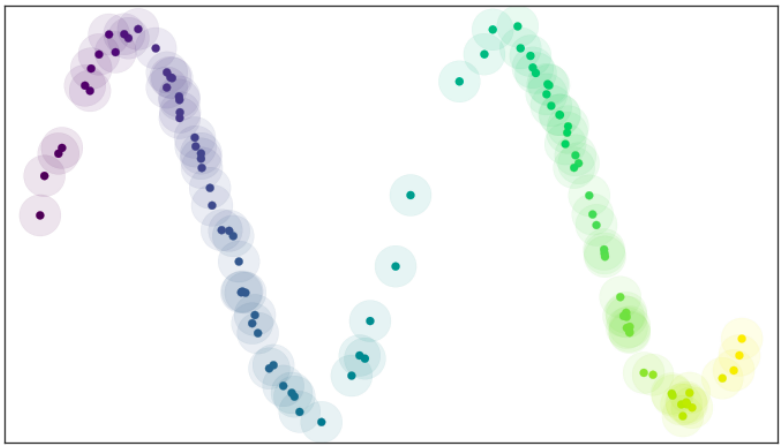

Figure 1: An example dataset with a ball of fixed radius around each point [3]

We can think of a way to overcome this problem if we can understand the nature of real world datasets. Suppose that we have a fashion dataset, which contains images of shirts, shorts, shoes etc. The shirts might exist in a dense group in our dataset whereas the shoes might be more sparsely distributed but

still together. This depends on how shirts and shoes' attributes were labeled in the process of collecting the data but since it is natural to have used a different metric (rating system) for different groups, the data may end up being non-evenly distributed. So, this issue is due to applying different metrics when creating the data and therefore, is not a characteristic property of the dataset which needs to be preserved and represented. This allows UMAP to ignore the differences in the metrics and to make the assumption that the data points are uniformly distributed on a manifold.

However, the problem is that the source data is not provided in a uniformly distributed manner, and needs to be turned into a uniformly distributed form by some method. The way UMAP does this is by using the Riemannian metric that can vary from point to point in the space. So, let's think about what we want the Riemannian metric exactly to do. The distances between points in dense groups should increase quickly whereas the distances between points in sparse groups should increase slowly. But how exactly will it do that, and what should it consider about real world datasets?

It needs to take into account that some points can be completely isolated, and therefore may not be part of any cluster. However, UMAP assumes that the data is locally connected: every point should be similar to at least some other point. It is said in the original UMAP paper that "it is safe to assume that the manifold M is locally connected[4]." I think that the reason why it is safe is that it is not realistic to have disconnected points in real world datasets in terms of the way things work: every object is at least a little bit similar to some other object if the data is large enough and not completely random. So, this is why the distance to the nearest neighbour is taken as 0 and only then, the distance starts increasing. In order to account for different levels of sparsity or density, the distance will keep increasing with each intersection with another point [3]. This is where the idea of fuzziness comes into play: the ball around a point becomes fuzzier and fuzzier as it meets with more neighbouring points (see figure 2). (It is important to note that the ball stops growing after it reaches a certain number of neighbours, and it is controlled by the parameter, n, the details of which I will discuss later in this paper.)

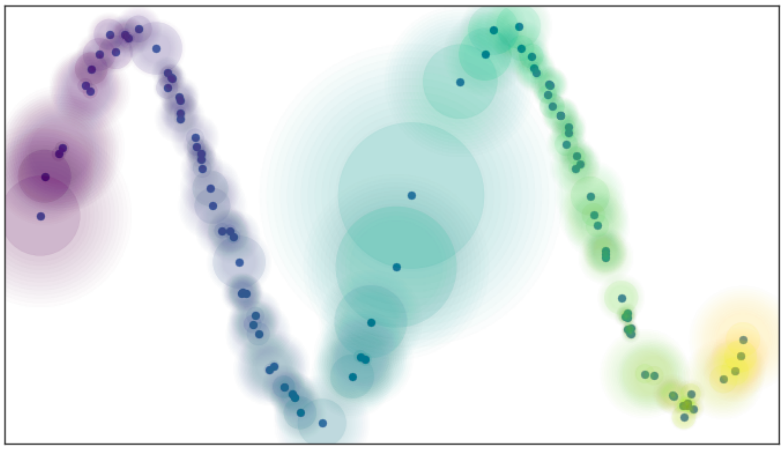

Figure 2: An example dataset with fuzzy balls that locally vary. Fuzziness starts only after the intersection with the nearest neighbour to ensure local connectivity. [3]

In order to create a topological representation of the source data, UMAP turns these distances into something called "simplicial set". A simplicial set is a set of n-simplices for each n where n is a non-negative integer. For example, 0-simplices are a set of vertices; 1-simplices are a set of edges between points; 2-simplices are a set of triangles; 3-simplices are a set of tetrahedrons and so on (see figure 3). If some points are found to be similar to each other, they will be part of the same simplice. However, since a distances is not a binary value, the membership of a point to a simplice cannot be a binary value where it is a member or not, but rather it should be a number in some range (or fuzzy) [4]. This is why these sets are called the fuzzy simplicial sets.

