Budapest University of Technology and Economics
Faculty of Electrical Engineering and Informatics
Department of Measurement and Information Systems

Using unsupervised clustering methods on the latent space of drug-like molecules

Bachelor's Thesis

Author
György Bence Józsa

Advisor
Péter Sárközy

December 13, 2021

Contents

HALLGATÓI NYILATKOZAT

Alulírott *Józsa György Bence*, szigorló hallgató kijelentem, hogy ezt a szakdolgozatot meg nem engedett segítség nélkül, saját magam készítettem, csak a megadott forrásokat (szakirodalom, eszközök stb.) használtam fel. Minden olyan részt, melyet szó szerint, vagy azonos értelemben, de átfogalmazva más forrásból átvettem, egyértelműen, a forrás megadásával megjelöltem.

Hozzájárulok, hogy a jelen munkám alapadatait (szerző(k), cím, angol és magyar nyelvű tartalmi kivonat, készítés éve, konzulens(ek) neve) a BME VIK nyilvánosan hozzáférhető elektronikus formában, a munka teljes szövegét pedig az egyetem belső hálózatán keresztül (vagy autentikált felhasználók számára) közzétegye. Kijelentem, hogy a benyújtott munka és annak elektronikus verziója megegyezik. Dékáni engedéllyel titkosított diplomatervek esetén a dolgozat szövege csak 3 év eltelte után válik hozzáférhetővé.

Budapest, 2021. december 13.

Józsa György Bence
hallgató

Kivonat

A modern gyógyszerkutatás egyik legfőbb motivációja az gyógyszerként viselkedő vegyületek előállítása. Az új gyógyszermolekulák felfedezése lehetővé teszi az eddig gyógyíthatatlannak vélt betegségek kezelését. A *de novo* molekulagenerálás egy olyan folyamat, mely során egy adott, molekulákat tartalmazó adathalmaz alapján generálunk molekulákat, amelyek az adatbázisban lévőekhez hasonlóak, de azoktól eltérőek. Ez a módszer az utóbbi évtizedekben egyre népszerűbbé vált.

Az új gyógyszerszerű molekulák előállítása nagyon költséges és időigényes. A folyamat felgyorsítására az elmúlt három évtizedben a gépi tanulást és a mély neurális hálózatokat alkalmazták. Különösen népszerű módszer a variációs autoenkóder használata a célzott keresésre alkalmas gyógyszerszerű molekulák látens terének létrehozására.

Egy ilyen látens tér azonban csak akkor használható, ha az a molekulák kémiai szerkezetére nézve sima. Ennek eldöntése azonban nem triviális, mivel a molekulák kémiai szerkezete nem könnyen számszerűsíthető, és az ilyen látens tereknek általában magas a dimenzionalitása, ami miatt szükséges dimenzió redukciós és vizualizáló algorithmusok használata.

Az elmúlt évtizedekben számos dimenzió redukciós és vizualizációs algoritmust fejlesztettek ki. A dolgozatomban öt gyakran használt algoritmust - PCA, t-SNE, UMAP, TriMAP és PaCMAP - vizsgálok meg, hogy mennyire használható eredményt adnak egy adott adathalmazra.

Mindegyik algoritmust abból a szempontból vizsgálom, hogy képesek-e egy 64 dimenziós látens teret úgy átalakítani, hogy az így kapott kétdimenziós tér a molekulák kémiai szerkezetére nézve sima legyen. Optimalizálom az egyes algoritmusok hiperparamétereit, ezzel vizsgálva, hogy hogyan alakítják át a kapott beágyazást, és elvégzek egy LERP-tesztet a teljes tér leképzésének vizsgálatára.

Abstract

In modern drug research, one of the most important tasks is finding novel drug-like molecules. The discovery of new drug molecules makes it possible to treat diseases previously thought incurable. *De novo* molecule design is the process of generating novel chemicals based on a dataset of drug-like molecules. This method has gained popularity in recent decades.

The cost of generating novel drug-like molecules is very costly and time-consuming. To speed the process up, machine learning and deep neural networks have been used in the last three decades. A particularly popular method is using a variational autoencoder to generate a latent space of drug-like molecules suitable for targeted searching.

Quantifying the quality of such a latent space is vital for effective usage. This task is not trivial however, as the chemical structure of molecules cannot be easily quantized and such latent spaces tend to be high-dimensional, leading to the need for dimension reducing visualization algorithms to be applied.

Many dimension reduction and visualization algorithms have been developed in recent decades. In this paper, I evaluate five commonly used algorithms – PCA, t-SNE, UMAP, TriMAP and PaCMAP – to see how well they perform on a given dataset.

I examine each algorithm on their ability to transform a 64-dimensional latent space such that the resulting two-dimensional space is smooth over chemical structure. I optimize the hyperparameters of each algorithm to see how they transform the resulting embedding, and perform a LERP test to see how they map the entire space into two dimensions.

Chapter 1

Introduction

1.1 Motivation

Computational bio- and chemoinformatics often requires the usage and processing of high-dimensional data. Working with such data poses certain challenges however. While some datasets are constructed with meaningful and easily interpretable feature sets, often times the dimensions carry no direct interpretation, forming a *latent space* of data points.

High dimensionality introduces redundancy in the representation of data points. It is also undesirable as the time of operations scale with the number of dimensions in the dataset.

Latent representation of data is advantageous for eliminating redundancy in the dataset. This is because features taken from real descriptors (such as number of carbon atoms in a molecule and molar mass) are often not independent of each other. These correlations of dimensions lead to spaces of lower dimension with near equivalent descriptive power to the original space.

High dimensionality also poses challenges in interpreting data. Correlations between features, or sets of features become difficult to discover as the number of potential pairings increase. Datasets with more than three dimensions pose additional challenges as visualizing such datasets is difficult and often impractical, rendering visual interpretation methods unusable.

1.2 Exploring the chemical space

One of the more important goals of modern pharmaceutical research is the discovery of novel drug-like molecules. This development however is a long and costly process. The entire procedure can in some cases take 20 years [20], and may cost as much as 2.6 billion dollars [3]. During the development, thousands of potential compounds are being tested, of which only half a dozen candidates will reach clinical trials, where their effects are tested in humans. In most cases, only one drug is approved so that it can reach doctors and patients.

In order to reduce the time cost of the procedure, as well as the financial cost, targeted search of the chemical space is necessary. This conclusion is also supported by the fact that while scientists have successfully synthesised millions of molecules with molar weight less than 500, the number of potential such molecules may be 10^{24}, or according to some

estimates, even 10^{60} [7], making random searches of the chemical space effectively impossible.

Using targeted searching on the space of drug-like molecules is a heavily researched topic. Algorithms exist, however, finding efficient searching algorithms without enumerating every element of the subspace is difficult. Deep neural networks have been used for *de novo* molecule generation with promising results [25]. One particularly promising method is the use of VAE's *variational autoencoders* [15] to transform the chemical space into a latent space that is smooth over chemical structure.

The number of dimensions of an autoencoder's latent space is usually between 32 and 128, which is rather large, and carries the challenges in the understanding of the underlying structure presented in section 1.1. The analysis of such latent spaces is important for finding novel molecules with desirable drug-like properties.

1.3 Unsupervised clustering

In data science, a very common task is discovering similarities and dissimilarities between data points. Categorizing points of data is perhaps the most obvious example of this, however, more general tasks exist in datasets where no such clean categories exist. These problems require a more general solution for representing similarities of points.

Cluster analysis is the grouping of objects such that objects in the same cluster are more similar to each other than they are to objects in another cluster. The classification into clusters is done using criteria such as smallest distances, density of data points, graphs, or various statistical distributions. Cluster analysis has wide applicability, including in unsupervised machine learning, data mining, statistics, graph analytics, image processing, and numerous physical and social science applications.

Clustering is used to identify groups of similar objects in datasets with two or more variable quantities. In practice, this data may be collected from marketing, biomedical, or geospatial databases, among datasets that come from preprocessing such databases.

Clustering is often used to discover underlying structure in data through *unsupervised learning*. No labels are given to the learning algorithm, leaving it on its own to find structure in its input. Unsupervised learning can be a goal in itself (discovering hidden patterns in data) or a means towards an end (feature learning).

1.4 Outline of work

In my thesis, I examine five dimension reduction algorithms, namely PCA, t-SNE, UMAP, TriMAP and PaCMAP. In chapter (2), I introduce each algorithm, and provide simple description of how they work. I will also present a basic comparison of based on these descriptions.

In chapter (3), I discuss the resources that I had during my work. This includes the hardware on which I ran the algorithms and the software packages that I used, including the implementations of all algorithms presented. I also describe the dataset that I used in my investigation.

Following that in chapter (4), I compare the algorithms in detail, running them with different parametrization on the latent dataset of drug-like molecules. With optimizing

parameters and running tests, I examine which unsupervised clustering algorithm works best on the given dataset.

Finally, in chapter (5), I summarize my findings and talk about future plans.

Chapter 2

Dimension reduction algorithms

For completeness, this chapter contains the theoretical background of PCA, t-SNE, UMAP, TriMAP and PaCMAP in common notation. After introducing every algorithm, their advantages and disadvantages are listed. At the end of the chapter, I compare these algorithms on a theoretical basis without my results working on the dataset detailed in 3.1.

$\mathbf{X} \in \mathbb{R}^{N \times M}$ is used to denote the original dataset, for which an embedding $\mathbf{Y} \in \mathbb{R}^{N \times K}$ is constructed. For each observation $i \in [N]$, $\mathbf{x}_i$ and $\mathbf{y}_i$ is used to denote its corresponding representation in $\mathbf{X}$ and $\mathbf{Y}$ respectively. $\mathbf{x}_i$ is a given value and $\mathbf{y}_i$ is a K-dimensional decision variable. The distance between $\mathbf{x}_i$ and $\mathbf{x}_j$, which is used as a measure of similarity between the points is denoted by $d(\mathbf{x}_i, \mathbf{x}_j)$. This distance metric may be defined differently between methods, but it is always a metric in $\mathbb{R}^K$.

2.1 PCA

Principal Component Analysis[10], or PCA, for short, is the most widespread dimension reduction algorithm. It works by finding *principal components*. These are a series of p unit vectors, where the i-th vector minimizes the average squared distance from the points to the line while also being orthogonal to the first $i-1$ vectors.

PCA is the process of computing the principal components of the data and performing a change of basis, usually using only the first few principal components and ignoring the rest. This leads to a lower dimensional representation of the data, where only the components that preserve as much of the data's variation as possible are included.

2.1.1 Mathematical details

The goal of the PCA algorithm is to find unit basis vectors $\mathbf{w}_{(i)}$, such that the projected data inherits the maximum possible variance of the original data matrix $\mathbf{X}$.