Figure 3: An example of simplices that goes up to 3-simplices at maximum.

However, while forming the edges between points, the use of a locally varying metric will create a problem: the distance between a pair of points may not be

the same from the perspective (metric) of each point since each point has its own custom metric. For this, UMAP algorithm thinks of each pairwise distance as a probability that an edge between them will be constructed. So, each edge will have a weight equal to the probability. Then, if a is the weight of an edge (the probability of that edge forming), using one of the points' metric, and b is the weight of the other edge between them, then the combined weight is the probability that at least one of them is formed, and it is given by the following equation [2]:

$$a + b - a * b \tag{1}$$

Proof:
Let k be the probability that neither of the edges is formed.
Then, $k = (1 - a) * (1 - b) = 1 + a * b - (a + b)$
Let c be the probability that at least one of them is formed.
Then, $c = 1 - k = 1 - (1 + a * b - (a + b)) = a + b - a * b$

So far, I explained how UMAP derives the fuzzy simplical sets from from the distances maintained by using a locally varying metric. Now, we can look at how UMAP finds a suitable low dimensional representation.

The low dimensional representation will be in Euclidean space and will use the squared Euclidean distance, not a varying metric. We said that UMAP regards the distances in the source data as probabilities and expresses them as weights of edges (or 1-simplices). The most straightforward way of making a low dimensional representation that comes to mind is to scale down all the distances by some number and locate the points in the Euclidean space with those scaled distances directly. However, it is not that easy, and in fact it wouldn't work. Assume that we located some of the points in the graph such that the distances between are correctly represented. Now, if we add some other point, this point's distance to all other pre-existing points should be also accurate. But it could mean changing all other points' locations as well. So, we actually need a better way to do it, and inevitably it will be more complex. The way UMAP does it is similar to the graph layout algorithms that work based on attraction and repulsion. The equation involved is called Cross Entropy. Let the set of all 1-simplices be E, $w_h(e)$ the high dimensional weight of an edge and the $w_l(e)$ the low dimensional weight of that edge. Then, the Cross Entropy is given by:

$$\sum_{e \in E} w_h(e) \log \left(\frac{w_h(e)}{w_l(e)} \right) + (1 - w_h(e)) \log \left(\frac{1 - w_h(e)}{1 - w_l(e)} \right) \tag{2}$$

Explanation:

The first term acts as an attractive force, and the second term acts as a repulsive force. Notice that the expressions in the term, $w_h(e) \log\left(\frac{w_h(e)}{w_l(e)}\right)$*, are probabilities of edges occurring whereas the expressions in the second term are probabilities of edges not occurring since they have a* $(1 -)$ *in front of them. Also, if the value of the overall expression is negative, it means repulsion; if positive, it means attraction between the two points.*

So, for example, if $w_h(e)$ *is high and* $w_l(e)$ *is low, then the second term will be so small that we don't need to consider it. The value of the first term will be positive, causing an attraction and consequently lowering the* $w_l(e)$*, which is proportional to the low dimensional distance .*

The other case would be that $w_h(e)$ *is low, and* $w_l(e)$ *is high, the first term will be quite small (so, we don't need to consider it), and the second term will be negative, causing a repulsion. So, over many iterations, these small amounts of attraction and repulsion will produce a graph that correctly represents the high dimensional data [2].*

Now, we are done with explaining the mathematical theories behind UMAP and its general algorithm. But we haven't discussed the computational sides of things. For example, when we were discussing the way the distances will be calculated and the idea of local connectivity (that is every point should be connected to at least some other point), we didn't tell which equations exactly the computer will use. In this paper, I will not explain all of the computational processes but I will briefly refer to them in the section where I will discuss the differences between UMAP and t-SNE. So, now I will move on to the discussion of t-SNE.

3 An Overview of t-SNE

In this section, I will give a very brief description of the t-SNE algorithm. The detailed description of the differences between the two techniques will be left to the next section.

t-SNE looks at the nearby points for each point P in the dataset, and creates a Gaussian distribution to reflect the probabilities of edges between P and the nearby points. The probabilities are derived from the Euclidean distances (unlike UMAP's locally varying metric). A greater distance would mean a smaller probability, and vice versa [7].

However, as explained in the UMAP section, UMAP resorts to a locally varying metric in order to resolve the problem of varying densities of data points.

t-SNE also needs to take care of this problem and it does so by adding the σ variable to the probability equation, which is computed for each point separately to account for the varying densities [7].

t-SNE does not create a topological description using the simplices as opposed to UMAP. It skips that part completely and creates a t-Distribution for the probabilities between points in the low dimension. Then, it tries to make the values from the Gaussian distribution resemble the values from the t-Distribution by using KL-Divergence equation. The KL-Divergence equation is another cost function similar to the Cross Entropy which is used by UMAP. It also works as an attraction-repulsion algorithm but it causes loss of the information between distant points, and I will discuss its details in the next section (see section 4).

The reason why t-SNE uses a t-Distribution for the low dimensional graph is that it has a heavier tail than the Gaussian. So, the distant points in the Gaussian distribution will map to even more distant points in the t-distribution, and the close points will map to even closer points in the low dimension [5] (see figure 4).

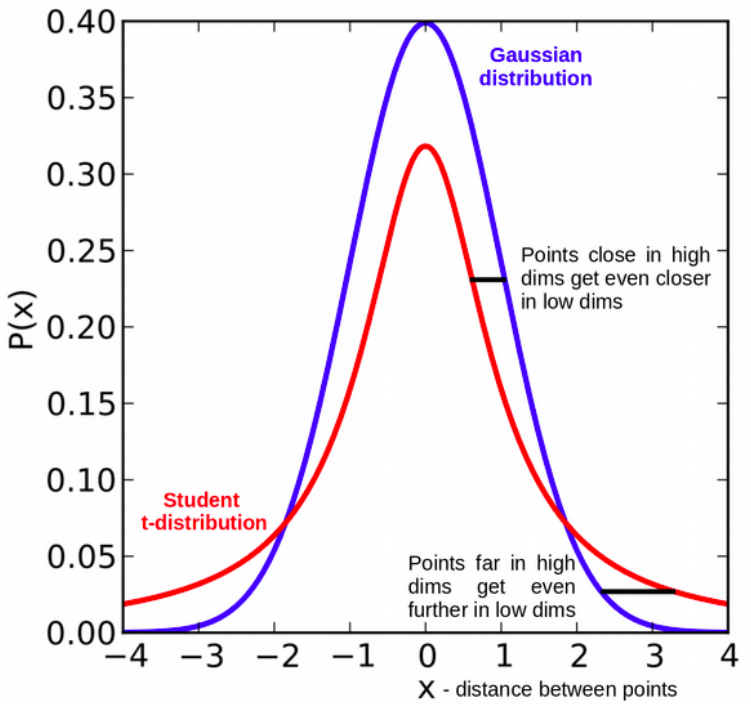

Figure 4: Side by side comparison of Gaussian and Student t-distribution curves

So, this is actually necessary to overcome the crowding problem that is common in dimension reduction. The crowding problem can be described as follows: Suppose you have a point in 2-D, and you have 12 twelve points that are equidistant from this point. When you project this dataset to 1-D, you have only two places to put these points, which will cause the points to overlap. So, the t-Distribution actually gives more available space to the low dimensional graph to place the objects into [6]. UMAP resolves this problem nicely by using the *min_dist* parameter and I will discuss that in section 5 (see section 5).

4 Similarities and Differences Between t-SNE and UMAP

Nikolay Oskolkov says that his first impression when he heard about UMAP was that it was "a completely novel and interesting dimension reduction technique which is based on solid mathematical principles and hence very different from tSNE which is a pure Machine Learning semi-empirical algorithm." He doesn't explicitly say whether or not his first impression came out to be accurate in his paper, but at the end of the paper, he says "UMAP overall follows the philosophy of tSNE, but introduces a number of improvements such as another cost function and the absence of normalization of high- and low-dimensional probabilities. [5]" This final statement suggests that he thinks that his first impression was not so true because UMAP follows the logic of t-SNE at the end of the day even though it has some improvements. Now, in this section, firstly, I will look at why they are similar overall and then I will look at some improvements of UMAP.

They are similar because the overarching methods that they follow are very similar: They look at a certain number of neighbouring points around each point to generate the distances first and then the probabilities. UMAP uses a locally varying metric, but even though t-SNE uses the Euclidean distance, it does a similar thing when it uses the σ variable that varies from point to point to account for the varying densities. UMAP has an extra step of creating a topological description with fuzzy simplical sets but this is not adding much to the overall algorithm because it is not adding any new information beyond the information provided by the distances. Actually, any paper that I read doesn't mention the topological representation as a process that makes UMAP significantly better. Even the original UMAP paper doesn't mention it where it compares UMAP with t-SNE. However, the importance of this step is due to the fact that it provides the underlying mathematical theory for UMAP's ability to capture the topological description of the source data. So, the entire algorithm becomes mathematically "solid" [4].

The main improvements of UMAP can be summarized as a better preservation of the global structure and a better computational speed [5]. UMAP is able to preserve the global structure better because it uses a cost function that can more accurately map large high dimensional distances to large low dimensional distances; and that can map small distances to small ones in the low dimension as well. This was discussed in detail in section 2 (see section 2). In contrast, tSNE uses KL-divergence, a cost function that does map small distances in the high dimension to small distances in the low dimension but it cannot do that for large distances.