The first vector $\mathbf{w}_{(1)}$ has to satisfy

$$\mathbf{w}_{(1)} = \arg\max_{\|\mathbf{w}\|=1} \left\{ \sum_i \left(\mathbf{x}_{(i)} \cdot \mathbf{w}\right)^2 \right\} \tag{2.1}$$

The k-th component can be found by subtracting the first $k-1$ components from $\mathbf{X}$

$$\hat{\mathbf{X}}_k = \mathbf{X} - \sum_{s=1}^{k-1} \mathbf{X}\mathbf{w}_{(s)}\mathbf{w}_{(s)}^{\mathsf{T}} \tag{2.2}$$

and then computing the first principal component of the new matrix:

$$\mathbf{w}_{(k)} = \arg\max_{\|\mathbf{w}\|=1} \left\{ \|\hat{\mathbf{X}}_k \mathbf{w}\|^2 \right\} \tag{2.3}$$

The projected dataset is given by the following linear transformation:

$$\mathbf{T} = \mathbf{X}\mathbf{W} \tag{2.4}$$

2.1.2 Benefits of PCA

The popularity of PCA lies in its ease of use. It is easily accessible for data science use-cases, with a variety of implementations in a number of environments.

The application of PCA to a given dataset is trivial. The only parameter of the algorithm is the output dimension, and the result is a deterministic function of the dataset.[1] This means that using PCA does not involve fine-tuning hyper-parameters, which can take a long time and take the focus away from understanding the underlying data.

Furthermore, the PCA algorithm is simple, without any computationally expensive steps. This results in an algorithm that uses minimal hardware resources. Even working with large datasets, no supercomputer is needed for the application of principal component analysis.

2.1.3 Limitations of PCA

PCA has a number of limitations that make it unfavourable to us in certain use-cases. These limitations arise from certain assumptions made in its derivation.

One such assumption is the standardization of the used dataset. This means that the scaling of variables affects the result of PCA. Differently scaled features not only change the outcome of the transformation, but there is also a high likelihood of information loss. This can be counteracted by scaling the features of the dataset by its standard deviation. However, this is not always desirable.

Another assumption made is that the dataset contains only linear dependencies. PCA can capture linear correlations between the features but cannot capture nonlinear dependencies between them. In some cases, a transformation of the coordinates, such that the resulting representation only contains linear correlations between features, may enable PCA to successfully capture such dependencies. This is the basis for a generalization of the technique called *nonlinear PCA* [18]. This however requires a function that transforms the initial dataset to an entirely linearly dependent representation, which can be difficult to find, and in some cases is impossible.

[1] In practice this is not entirely true, since the search for $\mathbf{w}_{(i)}$ given by equation (2.1) cannot be implemented without some error.

2.2 t-SNE

t-distributed stochastic neighbour embedding [27], abbreviated to t-SNE is one of the more widely methods for visualizing high dimensional data in usually two or three dimensions. The technique is based on stochastic neighbour embedding [12] but utilising Student's t-distribution.

2.2.1 Algorithm

t-SNE consists of two main stages. The first one is constructing a probability distribution over pairs of high dimensional objects such that similar points are assigned higher probabilities while dissimilar objects are assigned lower probability. In the second stage, t-SNE constructs a similar probability distribution over the points in the low-dimensional map, then minimizes the Kullback–Leibler divergence [16] between the two distributions with respect to the positions of points in the map. The original version of t-SNE uses Euclidean distance as a metric for similarity, however, this can be changed in popular implementations where a different metric is more appropriate.

In the first step of the algorithm, a probability distribution is constructed over the pairs of high dimensional objects. The exact formula for the probability assigned to the pair $\mathbf{x}_i$ and $\mathbf{x}_j$ is given by

$$p_{i|j} = \begin{cases} \frac{\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/2\sigma_i^2)}{\sum_{k \neq i} \exp(-\|\mathbf{x}_i - \mathbf{x}_k\|^2/2\sigma_i^2)} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \tag{2.5}$$

Van der Maaten and Hilton in their paper [27] give a very good explanation for this probability. "The similarity of data point $\mathbf{x}_j$ to data point $\mathbf{x}_i$ is the conditional probability, $p_{j|i}$, that $\mathbf{x}_i$ would pick $\mathbf{x}_j$ as its neighbour if neighbours were picked in proportion to their probability density under a Gaussian centred at $\mathbf{x}_i$."

Since this formula depends on the order of points ($p_{i|j} \neq p_{j|i}$), we define

$$p_{ij} = \frac{p_{j|i} + p_{i|j}}{2N} \tag{2.6}$$

where N is the number of data points. Note that $p_{ij} = p_{ji}$, $p_{ii} = 0$ and $\sum_{i,j} p_{ij} = 1$.

The value of σ_i is set in such a way that the *perplexity* of the conditional distribution equals the predefined perplexity value given to t-SNE as a hyperparameter. Specifically, SNE performs a binary search for the value of σ_i that produces probability distribution with a fixed perplexity that is specified by the user. This perplexity is calculated by the following formula:

$$Perplexity = 2^{-\sum_{p_{j|i}} \log_2 p_{j|i}} \tag{2.7}$$

For the lower dimensional map, a similar probability distribution is constructed over pairs of low-dimensional point pairs.

$$q_{ij} = \begin{cases} \frac{(1+\|\mathbf{y}_i - \mathbf{y}_j\|^2)^{-1}}{\sum_k \sum_{l \neq k} (1+\|\mathbf{y}_k - \mathbf{y}_l\|^2)^{-1}} & \text{if } i \neq j \\ 0 & \text{if } i = j \end{cases} \tag{2.8}$$

Herein a heavy-tailed *Student's t-distribution* is used to measure the similarity between low-dimensional points in order to allow dissimilar objects to be modelled far apart in the map.

The *Kullback-Leibler divergence* (KL-divergence) [16] of the probability distributions P and Q is then calculated as an error function:

$$\mathrm{KL}\left(P \parallel Q\right) = \sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}} \tag{2.9}$$

According to the original paper [27], the output matrix Y is initialized using the multivariate Normal distribution $\mathcal{N}(0, 10^{-4}I)$, where I denotes the K-dimensional identity matrix, and K is the dimension of the output of t-SNE. In practice, however, different initializations are utilised. One popular initialization is PCA, as it efficiently explores linear correlations between features of data.

The final locations of the points y_i in the map are determined by minimizing the Kullback-Leibler divergence of the probability distributions with respect to the points y_i. This minimization is performed by gradient descent [24]. The result of this optimization is a map that reflects the similarities between the high-dimensional inputs.

2.2.2 Advantages over PCA

t-SNE is a nonlinear dimensionality reduction algorithm and, as such, can be used on datasets with nonlinear dependencies – relationships in the data that PCA cannot capture. This property is essential in applications in computational chemistry, *de novo* molecule generation and bioinformatics, where gathered data is rarely linear in nature.

Using t-SNE does not require standardization of data, as the used metric is the distance between data points. This, however, is not an advantage in cases where the data used is already standardized.

The algorithm has many hyperparameters, such as perplexity, and all parameters of gradient descent, such as learning rate, early exaggeration and number of iterations. This means that data scientists can "tinker" with t-SNE to find an embedding suitable for their use-case.

2.2.3 Disadvantages

The greatest drawback of using t-SNE lies in its extensive time and computation requirement. This is due in part to gradient descent, but the main reason for it is the use of computationally intensive operations, which will be detailed in section 2.3. t-SNE does not scale well for larger datasets for this reason. Attempts to speed it up with *FItSNE* [17] lead to large memory consumption, making it impossible to do analysis outside of computer clusters. On large datasets, using t-SNE requires powerful computers with large amounts of RAM and strong processors, or even GPU resources to be able to embed data points to a lower dimensional representation in sensible time.

Another disadvantage of using t-SNE – which is not uniquely a property of t-SNE – is the time investment of finding optimal hyperparameters. Particularly unfit parameters result in an embedding where clusters do not even form, and points of data translate to be randomly placed in the output space. [29]

The t-SNE algorithm produces maps that do not preserve global structure of data. This means that while distances within clusters are a meaningful indicator for similarity, distances between clusters carry little to no meaning. This also means that t-SNE can translate similar points into two or more clusters.

t-SNE can practically only embed into two or three dimensions, which means it is only useful for visualization purposes, not general dimension reduction. This is still a problem for the more modern FItSNE algorithm.

2.3 UMAP

Uniform Manifold Approximation and Projection [19], or UMAP for short is a dimension reduction algorithm developed by Leland McInnes, John Healy and James Melville in 2018. It is constructed from a theoretical framework based in Riemannian geometry and algebraic topology. This technique was developed to overcome the shortcomings of t-SNE. The result is an algorithm that is competitive with t-SNE for visualization quality, arguably preserves more global structure and has superior run time performance. UMAP also has no restrictions on embedding dimension, making it suitable for general dimension reduction use cases.

The theoretical description of UMAP works in terms of *fuzzy simplicial sets*, which are higher-dimensional generalizations of directed graphs, partially ordered sets and categories. Indeed, from a practical computational perspective, UMAP can ultimately be described in terms of, construction of, and operations on, weighted graphs. This puts UMAP in the class of k-neighbour based graph learning algorithms such as Laplacian Eigenmaps [5], Isomap [26], or t-SNE.

The theoretical description of the algorithm uses a few basic assumptions. For completeness, these are:

- There exists a manifold on which the data would be uniformly distributed.
- The underlying manifold of interest is locally connected.
- Preserving the topological structure of this manifold is the primary goal.

2.3.1 Algorithm

There are many similarities between the t-SNE and UMAP algorithm. Both techniques consist of two distinct phases, which are similar in many ways and as such the most effective way to understand UMAP is to highlight the differences between the two.

The first phase of UMAP is constructing a high dimensional graph over the point of data using k-neighbour search, in order to approximate the topology of the dataset.

The first step in constructing the higher dimensional weighted graph is finding the k nearest neighbours for every observation for a given distance metric $d(\mathbf{x}_i, \mathbf{x}_j)$. This metric is *typically* Euclidean, however, most implementations offer other metrics. For every data

point, the minimum positive distance from observation i to a neighbour is calculated as ρ_i. In mathematical terms:

$$\rho_i = \min\left\{d(\mathbf{x}_i, \mathbf{x}_{i_j}) | 1 \leq j \leq k, d(\mathbf{x}_i, \mathbf{x}_{i_j}) > 0\right\} \tag{2.10}$$

After this, σ_i is computed for every data point, by solving the equation:

$$\log_2(k) = \sum_{j=1}^{k} \exp\left(\frac{-\max\{0, d(\mathbf{x}_i, \mathbf{x}_j) - \rho_i\}}{\sigma_i}\right), \tag{2.11}$$

or in a slightly different form:

$$k = 2^{\sum_{j=1}^{k} \exp\left(\frac{-\max\{0, d(\mathbf{x}_i, \mathbf{x}_j) - \rho_i\}}{\sigma_i}\right)}. \tag{2.12}$$

This computation is the counterpart of equation (2.7) of t-SNE. Note that this equation does not contain a $\log_2$ component, which is computationally expensive, resulting in the computation of σ_i being faster in the UMAP algorithm.