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

Figure 5: *KL-Divergence: It can be seen that if p_{ij} is high, the penalty (the value of C) will be big, making q_{ij} higher. If p_{ij} is low, the log term doesn't have much effect, so there will be no significant penalty.*

So it is not able to capture the global structure as well as UMAP because capturing the global properties requires a good representation of relationships between distant points as well. Laurens van der Maaten, one of the developers of t-SNE, talks about the KL-Divergence cost function in a presentation that he gives about t-SNE and says, "So, what this Kullback-Leibler divergence does is, it really tries model large p_{ij}'s, so similar high dimensional points by large q_{ij}'s, so points that are close together in the two dimensional map. It doesn't work the other way around. If you have two points that are very dissimlar, sort of small p_{ij}, then you don't really care about what the corresponding q_{ij} is [6]." This means that the edges with high probabilities in the high dimension (short distances) map to high probabilities in the low dimension (short distances). But it doesn't work for the large distances. Large high dimensional distances can map to anything; there is no penalty (or cost) for that in the equation.

When it comes to the runtime of algorithms, there is no doubt that UMAP is much faster than t-SNE. For example, on the MNIST dataset, UMAP takes 87 seconds whereas t-SNE takes 1450 seconds [4]. The reason why t-SNE is slower is its use of the σ variable that needs to be calculated for each point In its calculation, there is a logarithmic term and it requires a lot of computational work [5]. Also, t-SNE normalizes high dimensional data, which is also a computationally expensive process [5].

5 Example Dataset Analysis

In this section, I will analyze the UMAP and t-SNE representations of a dataset on costs of living in different cities around the world. The dataset contains 160 cities (or data points) and 55 attributes (or dimensions) for each city [1]. There are cities from all six continents on which human populations live. The attributes include the prices of various commodities and services such as a pair of jeans, fitness club monthly fee for an adult and one kilogram of apples. I will explain the effects of varying the parameters of both algorithms on the outputs, and I will briefly explain the specific role of each parameter along the way. Then, I will do a comparison of both algorithms at the end.

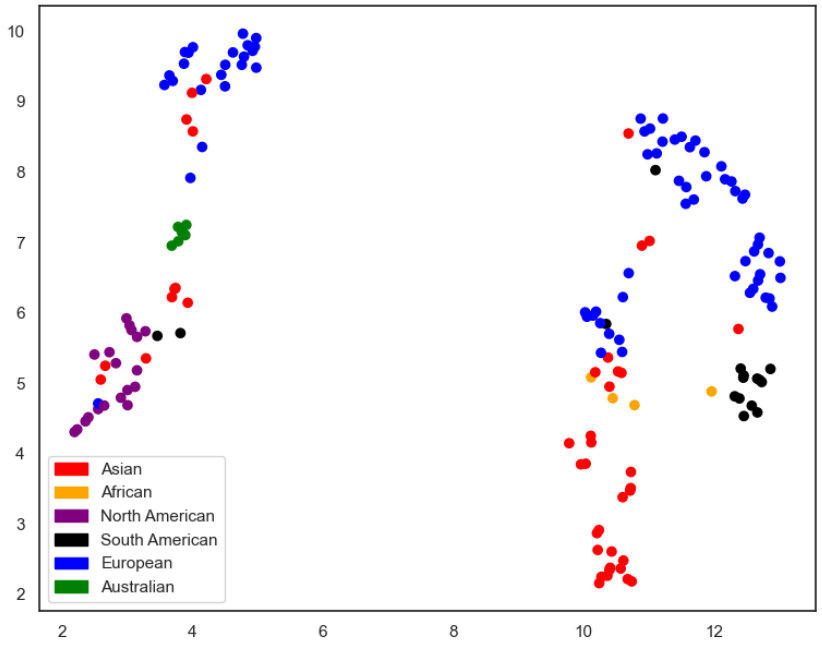

Figure 6: UMAP, $min\_dist = 0.1$, $n\_neighbours = 15$, 2-D, color-coded by continent

I firstly ran UMAP on the dataset with the parameters $min\_dist = 0.1$, $n\_neighbours = 15$ in 2-D, and I color-coded the cities by their continents (see figure 6). The resulting graph seems to be able to represent the costs of liviving in different cities pretty well. Most of the cities cluster together according to the

their continents, which makes sense since the costs really change from continent to continent. It seems like there are two clusters of Asian cities, and some of them are placed together with North American and European cities. But since they are cities like Hong Kong and Dubai, which are some of the richest Asian cities, it makes sense to see them there. The distances between clusters seem to be meaningful as well. From left to right in a semi-circular shape, the cities are placed roughly in the following order: North American, Australian, European, African, South American, Asian. So, from left to right, the costs of living in cities decrease. For example, the North American and European cities are close, and the African and South American cities are close. Also, there are two separate clusters of European cities. But this also is meaningful: one of the clusters contains more expensive cities such as Rome, Zurich, Paris and London and the other contains cities like Tallinn, Warsaw and Zagreb. So, overall, UMAP produces a good 2-D representation of this dataset.