The weight function is defined between $\mathbf{x}_i$ and $\mathbf{x}_j$:

$$w(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(\frac{-\max\{0, d(\mathbf{x}_i, \mathbf{x}_j) - \rho_i\}}{\sigma_i}\right). \tag{2.13}$$

Note that this is weight function serves the same purpose as $p_{i|j}$ in t-SNE. A seemingly small but important difference between the two formulas is the omission of normalization. This has dramatic effect on performance, since summation and integration are computationally expensive operations.

The weighted graph G is defined whose vertices are individual observations from $\mathbf{X}$ and where for each edge (i, j) it holds that $\mathbf{x}_i$ is a nearest neighbour of $\mathbf{x}_j$, or vice versa. The weight of an edge (i, j) is the symmetrization of $w(\mathbf{x}_i, \mathbf{x}_j)$ and $w(\mathbf{x}_j, \mathbf{x}_i)$. This symmetric weight is defined as:

$$\bar{w}_{i,j} = w(\mathbf{x}_i, \mathbf{x}_j) + w(\mathbf{x}_j, \mathbf{x}_i) - w(\mathbf{x}_i, \mathbf{x}_j) \cdot w(\mathbf{x}_j, \mathbf{x}_i). \tag{2.14}$$

Note that this G graph is analogous to t-SNE's P probability distribution matrix.

The second phase of UMAP is the construction and optimization of output matrix (graph) Y. The initialization is performed canonically by using spectral embedding. As with t-SNE, implementations of UMAP usually offer different initialization options.

UMAP uses a force directed graph layout algorithm to optimize Y. This algorithm applies attractive forces along the edges in G and repulsive forces along $\bar{G}$, the complement of G. For every edge (i, j) in G, the attractive force is defined as:

$$\alpha \cdot \frac{-2ab\|\mathbf{y}_i - \mathbf{y}_j\|_2^{2(b-1)}}{1 + a\left(\|\mathbf{y}_i - \mathbf{y}_j\|_2^2\right)^b} \bar{w}_{i,j}(\mathbf{y}_i - \mathbf{y}_j), \tag{2.15}$$

while for every edge (i, k) that is not in G, the following repulsive force is calculated:

$$\alpha \cdot \frac{b}{\left(\epsilon + \|\mathbf{y}_i - \mathbf{y}_k\|_2^2\right)\left(1 + a\left(\|\mathbf{y}_i - \mathbf{y}_k\|_2^2\right)^b\right)} \left(1 - \bar{w}_{i,k}\right)\left(\mathbf{y}_i - \mathbf{y}_k\right), \tag{2.16}$$

where ϵ is a small positive constant, to prevent division by zero.

The hyperparameters a and b are tuned using the data by fitting the function $\left(1 + a\left(\|\mathbf{y}_i - \mathbf{y}_j\|_2^2\right)^b\right)^{-1}$ to the non-normalized weight function $\exp\left(-\max\{0, d(\mathbf{x}_i, \mathbf{x}_j) - \rho_i\}\right)$ with the goal of creating a smooth approximation. The hyperparameter α indicates the learning rate. This family of curves is similar to Student's t-distribution used by t-SNE (2.8), but without integration in the denominator. This speeds up the calculation.

The last major difference between the t-SNE algorithm and UMAP is that while the former uses regular gradient descent [24], UMAP uses stochastic gradient descent [8]. This both speeds up the computations and consumes less memory.

2.3.2 Advantages

UMAP has quite a few advantages over PCA and t-SNE. As a nonlinear dimension reduction algorithm, UMAP is able to discover nonlinear correlations in the dataset, just like t-SNE. However, while practical application of t-SNE is only feasible in two or three dimensions due to tree-based algorithms for nearest neighbour search such as Barnes-Hut simulation [4], UMAP has no such limitations, making it useful for general dimension reduction use-cases.

While UMAP also has many hyperparameters, these are more intuitive than those of t-SNE. Specifically, the perplexity parameter of t-SNE is only indirectly affecting σ_i, while UMAP's `n_neighbour` parameter has a direct effect on σ_i. This means that users can more intuitively tune the algorithm for their use-cases.

In terms of performance, UMAP has been designed to run faster than t-SNE by omitting computationally expensive calculations and replacing them with similar but more easily executable formulas. This approach does not sacrifice the quality of embedding while simultaneously speeding up calculations.

UMAP also consumes less memory overall, since instead of gradient descent, it uses stochastic gradient descent, so it needs to keep the gradients for only a subset of the observations.

The last and often overlooked advantage of UMAP lies in its derivation. UMAP is rigorously built from mathematical foundations, proving every step of the algorithm. As opposed to all other algorithms in this paper (with the exception of PCA), UMAP's steps are mathematically proven to work, and do not rely on a heuristic approach.

2.3.3 Shortcomings

As do all dimension reduction algorithms, UMAP has certain disadvantages when used on some datasets. Similarly to t-SNE, UMAP is a *near-sighted* algorithm, meaning that while it is very capable of preserving the local structure of the input space, it struggles to preserve global structure. This, however, is not as severe in UMAP as in t-SNE, since the hyperparameter `n_neighbour`, which determines the number of nearest neighbours in the initial phase of the algorithm when set to large values, results in an embedding with

more global structure preserved. This, however, can increase the memory usage of the algorithm.

The computational representation of k-nearest neighbour search builds a matrix with $N \times N \times k$ values. With large datasets, increasing k leads to large memory requirements, especially since with large values of N, preserving global structure requires an even larger k value than for a smaller N. In practice, this leads to the projection of large datasets, resulting in less global structure preservation.

2.4 TriMAP

Both t-SNE and UMAP are considered near-sighted algorithms, meaning they preserve only local structure as opposed to global structure. While PCA is designed to preserve global structure, it suffers from the limitations of nonlinear correlations. TriMAP [1] is a graph-based nonlinear dimension reduction algorithm that aims to improve global structure preservation by using triplets of data points instead of pairs.

2.4.1 Algorithm

The algorithm defines triplets (i, j, k) such that $d(\mathbf{x}_i, \mathbf{x}_j) < d(\mathbf{x}_i, \mathbf{x}_k)$. A subset of all triplets $\mathcal{T} \coloneqq \{(i, j, k)\}$ is used to approximate the structure of the dataset.

$\mathcal{T}$ is constructed in the following way:

- For each observation i, `n_inliers` nearest neighbour is found according to the distance metric used.
- For every sample i and one of its neighbour j, `n_outliers` points k are sampled that are not neighbours of i.
- For each point i, two points are randomly sampled to make a triplet. This is repeated `n_random` times.
- This results in set $\mathcal{T}$ that contains $|\mathcal{T}| = (\texttt{n\_inliers} \cdot \texttt{n\_outliers} + \texttt{n\_random}) \cdot N$ triplets.

The loss function is defined in multiple steps. The first one is the definition of $s(\mathbf{y}_i, \mathbf{y}_j)$ the following way:

$$s(\mathbf{y}_i, \mathbf{y}_j) = \left(1 + \|\mathbf{y}_i - \mathbf{y}_j\|^2\right)^{-1} \tag{2.17}$$

This is very similar to UMAP's formula for finding its hyperparameters a and b, and to t-SNE's Student's t-distribution.

A weight $w_{i,j,k}$ is defined for every triplet.

$$\omega_{i,j,k} = \log\left(1 + 500\left(\frac{e^{d_{i,k}^2 - d_{i,j}^2}}{\max_{(i',j',k') \in \mathcal{T}} e^{d_{i',k'}^2 - d_{i',j'}^2}} + 10^{-4}\right)\right), \tag{2.18}$$

where $d_{i,j}^2$ is defined as

$$d_{i,j}^2 = \frac{d^2(\mathbf{x}_i, \mathbf{x}_j)}{\sigma_i \sigma_j}, \tag{2.19}$$

and σ_i is the average distance between $\mathbf{x}_i$ and the set of its 4–6 nearest neighbours.

Intuitively, $\omega_{i,j,k}$ is larger when the distances in the tuple are more significant, suggesting that it is more important to preserve this relation in the low-dimensional space.

For every triplet, a certain loss value can be calculated:

$$l_{i,j,k} = \omega_{i,j,k} \frac{s(\mathbf{y}_i, \mathbf{y}_k)}{s(\mathbf{y}_i, \mathbf{y}_j) + s(\mathbf{y}_i, \mathbf{y}_k)} \tag{2.20}$$

Note that $\omega_{i,j,k}$ is a static value which is only needed to be calculated once based on the higher dimensional input triplets. $s(\mathbf{y}_i, \mathbf{y}_j)$ approaches 1 then $\mathbf{y}_i$ and $\mathbf{y}_j$ are closer and approaches 0 if the points are very distant. This implies that the fraction in the triplet loss function would approach the maximal value of 1 when i is placed close to k and far away from j (which contradicts the definition of a triplet (i, j, k) where i should be closer to j than to k). Otherwise, as k moves further away, the loss approaches 0.

For the total loss of the embedding the individual triplet loss values are summed.

$$l_{\text{TriMAP}} = \sum_{(i,j,k) \in \mathcal{T}} l_{i,j,k} \tag{2.21}$$

As with t-SNE, TriMAP uses full batch gradient descent to optimize the low dimensional embedding. The initialization of points in lower dimensional space is done via PCA initialization.

TriMAP intuitively tries to find a representation of data that preserves the ordering of distances within a subset of triplets. These triplets are mostly made up of a central point and one of its nearest neighbours along with another non-neighbour point, with a few triplets containing randomly chosen observations. Admittedly, the algorithm contains many design choices that work empirically, but it is not evident which of these choices are essential to creating good visualizations.

2.4.2 Benefits

The main benefit of TriMAP is the preservation of the global structure of the dataset. While t-SNE, and to some extent, UMAP struggle to find an embedding that faithfully captures the global structure of the input, TriMAP can find such an embedding with little difficulty.

The loss function of TriMAP uses very computationally inexpensive operations, which in theory makes the algorithm faster than even UMAP. This theoretical advantage is entirely negated by the implementations available. As of writing this paper, UMAP has both a multithreaded and a GPU implementation, while TriMAP only has one single-threaded implementation.

2.4.3 Drawbacks

Since the sampled triplets contain mostly neighbours with another distant point, with some randomly chosen triplets mixed in, *theoretically* TriMAP preserves both local and global structure within the dataset. In practice, however, it is typically found to be prone to struggling with local structure.

The main hyperparameters of the algorithm should control the focus on global versus local structure, however, in practice, these parameters change little on the outcome of the algorithm.

2.5 PaCMAP

Pairwise Controlled Manifold Approximation Projection, or PaCMAP [28] for short is the newest dimension reduction algorithm out of the examined ones. It was designed in 2020 by studying the previous three algorithms, uncovering a common interpretation of all of them being graph-based algorithms with nodes being observations and edges constituting a similarity metric between these observations. With this insight, the writers define a set of principles of a good dimension reduction algorithm. They investigate the choice of loss function, the initialization and the algorithm's robustness to initialization. They also investigate the optimization of lower dimensional representation to be able to capture both local and global structure.