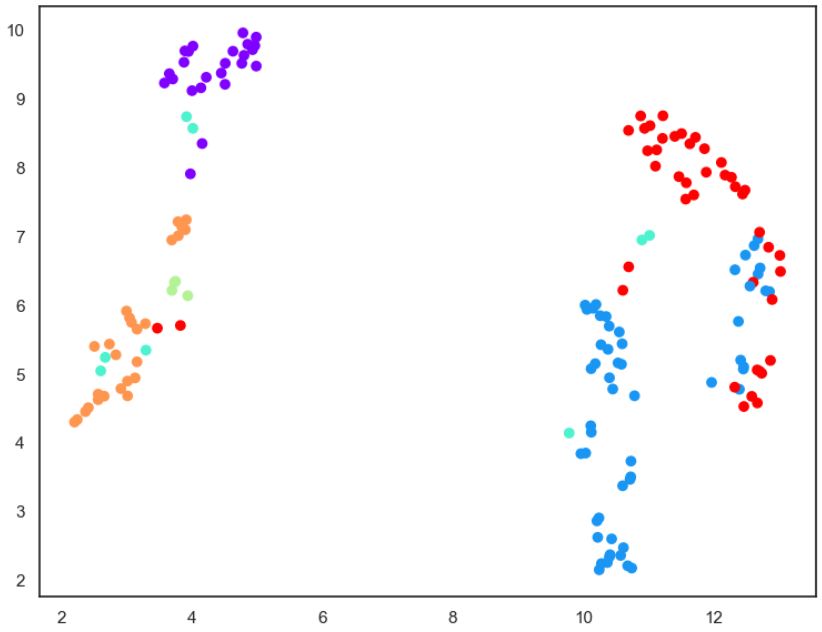

Figure 7: UMAP, $min\_dist = 0.1$, $n\_neighbours = 15$, 2-D, color-coded by k-means

Then, I applied k-means clustering to the source data and then, I plotted the same 2D UMAP graph from above, using the labels from the k-means as the color

code (see figure 7). The resulting graph is also successful. The k-means clustered together most of the cities from the same (looking at he colors and comparing with the original graph shows this) but there are a good number of cities that clustered with cities from different continents. However, it does the task of clustering the cities according to how expensive they are: the clusters that UMAP produces and those that k-clustering finds match mostly.

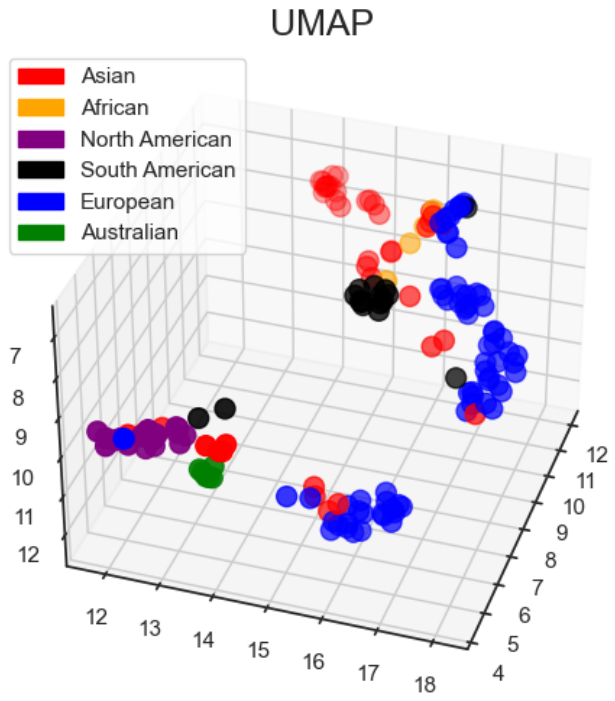

Figure 8: UMAP, $min\_dist = 0.1$, $n\_neighbours = 15$, 3-D, color-coded by continent

I ran UMAP again with the same parameter values but this time in 3-D (see figure 8). The resulting graph is color-coded by continent and it reveals approximately the same information about clustering as the 2-D version. We can observe the same types of clusters.

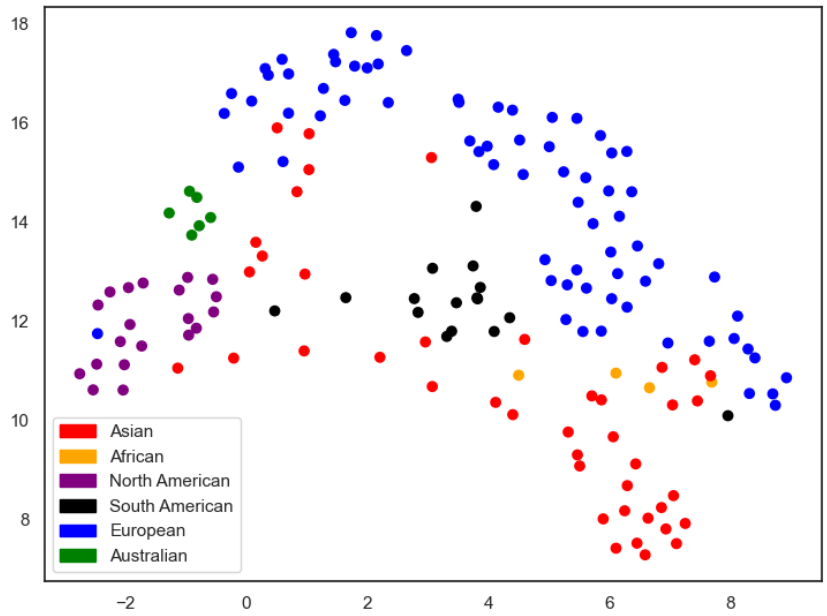

Figure 9: UMAP, $min\_dist = 0.7$, $n\_neighbours = 15$, 2-D, color-coded by continent