The outcome of this investigation is the algorithm called PaCMAP, which satisfies all the principles set by the authors.

2.5.1 Algorithm

Similarly to t-SNE and UMAP, PaCMAP uses pairs of observations to approximate the topology of the dataset. These pairs are sorted into three distinct groups:

- Near pairs: Pair i with its nearest n_{NB} neighbours defined by the scaled distance $d_{ij}^{2,\text{select}}$. This distance is defined in equation (2.22). Algorithmically, this selection comes from first performing a k-nearest neighbour search on the dataset with $k = n_{NB} + 50$ and then selecting the subset of this where the scaled distance is lowest.
- Mid-near pairs: For every observation i, six other points from the dataset is sampled. The second closest one id paired with i. The number of mid-near pairs is proportional to the number of near pairs, and their ratio is a hyperparameter of the algorithm. $n_{MN} = \lfloor n_{NB} \times MN\_ratio \rfloor$
- Further pairs: These are constructed by sampling non-neighbours. The number of such pairs is $n_{FP} = \lfloor n_{NB} \times FP\_ratio \rfloor$

The scaled distance metric is defined as

$$d_{i,j}^{2,\text{select}} = \frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\sigma_i \sigma_j}, \tag{2.22}$$

where σ_i is the average distance between observation i and its Euclidean nearest fourth to sixth neighbours. These values are not computed for every observation since only a subset

of them is used in the algorithm. This equation is very similar to TriMAP's formula for calculating the square distance between pairs in equation (2.19).

The embedding matrix $\mathbf{Y}$ is initialized by using PCA. In practice, other initializations are permitted by implementations, should the need arise.

For the loss function, the following shorthand is used:

$$\tilde{d}_{ij} = \|\mathbf{y}_i - \mathbf{y}_j\|^2 + 1 \tag{2.23}$$

As for the full loss function:

$$\begin{aligned} \text{Loss}^{\text{PaCMAP}} = & w_{NB} \cdot \sum_{i,j \text{ are neighbours}} \frac{\tilde{d}_{ij}}{10 + \tilde{d}_{ij}} \\ & + w_{MN} \cdot \sum_{i,k \text{ are mid-near pairs}} \frac{\tilde{d}_{ik}}{10000 + \tilde{d}_{ik}} \\ & + w_{FP} \cdot \sum_{i,l \text{ are further points}} \frac{1}{1 + \tilde{d}_{il}}. \end{aligned} \tag{2.24}$$

The optimization consists of three distinct stages. In each stage w_{NB}, w_{MN} and w_{FP} are set differently to allow focus on more local or global structural preservation. In the first stage through iterations τ_1 to τ_2 the weights are set by the following equations:

$$\begin{aligned} w_{NB} = & \quad 2 \\ w_{MN}(t) = & \quad 1000 \cdot \left(1 - \frac{t-1}{100}\right) + 3 \cdot \frac{t-1}{100} \\ w_{FP} = & \quad 1 \end{aligned} \tag{2.25}$$

In this phase, the weight of the mid-near pairs gradually decreases to transition from focusing on global structure to focusing on local structure.

In the second phase, from τ_2 to τ_3, the values of the weights are:

$$\begin{aligned} w_{NB} = & \quad 3 \\ w_{MN} = & \quad 3 \\ w_{FP} = & \quad 1 \end{aligned} \tag{2.26}$$

At this stage, the goal is to improve the local structure while maintaining the global structure captured during the first phase by assigning a small (but not zero) weight for mid-near pairs.

Together, the first two phases try to avoid local optima using a process that bares similarities with simulated annealing and the "early exaggeration" technique used by t-SNE. However, early exaggeration places more emphasis on neighbours, rather than mid-near points, whereas PaCMAP focuses on mid-near pairs first and neighbours later.

In the third phase, for the rest of the iterations, the weights are set to

$$\begin{aligned} w_{NB} = & \quad 1 \\ w_{MN} = & \quad 0 \\ w_{FP} = & \quad 1. \end{aligned} \tag{2.27}$$

By reducing the weight of mid-near points to zero and lowering the effects of near points, the repulsive force of further points is magnified, helping the separation of clusters to make borders clearer. This stage has a greater effect on datasets with primarily local structure.

According to the original paper, gradient descent is used to optimize the embedding. Adam [14] optimizer is used to regulate this procedure.

2.5.2 Strengths

PaCMAP has been designed with all previous methods in mind, collecting their strengths and weaknesses to create the best algorithms. In general dimension reduction, PaCMAP delivers a good middle ground between preserving only local and only global structure.

2.5.3 Weaknesses

The principles on which PaCMAP is built are not proven theorems, but only assumptions. They are made from intuitive reasoning. However, no rigorous mathematical proof backs them up.

Similarly to TriMAP, PaCMAP currently only has one implementation that uses only one single CPU core. This puts PaCMAP at a disadvantage in performance when tested against GPU implementations of t-SNE and UMAP. This is not an intrinsic weakness of the algorithm, as PaCMAP can theoretically be parallelized.

2.6 Theoretical comparison of algorithms

We have seen that the different algorithms were designed with different goals in mind. While one algorithm aims to reduce the numbers of dimensions with retaining as much of the variance of the original dataset, another makes effort to faithfully preserve the local and global structure of its observations. For selecting the best algorithm for a certain use-case however, a comprehensive comparison is helpful. In this section, I will compare the different methods, highlighting their similarities and differences. Figures shown in this section come from Yingfang Wang's paper [28].

Algorithm	Edges	Loss function
t-SNE	(i,j) pairs	$\sum_{i \neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$
UMAP	(i,j) pairs	$\bar{w}_{i,j} \log \left(z(\mathbf{y}_i, \mathbf{y}_j)\right)^{-1}, (i,j) \in G$ $(1 - \bar{w}_{i,j}) \log \left(1 - z(\mathbf{y}_i, \mathbf{y}_j)\right)^{-1}, (i,j) \notin G$ $z(\mathbf{y}_i, \mathbf{y}_j) = 1 + a \left(\lVert \mathbf{y}_i - \mathbf{y}_j \rVert_2^2\right)^b$
TriMAP	(i,j,k) triplets	$\omega_{i,j,k} \frac{s(\mathbf{y}_i, \mathbf{y}_k)}{s(\mathbf{y}_i, \mathbf{y}_j) + s(\mathbf{y}_i, \mathbf{y}_k)}$
PaCMAP	(i,j) pairs	$w_{NB} \cdot \sum_{(i,j) \in \text{NB}} \frac{\tilde{d}_{ij}}{10 + \tilde{d}_{ij}}$ $+ w_{MN} \cdot \sum_{(i,k) \in \text{MN}} \frac{\tilde{d}_{ik}}{10000 + \tilde{d}_{ik}} + w_{FP} \cdot \sum_{(i,l) \in \text{FP}} \frac{1}{1 + \tilde{d}_{il}}$

Table 2.1: Comparison of graph based methods with structure of edges and loss function used in optimizing lower dimensional graph.

With the exception of PCA, all discussed methods can be interpreted as algorithms that work on weighted graphs, with nodes being observations and edges consisting of some

similarity metric over the space of data points seen in table (2.1). These algorithms work by constructing a low-dimensional graph and then optimizing it to fit the original structure of the dataset.

As can be seen in figure (2.1), the effect of each graph-based dimension reduction algorithm is different on a given dataset. The quality of embedded space is very dependant on the hyperparameters chosen. The importance of effective parametrization is key in making a good embedding.

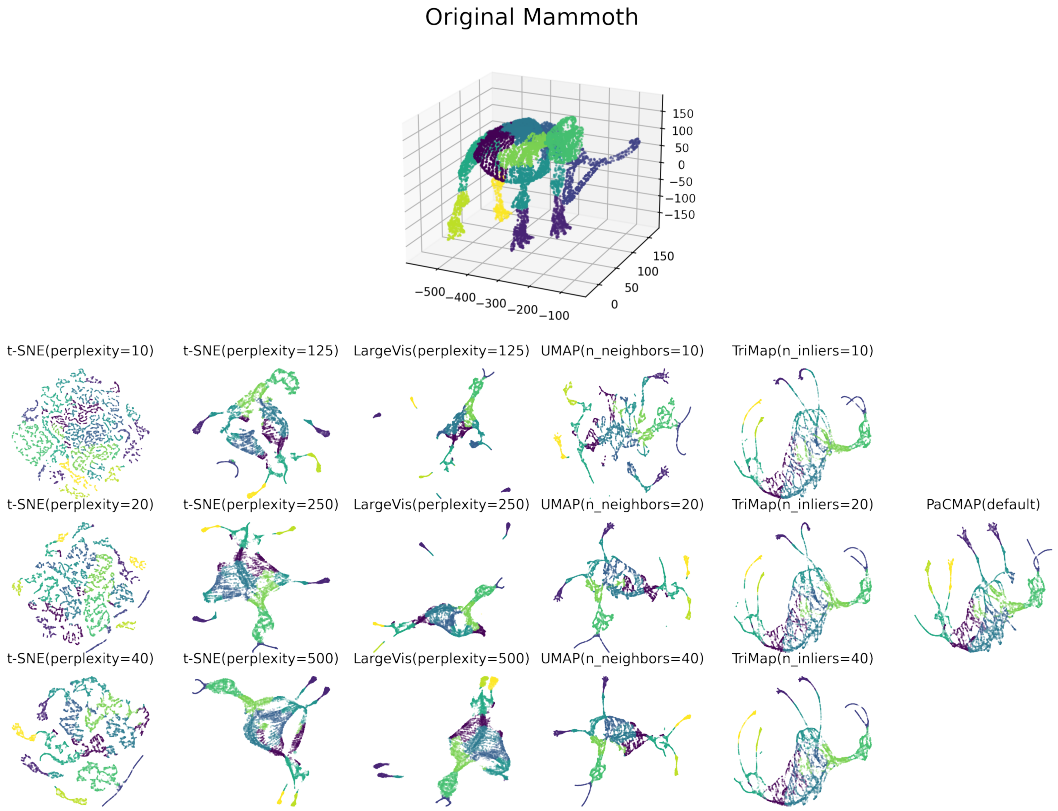

Figure 2.1: Application of t-SNE, LargeVis, UMAP, TriMAP and PaCMAP to the Mammoth dataset, with their most important parameters shown. The preservation of local versus global structure is clear from looking at small details (such as the toes and tail of the mammoth) and the overall shape of the embedding.

Chapter 3

Resources

In order to effectively explore the differences between the previously described algorithms, I needed a sufficiently large dataset and computational resources. In this chapter, I will describe the data on which I used these algorithms and the hardware resources I utilised to run the clusterings.

3.1 Used data

The data I used was the latent representation of more than 1.6 million unique SMILES strings [30]. These strings were collected from two databases and served as an input to a variational autoencoder [15] in order to embed them into a 64-dimensional latent representation that preserved some chemical structure from the kernel space to the image [21].

The preservation of chemical structure was ensured by training this model with a property predictor [11]. If this preservation of chemical structure exists, we should find that after running the previously described algorithms, the resulting two-dimensional space also has a smoothness over chemical structure.

The dataset had more features than the SMILES and latent representation of the molecules. Certain chemical descriptors, such as *quantitative estimate of drug-likeness* (qed [6]), *synthetic accessibility score* (SaS [9]) among many others. Overall, seven such chemical descriptors were included in the database along with one feature to indicate the database's origin. The last feature was an indicator of whether the data point was in the test set or not during the training of the model. These features – except for the last two – were all utilised by the property predictor of the autoencoder, so the latent space should be relatively smooth over these values.

It should be noted that the distribution of the latent space of the variational autoencoder is Gaussian in nature, therefore no standardization is needed for algorithms such as PCA for proper embedding.

3.2 Hardware resources

In terms of hardware, the requirements were quite beyond the specifications of a standard personal computer. Fortunately, I was granted access to a department server called Phoenix. This computer has 12 processing cores and 64 GB of RAM. In terms of GPU

resources, it has an nVidia TITAN Xp graphics card, which I was able to use for running a t-SNE implementation to drastically reduce the runtime of the algorithm.

At the end of the semester, I was granted access to another department computer, called Nyx, which has 48 processing units and 250 GB of RAM and also has an nVidia TITAN Xp. I did not use this server in the end, since by the time I got access to it, the development phase of the project was nearly completed. However, I plan to utilise it in the future for the continuation of the project.

3.3 Software resources

The first decision I had to make was about the development environment. The most reasonable approach was to develop in Python. This is due to the ease of development and the abundance of scientific packages, including at least one implementation of all five of the studied algorithms.

For the reading, processing and writing of data, I used the `numpy` and `pandas` libraries. For visualizing embedded latent spaces, `matplotlib` was used. The chemical computations were performed by `RDKit`[23].

For the PCA algorithm, I used the `scikit-learn==0.24.1` implementation. Two t-SNE implementation was used, `openTSNE==0.5.2` is a versatile CPU-based package, and `tsnecuda==2.1.1` is one with GPU utilization capabilities. The UMAP implementation I used was `umap-learn==0.5.1`. The TriMAP implementation was `trimap==1.0.15`. Finally, for PaCMAP, I used the package `pacmap==0.3`.

Chapter 4

Own work

In this chapter, I will describe my work, from the specification of the task through the testing process to the results. The structure of the experiments will mostly follow a chronological timeline, with the most basic questions answered first.

4.1 Defining the objective

The first problem I encountered during my work was that there is no clear definition of what a good embedding is. No simple metric exists that accurately and fully captures the nuances of embedding a latent chemical space in lower dimension. This is not surprising, since the very existence of so many different dimension algorithms comes from the fact that there is no consensus on that a good dimension reduction algorithm does. As such, I needed to define my own metrics for ranking each embedding. For this, I turned to the underlying goal of embedding these molecules.

The main objective of my thesis is to explore the capability of the previously described dimension reduction algorithms in novel drug discovery. Some assumptions must be made for the formalization of the requirements. The first such assumption is that molecules that have similar structures also have similar chemical properties. While this statement is not true for all chemical descriptors, it holds for most non-categorical metrics, such as the number of rings in the molecule, or the topological polar surface area (TPSA [22]). This intuition implies that molecules with similar structures have similar binding properties to certain proteins too, which is the most vital metric for novel drug research. Drug molecules act by binding to proteins, forming complexes that induce physiological changes in the body. The efficacy of binding depends on the chemical properties of the target protein (mainly its three-dimensional shape) and the chemical properties of the drug molecule.

From this, it logically follows that a chemical space that is ordered over the chemical structure of molecules allows the targeted search of potential drug candidates. Unfortunately, chemical structure can not be quantized in such a way as to be able to order them. Instead, I decided that what I needed was a space that *smooth* over chemical structure. In essence, this means that points close to each other have similar chemical structure. This is the exact property that is needed for targeted search. Importantly, this does not imply that points far away from each other have significantly different structures. Optimizing for embedding all similar molecules closely is a difficult task, and I can not be sure that points in the original 64-dimensional dataset even satisfies this condition. An algorithm that places all similarly structured molecules close together is advantageous, but for my purposes, it is not needed.

In summary, a good embedding is one that is locally smooth over chemical structure. This can most easily be determined by looking at chemical descriptors of molecules and assessing their local smoothness in the output space.

The choice of descriptors matters greatly in this question. The original model was trained on millions of molecules and some of their chemical properties. The properties used by the VAE's property predictor were part of the database, which means that the 64-dimensional latent space should, in theory, be relatively smooth over those metrics. Because of this, the inclusion of other chemical descriptors is needed for a proper examination of data. With mostly smooth transitions across a large number of descriptors, one can be certain that the chemical structure itself changes smoothly.

4.2 Initial testing

After defining the objective, the first thing that I did was run each algorithm with default parametrization to see baseline results. This served two purposes. Firstly, by generating baseline embeddings, the effect of different parametrizations could be more accurately assessed. Secondly, I recorded the runtime of each algorithm, so their performance could be compared from a different angle.

After running each algorithm once, I noticed that the t-SNE clustering was somewhat strange (as can be seen in figure (4.1)). The result of this run did not make much sense. It was completely different from what I expected. I had prior experience using t-SNE, albeit in another implementation. After rerunning the algorithm, I found a similar result, which led me to believe that the implementation had some bug. Indeed, when I ran t-SNE again in verbose mode, I found that instead of the 2000 iterations specified, it only ran for about 150, which was not enough to form proper clusters. I tried switching to openTSNE, the implementation I used heavily during my project laboratory. However, the sheer size of the dataset made it unusable as the system did not have enough memory to accommodate its needs.

After some consideration, I turned to tsnecuda, another familiar implementation that ran on the GPU (and, as such, used its own memory instead of the RAM of the system). I initially chose openTSNE for two reasons: firstly, it was more usable that tsnecuda, as it was far more parametrizable, allowing for custom callback functions and deterministic runs. Secondly, all other algorithms ran on the CPU, making performance comparisons more fair with openTSNE. Since openTSNE could not even complete one run, I was forced to use tsnecuda. This switch, however, highlighted something very interesting.

In the field of deep learning, GPUs are widely used as they are capable of running thousands of operations in parallel, leading to faster training of models. All graph-based clustering algorithms work very similarly to a neural network, in that they exclusively work with matrices and are highly parallelizable. This means that using GPU resources can significantly speed up the clustering of any given dataset. As can be seen on table (4.1), while the sklearn t-SNE implementation took almost exactly 25 hours to run for 150 iterations (plus constructing the higher dimensional probability distribution), tsnecuda did not even take six minutes to perform 5000 iterations (in fact, in the first run, I set it to only 2000 iterations, which ran for 102 seconds, an incredible 883 times faster).

Comparison of each algorithm based on the runtimes in table (4.1) should only be done in context. PCA is the fastest method of the bunch, this is because it is the only non-iterative algorithm. After that, the GPU implementation of t-SNE is the fastest, but this cannot be credited to the algorithm, but the implementation instead. In fact, the slowest

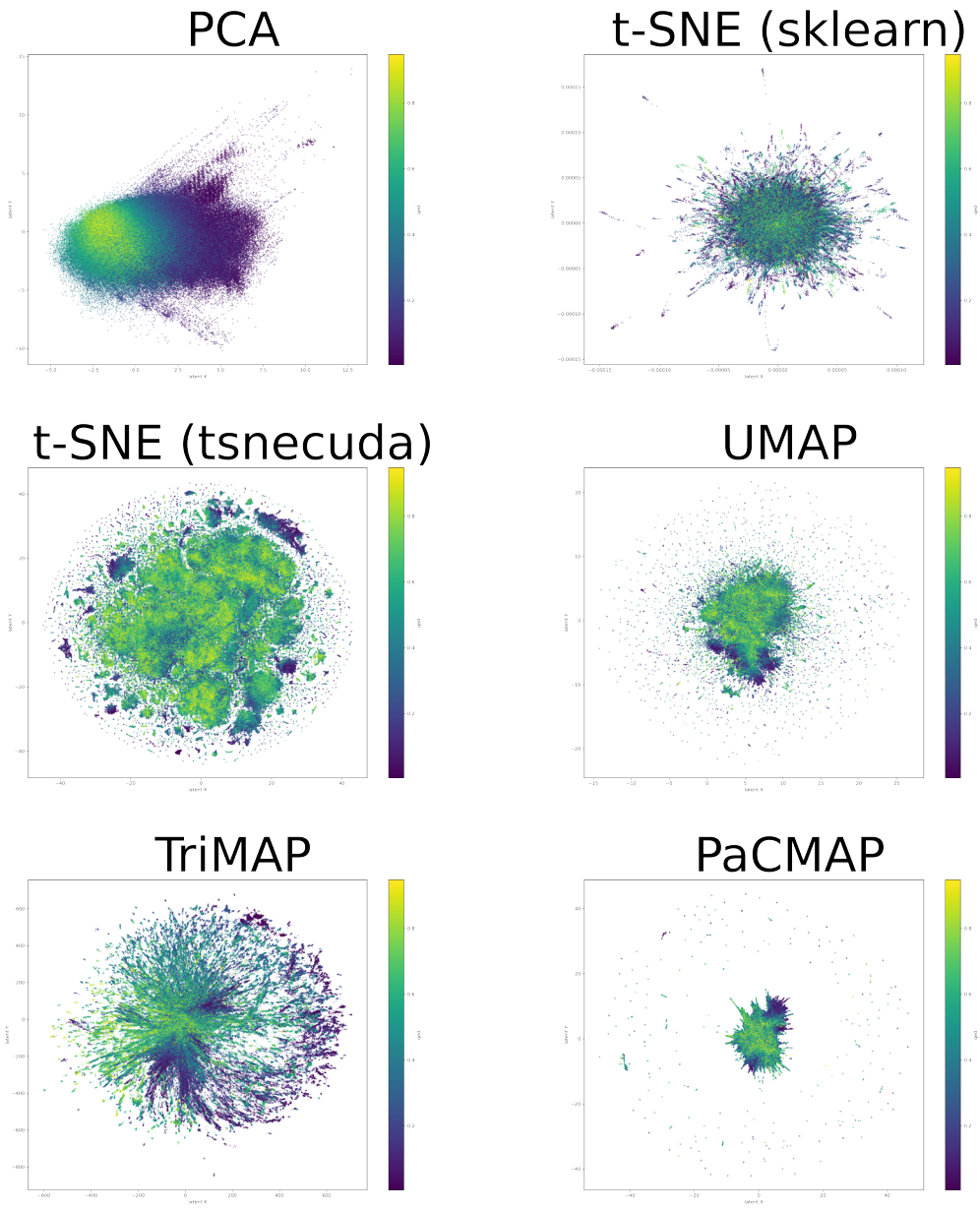

Figure 4.1: Results of PCA, t-SNE (sklearn and tsnecuda), UMAP, TriMAP, and PaCMAP algorithms on the dataset with default parameters, coloured by the quantitative estimate of drug-likeness of each data point. It can be clearly seen that each algorithm places points very differently.

running algorithm is also a t-SNE implementation, which ran on the CPU. As for the other graph-based algorithms, they are not equal either. UMAP has multithreaded capabilities, while TriMAP and PaCMAP do not.