Then, I varied the min_dist parameter, keeping $n\_neighbours = 15$. The min_dist parameter sets the minimum distance between points in the low dimension. So, a lower min_dist means a more tightly packed representation, which can be good to observe the clusters or the general topology of the data [2]. Also, in t-SNE section, we had discussed the crowding problem and t-SNE resolves it. UMAP can also overcome the crowding problem via sufficiently high values of min_dist. So, lowering the min_dist to values less than 0.1 doesn't effect the representation much. However, setting $min\_dist = 0.7$ produces the graph (see figure 9). We can see that the points are separated more from each other, but consequently, the clusters became less visible.

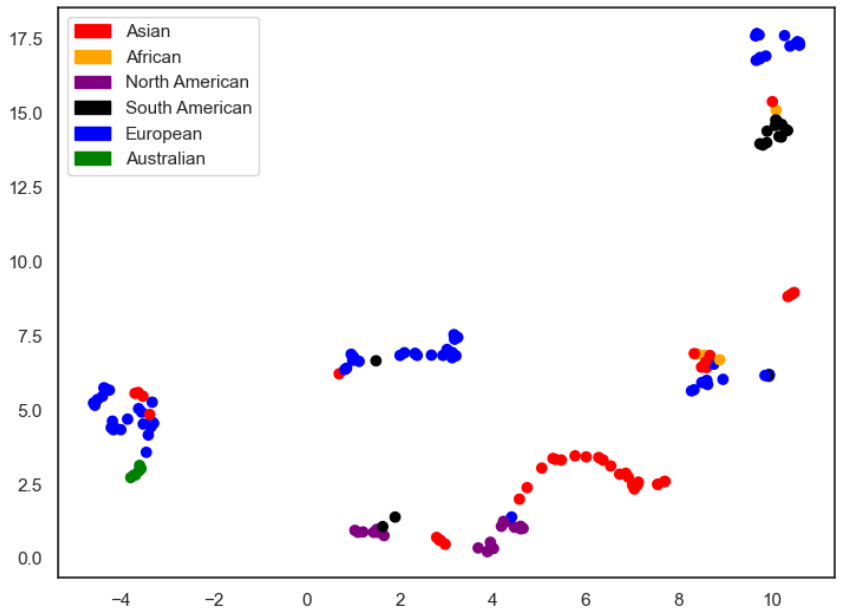

Figure 10: UMAP, $min\_dist = 0.1$, $n\_neighbours = 5$, 2-D, color-coded by continent

The last parameter that I checked is $n\_neighbours$. This parameter determines how many neighbours of each point UMAP will look when when constructing the high dimensional representation. Thus, it sets the balance between global and local structures: higher values of $n\_neighbours$ will focus more on the global structure, losing finer local details and lower values of $n\_neighbours$ will lose focus more on the local structure, putting together the closest points and leaving out the distant points [2]. For our case, it was equal to 15 for all of the above graphs. Setting it to $n\_neighbours = 5$ glues together very similar (or close) points together but the relationships between distant points get lost, causing the global structure to be less apparent (see figure 10). So, the meaningful distances between clusters mentioned above are no longer existent.

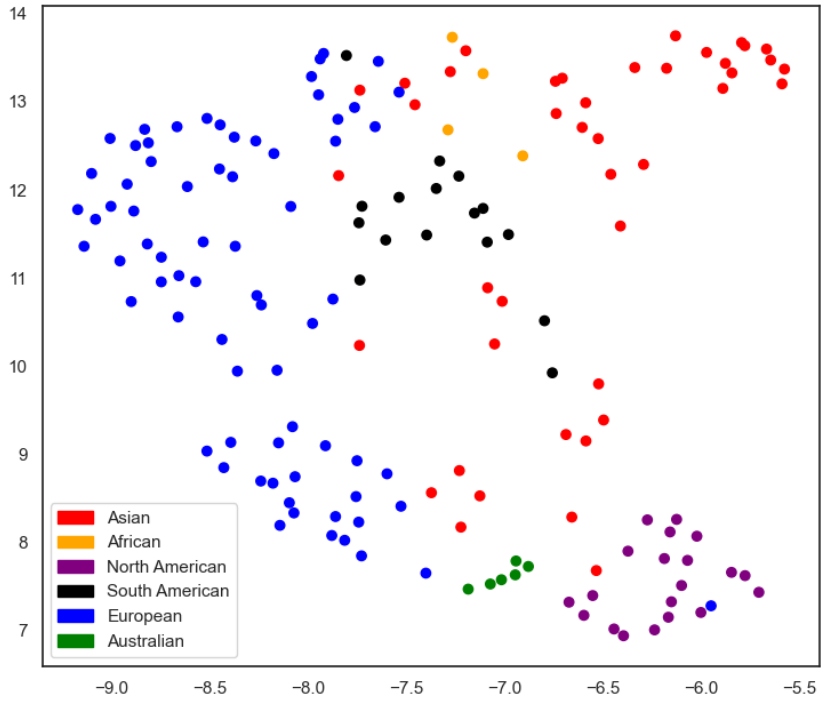

Figure 11: UMAP, $min\_dist = 0.1$, $n\_neighbours = 100$, 2-D, color-coded by continent