In summary, these runtime metrics are not intrinsic properties of the algorithms. They merely describe the performance of the current implementations. In time, more libraries will be available for every algorithm, and these numbers will change. t-SNE and UMAP both have GPU implementations that vastly outperform their CPU implementations, and even other algorithms that are supposed to be faster than them. In an engineering application, it is important to consider technical parameters. It should be noted that these

Algorithm (iterations)	runtime
PCA	19.45 s
t-SNE (sklearn, 150)	90 042 s
t-SNE (tsnecuda, 5000)	344.29 s
UMAP (1000 epochs)	5 866.55 s
TriMAP (2000)	26 756.64 s
PaCMAP (2000)	65 283.52 s

Table 4.1: Runtimes of each algorithm tested on the entire dataset (without duplicates, 1.6 million molecules) given in seconds. The number of iterations and epochs are indicated where applicable. PCA, the only non-iterative method is the fastest, while the others take significantly more time. The power of GPU usage is clearly demonstrated by the two t-SNE implementations.

parameters change as opposed to each algorithm's performance in terms of the quality of the embedding.

4.3 Result of first runs

I have briefly touched on the importance of chemical descriptors in the inference of structure in section (4.1). Since the original model was trained with a property predictor, it is to be expected that the latent space is smooth over those descriptors. This can be seen in figure (4.1) with the quantitative estimate of drug-likeness (qed [6]). Additional metrics were needed to properly be able to induce chemical structure. This is needed because while structural similarity between molecules can easily be captured by one number, the structure of individual molecules is typically described by fingerprint vectors. These would form a vector space that is not easy to interpret.

The choice of descriptors matters because different metrics relate differently to chemical structure. For example, the melting point of a substance is affected mostly by the secondary bonds it can form. Consider *ethane* and *ethanol*. These two molecules are structurally very similar, only differing by one oxygen atom. This, however, changes the strongest secondary bond that can form between individual molecules. While solid ethane is bound together only by *van der Waals forces* [13], solid ethanol is held together by *hydrogen bonds* [2], many times stronger. As such, while the melting point of ethane is -182.8 °C, ethanol melts at -114.14 °C. The difference is even more dramatic for boiling points, -88.5 °C and $+78.23$ °C for ethane and ethanol, respectively.

The ideal descriptors are those that do not change very much as the structure of molecules changes a little. In the end, I chose the following metrics: quantitative estimate of drug-likeness, ring count, number of hydrogen donors, number of hydrogen acceptors, topological polar surface area [22] and number of rotatable bonds.

It should be noted that some of these descriptors do jump slightly from molecule to molecule. However, this change is not too great to cause problems in interpreting data.

As can be seen in figure (4.1), the different algorithms yielded substantially different embeddings. This is a direct result of the differences highlighted in table (2.1).

PCA produces the smoothest transition. This might seem positive, but in reality, this is not ideal. Chemical descriptors, qed in particular, are not monotonous over chemical structure.

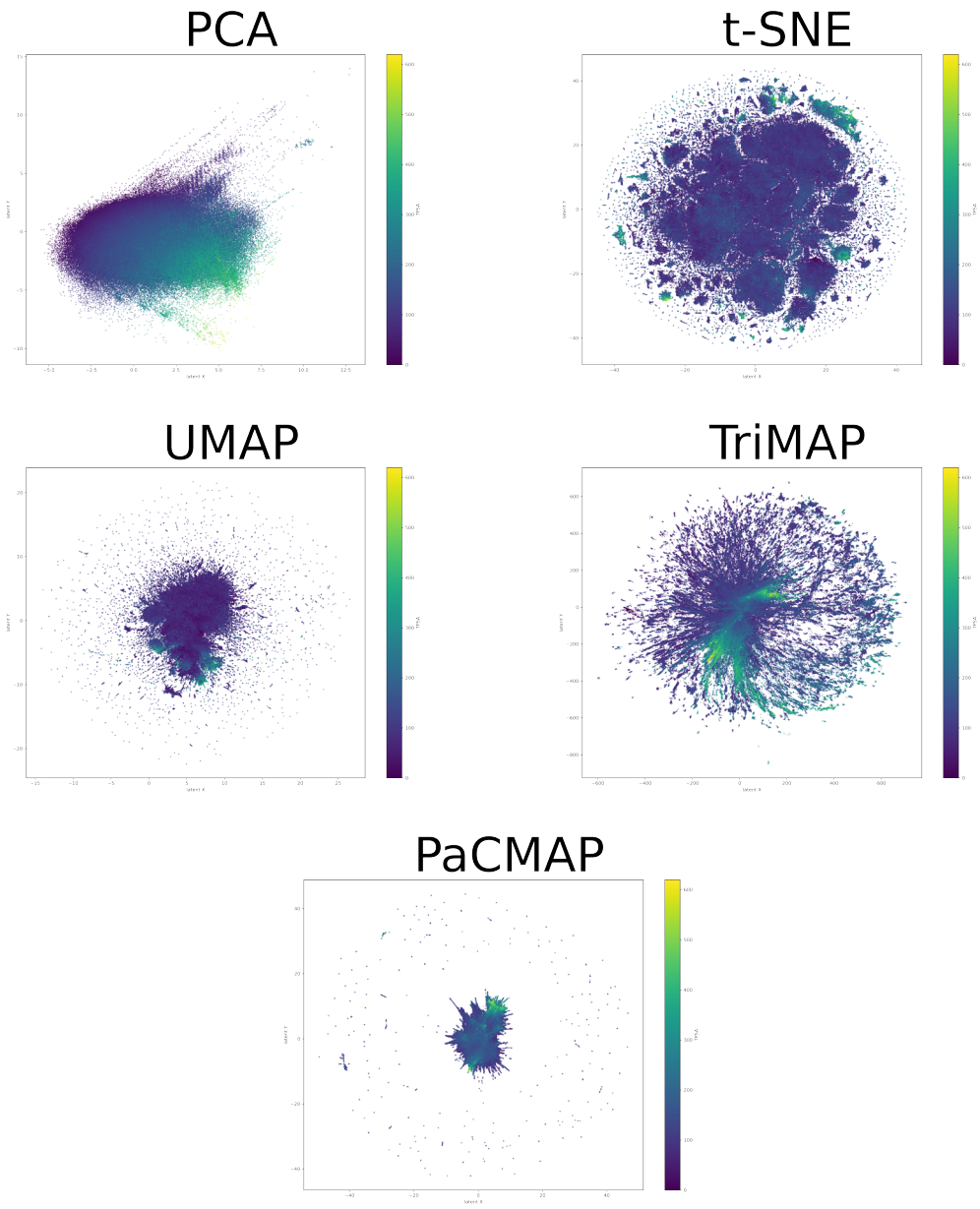

Figure 4.2: Results of PCA, t-SNE, UMAP, TriMAP and PaCMAP on the latent dataset, coloured by Topological Polar Surface Area (TPSA). This descriptor was not used by the property predictor in the making of the latent representation.

This means that while similar molecules have similar qed values, dissimilar molecules might have the exact same qed. For this reason, a complete gradient is undesirable. This stems from the fact that PCA performs a linear transformation, not allowing for the preservation of such structures.

Another interesting thing is the difference between sklearn t-SNE and tsnecuda (second and third graph, respectively). The sklearn implementation did not differentiate dissimilar points into distinct clusters. Investigating this phenomenon led to the discovery that sklearn's implementation was bugged, only iterating for a mere 150 iteration, which was not enough for separation of different molecules, as can be seen in figure (4.3). t-SNE

has the most uniform clustering in terms of density of points among the graph-based algorithms. It should be noted that t-SNE did not use the default perplexity value, as I have found previously that a perplexity parameter of around 1000 produces the most desirable output.

UMAP clusters the molecules into one big cluster (although not as tightly as PaCMAP), with a few little clusters on the side. This is partly due to the low default min_dist parameter, which makes ultra-tightly packed clusters. The space is very much smooth over qed value.

TriMAP creates objects resembling magnetic field lines. This is very interesting from a structural view. It should be noted, however, that it seems like the individual clusters overlap with each other in a way that no other algorithm resembles. For this reason, targeted search would be very difficult in the space created by TriMAP. This particular problem, as we will see, does not get better with tuning parameters, making TriMAP fundamentally unusable for this use-case.

PaCMAP – similarly to UMAP – produces one compact cluster in the middle, with some additional clusters nearby. This behaviour is not ideal for targeted search, since quite different molecules are placed near each other. PaCMAP does offer very much control over its sightedness however, and with different parameters, this can change.

In determining whether the space is smooth over the chemical structure of molecules instead of some descriptors, I investigated the embeddings by colouring the plots according to descriptors that have definitely not been used by the property predictor during the training of the VAE. As can be seen in figure (4.2), colouring by topological polar surface area, it can be stated that the algorithms produce a smooth gradient between high-valued areas and lower-valued areas. This indicates that the space is indeed smooth over the chemical structure. Multiple other descriptors have been tested, and they all yielded similar results.

Overall, with default parameters, it initially seems like t-SNE produces the most desirable embedding. TriMAP resulted in the least useful output space out of the graph-based algorithms. This will, however, change with other parameters as the values are tuned to represent the dataset faithfully.

4.4 Optimizing the parameters

As discussed previously, an advantage of parametrizable algorithms is the fact that the user is able to influence the preservation of local or global structure according to their use-case. For optimal results, each algorithm was run multiple times with different parametrizations. The choice of parameters to sweep was based on how much effect it had theoretically.

4.4.1 t-SNE

There are many common parameters of graph-based algorithms, mainly because of the use of gradient descent. Indeed, the effects of these parameters do not have significantly different effects from other methods. Because of this, I will only present the parameters associated with GD with t-SNE.

The optimization of gradient descent is in and of itself a vast topic worth many books. In fact, many papers have been written on this very topic. The graph-based algorithms all use gradient descent and all have some form of optimization laid out in their respective original

papers. For this reason, the only parameter I will show now is the number of iterations. This is one of the most important parameters of gradient descent, as too few iterations will inevitably prevent the convergence on a local minimum, while too many iterations needlessly increase the runtime of the algorithm and can even lead to overfitting, the production of an analysis that corresponds too closely or exactly to a particular set of data and may therefore fail to fit additional data or predict future observations reliably.

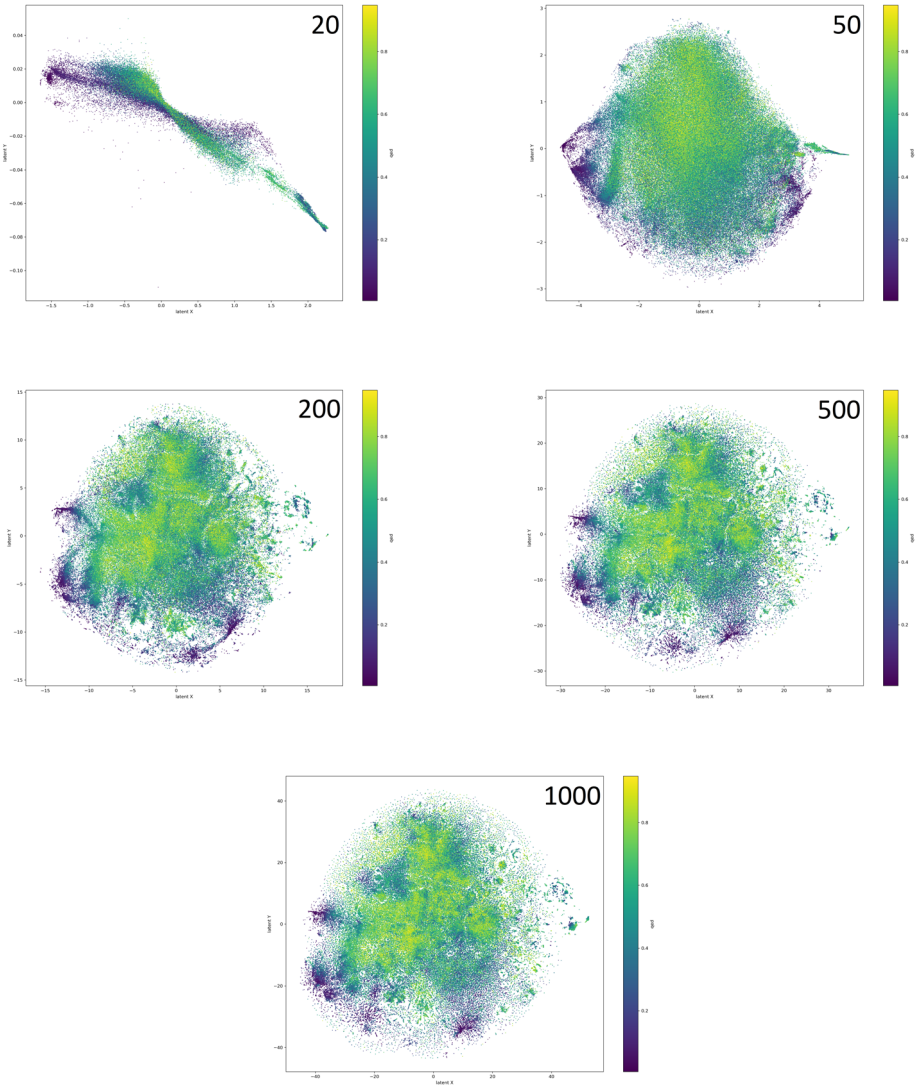

Figure 4.3: Result of the same t-SNE run after 20, 50, 200, 500 and 1000 iterations. The initial closeness of points comes from early exaggeration during gradient descent. It can be clearly seen that initial iterations change the embedding significantly, while later iterations only have minor effects.

The effect of n_iters does not scale linearly. In the initial iterations, the gradient of the error surface tends to be large, meaning that every step has a significant effect on the overall embedding. As the iterations go on, however, the output approaches a local minimum, changing less and less every step. This effect is illustrated by figure (4.3), where a significant change can be seen in the first few hundred iterations, but only minor differences emerge in the last 500 iterations.

The most important parameter of t-SNE is perplexity. As described in section (2.2), perplexity affects the σ_i used in equation (2.5). This parameter is responsible for the focus between local and global structural preservation. This focus shift is rather limited in t-SNE, as figure (4.4) illustrates.

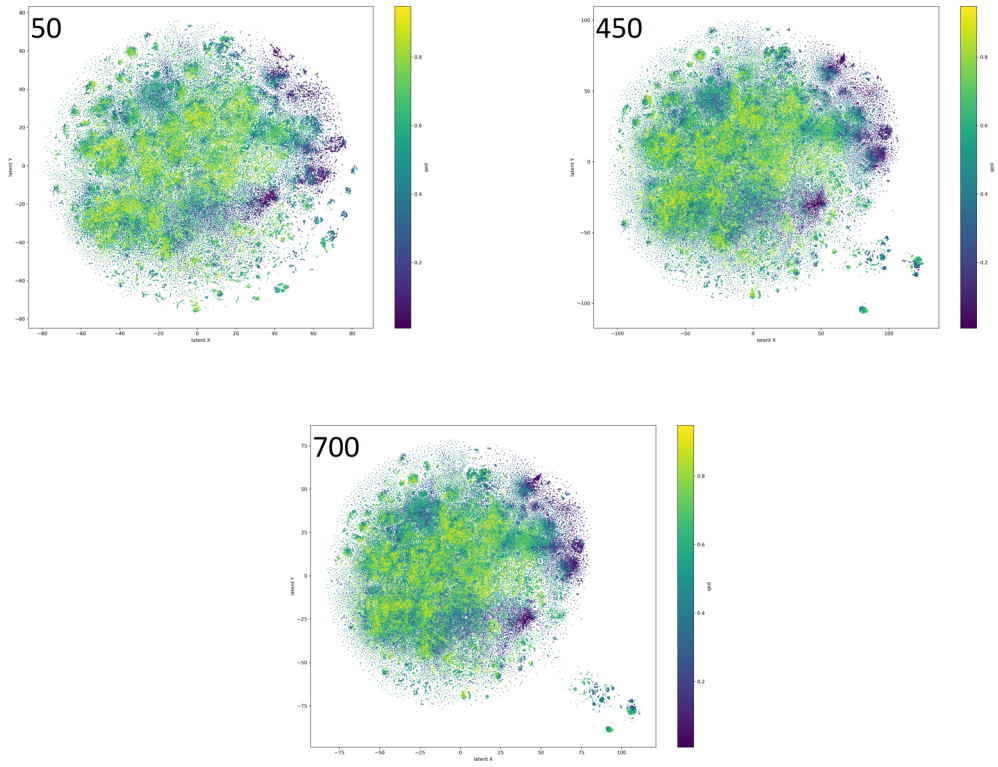

Figure 4.4: Effect of perplexity on t-SNE. The values 50, 450 and 700 were used on a subset ($n = 100000$) of the whole database. Even significantly different perplexity values generate similar embeddings.

It seems like this parameter does not have a large effect on the result. This can be attributed to the fact that perplexity does not directly affect the algorithm, only indirectly. Because of this, using the perplexity parameter does not allow fine control over the resulting embedding. It is possible that even higher, more extreme values of perplexity have more desirable effects, however, the memory consumption of k-nearest neighbour search for larger values of k – which is indirectly being increased with high perplexity – renders it practically unusable.

4.4.2 UMAP

UMAP has two particularly interesting parameters when it comes to the result of the embedding. These are min_dist and n_neighbours. The former dictates the minimum distance between points that are in the same cluster, while the latter is the number of nearest neighbours searched for in the initial stage of the algorithm.

The first parameter, min_dist has a very drastic effect on the embedding. With low values of min_dist, clusters there is no penalty for placing intra-cluster points very closely, forming highly compact clusters. This also means that different clusters are separated more clearly. With higher values, even points in the same cluster are placed further apart,

creating a more uniform distribution of points, that have less cluster separation. This effect is shown in figure (4.5).

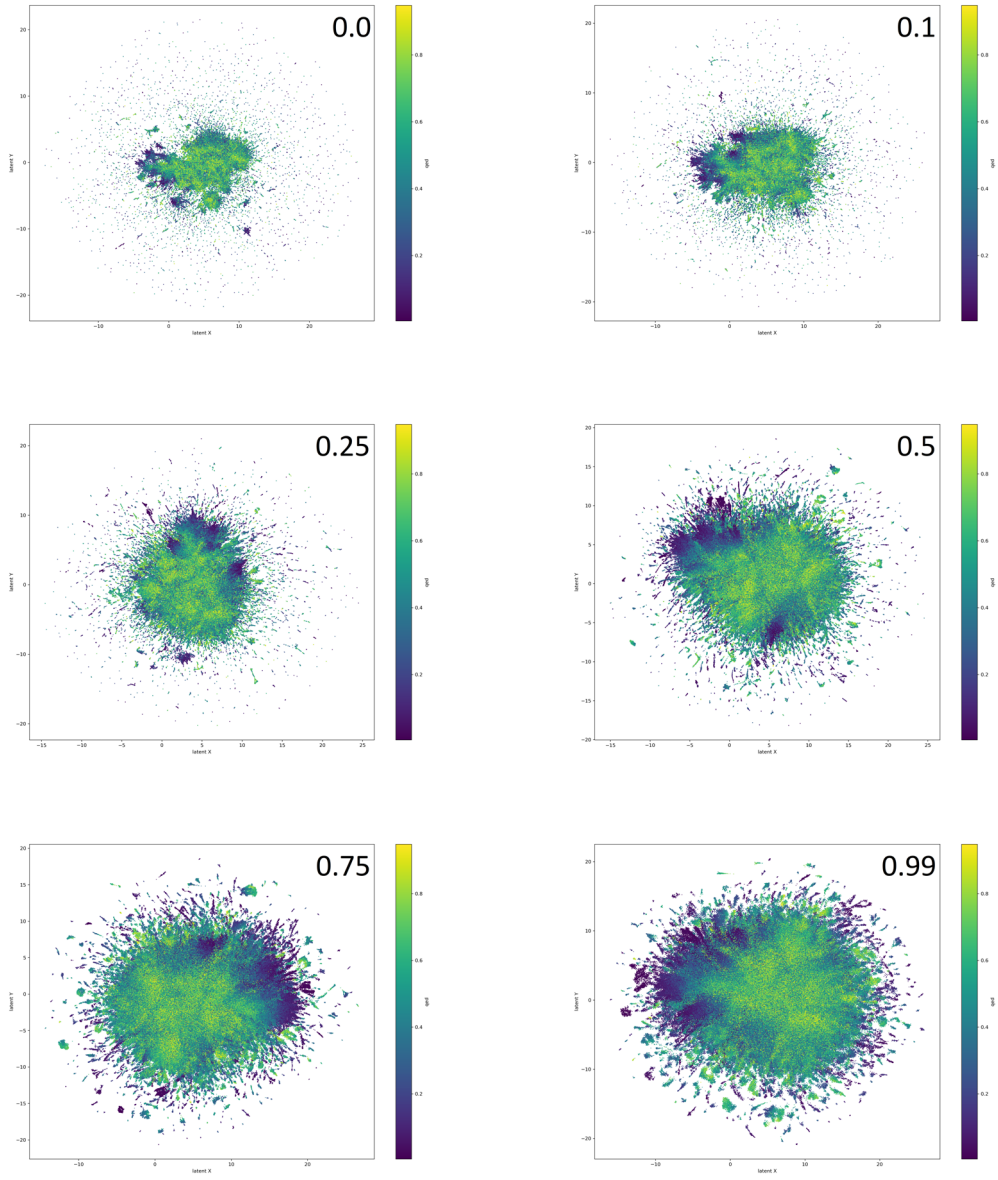

Figure 4.5: Effect of min_dist parameter on the embedding produced by UMAP. The values are in order: 0.0, 0.1, 0.25, 0.5, 0.75 and 0.99. As the value increases, so does the space between points in the same cluster, making the whole space more spread out.