Setting it to $n\_neighbours = 100$ helps us see the very broad structure (see figure 11): we can see that there are clusters formed; similar clusters are closer and dissimilar ones are apart. However, the finer details such as that there should be two clusters of European cities are less visible. However, since a high value of $n\_neighbours$ looks at more data neighbouring points and thus take into account the distant points as well, the resulting graph might suggest that the clusters of cities should not be very separate from each other like in the figure.

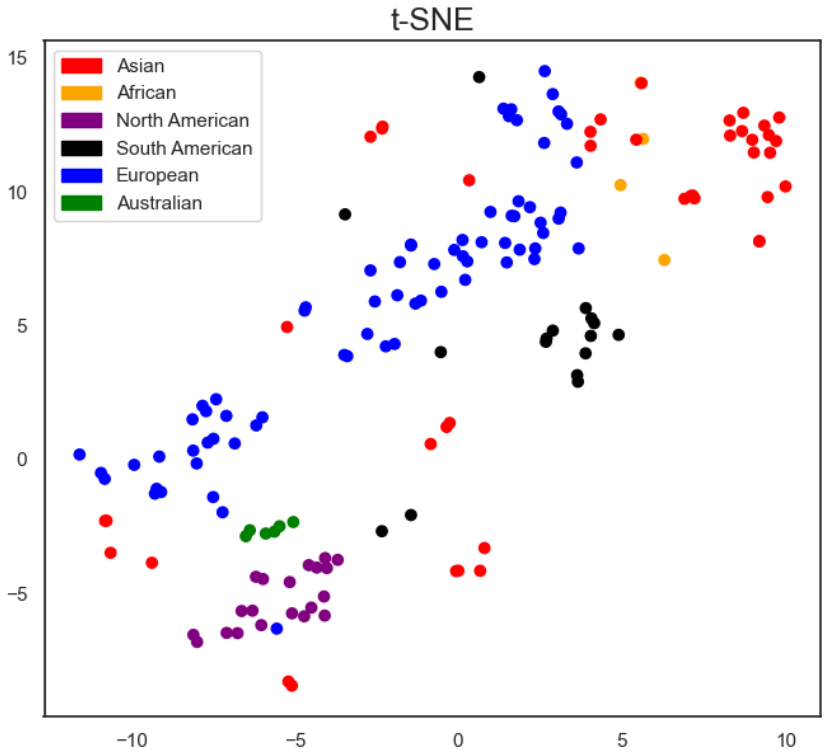

Figure 12: t-SNE, perplexity = 30, 2-D, color-coded by continent

Now, let us look at the visualisations produced by t-SNE. t-SNE has only one parameter that we can vary, which is perplexity. It functions very similar to the *n_neighbours* of UMAP. It determines how many neighbours of each point will be effective when building the representation [7]. Running t-SNE with $perplexity = 30$ results in a graph that is similar to the UMAP representation with $n\_neighbours = 100$: The clusters come very close to each other (see figure 12).

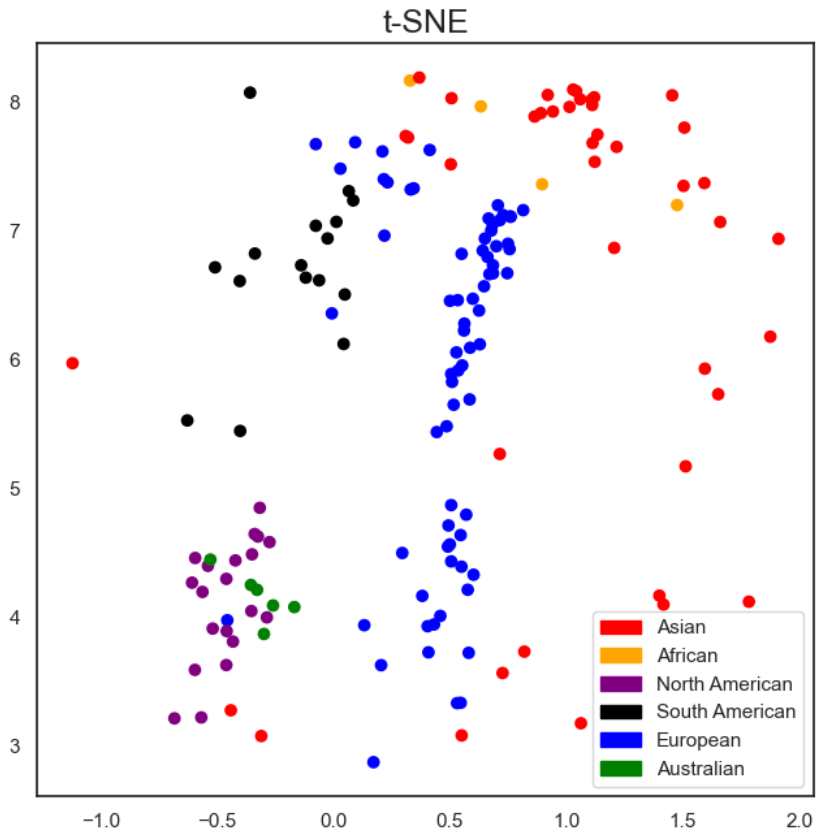

Figure 13: t-SNE, perplexity = 100, 2-D, color-coded by continent

Increasing *perplexity* to 100 will produce a graph, in which the fine details are very much lost: some clusters (for example, Australian and North American) mix up with each other (see figure 13).

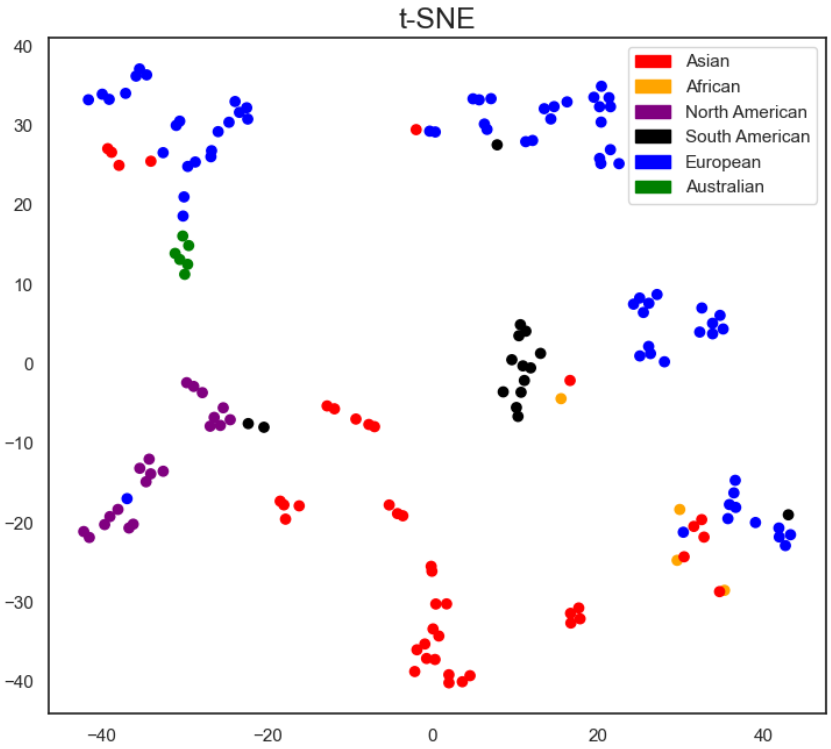

Figure 14: t-SNE, perplexity = 5, 2-D, color-coded by continent

Setting *perplexity* = 5, surprisingly results in a graph that is very similar to the graph of UMAP with *n_neighbours* = 15: there is more white space, and the clusters are more visible (see figure 14). However, it brings North American and Asian cities close to each other, which is not perfect.

Comparing the results maintained from UMAP and t-SNE can help us assess whether the two algorithms are actually similar or UMAP made a significant change in dimension reduction that leads to better representations. Of course, looking at only one dataset will not be enough but it can give us an idea when we also bring it together with the ideas from section 4 about why they might be similar. There are also other representations done by other people on which the two algorithms produce very similar results for a specific choice of parameters [3]. So, since when we run UMAP with *n_neighbours* = 15 (see figure 6) and t-SNE with *perplexity* = 5 (see figure 14), we get quite similar visualizations, we can conclude that they are similar algorithms, even though UMAP might have some

improvements. I was not able to test its possible improvement in respect to speed because my dataset is quite small (it has only 160 data points). However, in terms of the preservation of global structure, both are able to capture the global properties well for this dataset. However, UMAP is a little bit better because t-SNE brings together some Asian cities and North American cities, which doesn't seem to be accurate.

References

[1] Comparison of cost of living in various cities around the world. `https://www.kaggle.com/stephenofarrell/cost-of-living/metadata`. Accessed: 12/20/2020.

[2] How umap works. `https://umap-learn.readthedocs.io/en/latest/how_umap_works.html`. Accessed: 12/20/2020.

[3] Andy Coenen and Adam Pearce. Understanding umap. `https://pair-code.github.io/understanding-umap/`. Accessed: 12/20/2020.

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

[5] Nikolay Oskolkov. How exactly umap works. `https://towardsdatascience.com/how-exactly-umap-works-13e3040e1668`. Accessed: 12/20/2020.

[6] Laurens van der Maaten. `https://www.youtube.com/watch?v=RJVL80Gg3lA`, 2013.

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