While the ultra-tight clusters allow more separation of inter-cluster molecules, for the purpose of targeted search, they are undesirable. This is due to the fact that in tightly-packed clusters, sampling must be done very closely to the original point, meaning even small changes in the location of a sampled point can result in very different chemicals. This, however, can be solved by generating the low min_dist embedding and scaling every point too. The major drawback of such a procedure is that targeted searching in the scaled space results in scaled points that have to be rescaled for further use.

While the exact optimum of this parameter is not objectively measurable, according to figure (4.5) a value of around 0.75 seems to be the sweet spot for balance between tightly packed clusters and uniformly placed molecules.

The number of nearest neighbours is the parameter of UMAP that shifts the focus from local to global structure. With a larger number of nearest neighbours, a more global structure is approximated in the initial stage of UMAP. This in turn results in the lower-dimensional graph being optimized for more global structure preservation.

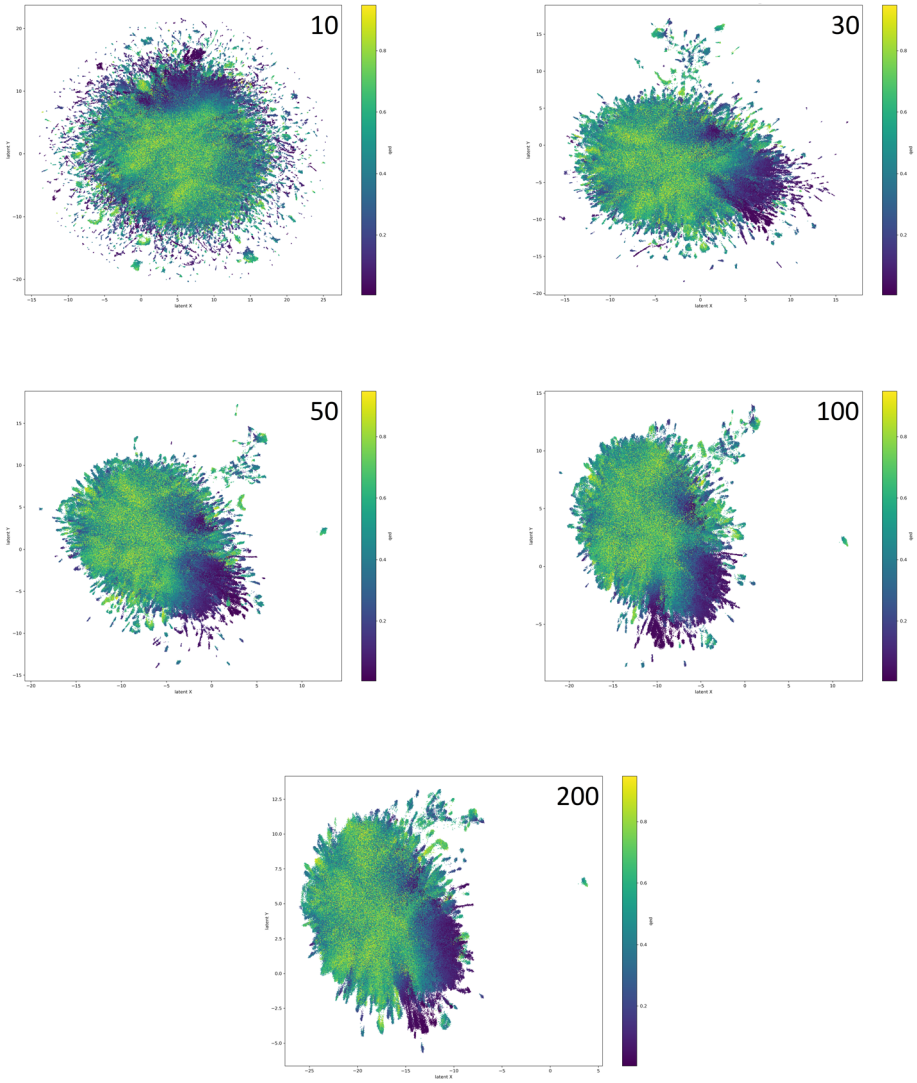

Figure 4.6: Effect of n_neighbours parameter at values of 10, 20, 30, 50, 100 and 200 with a min_dist parameter of 0.75. Higher values result in a more compact overall embedding, as more and more points are counted as nearest neighbours. At particularly high values (such as 200), very few distinct clusters are formed and all molecules are placed on top of each other, just like with TriMAP.

As can be seen in figure (4.6), the value of n_neighbours has a drastic effect when changed from a low value to a slightly higher one. However, at larger values, the difference in embedding starts to diminish. One interesting thing that can be seen is the gradual

disappearance of clusters in favour of a uniform blob. For targeted searching, this is potentially bad, since so many molecules are packed into one space. For this reason, very high values are undesirable for this use-case.

4.4.3 TriMAP

We have seen in section (4.3) that TriMAP produced very poor results for allowing targeted search. With this in mind and with the knowledge that locally-focused algorithms have performed much better, the obvious parameter choice is one (or more) that focuses on local structure. Out of the three main hyperparameters (`n_inliers`, `n_outliers`, `n_random`), the most relevant seems to be `n_inliers`.

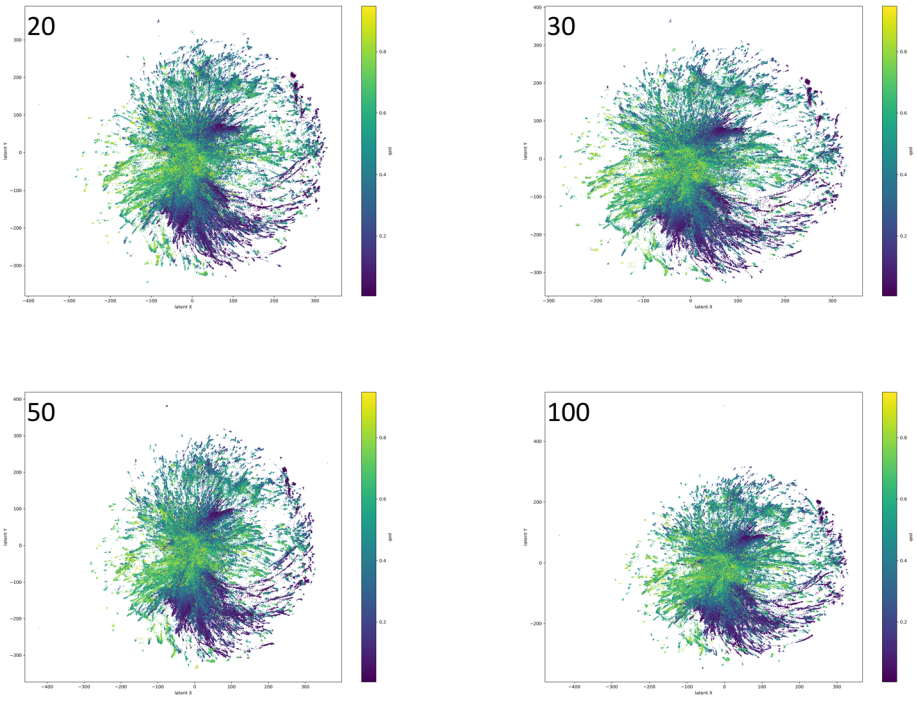

Figure 4.7: Different values of `n_inliers` (20, 30, 50 and 100) on a TriMAP embedding. This parameter did not substantially change the embedding.

After running TriMAP with a few values of `n_inliers`, the results showed very little change in the topology of the embedding (seen in figure (4.7)). The first thought I had upon seeing this result was that TriMAP was unable to focus on the local structure of the dataset. This was supported by most articles and papers I have read on the subject.

This meant that TriMAP is just unfit for the use-case of targeted searching in the latent space of drug-like molecules.

Later on, I realized that the parameter `n_inliers` serves a similar role as PaCMAP's near-pair count, influencing mostly global structural preservation. It can be seen in figure (4.7) that increasing the parameter significantly does not change the result much, specifically because TriMAP already focuses on global structure.

Unfortunately, by the time I realized this, I did not have enough time to sweep through other parameters, particularly `n_outliers`, which boost the repulsive forces exerted by

FP_ratio	Runtime (s)
0.5	1:52:43.78
7	7:14:07.76
10	9:36:32.74

Table 4.2: The runtime of a few runs with different FP_ratio parameters. There is a clear relation between the value and runtime. One should be careful setting extremely large values of this ratio.

dissimilar points. In the future, I want to see that parameter's effect on the result to give TriMAP its chance.

4.4.4 PaCMAP

PaCMAP has four main hyperparameters that change the topology of the resulting embedding. These are the number of near pairs (n_{NB}), the ratio between mid-near pairs and near pairs (MN_ratio), the ratio between further pairs and near pairs (FP_ratio) and the initialization of the lower dimensional graph. The original paper [28] describes the effect of changing each one briefly.

The paper showed that PaCMAP is rather robust to the choice of initialization, more so than all other algorithms. For this reason, I decided not to investigate this parameter, as it would have led to unfruitful optimizing.

The number of near pairs has a significant effect on the final embedding. In general, it can be said that for larger values of n_{NB}, PaCMAP preserves more global structure. This is due to the fact that increasing the nearest neighbours increases the attractive forces during optimization, resulting in less overall detail in the embedding.

MN_ratio has very little effect on the overall embedding. Only by having extreme values of it (both extremely small and large), does it affect the result. It is observed that extreme values of MN_ratio cause problems with both local and global structure. The omission of mid-near points caused the worst result, leading to no structural preservation. For computational and quality reasons, it is stated in the paper that this ratio should be fixed at the default 0.5.

The effect of FP_ratio on the resulting embedding is the exact opposite of that of n_{NB}. By sampling more further points, the repelling forces increase during optimization, leading to a more locally faithful representation of the original data.

With this in mind, I decided that the most important parameter to test was FP_ratio, as it had the most effect on preserving local structure. I used many different values for the parameter, the result of some of which can be seen in figure (4.8). Clearly this hyperparameter has an immense effect on the result of PaCMAP. With larger values of FP_ratio, the resulting space becomes more spread out, with distinct clusters forming. This is ideal for targeted searching, as the individual points are sorted into clusters based on similarity, pushing dissimilar points into other clusters.

It is important to note that this effect is not without drawbacks. As the FP_ratio parameter increases, so does runtime. This relationship is captured on table (4.2). This is because the subset of data points used by PaCMAP increases as more further points are sampled for each observation.

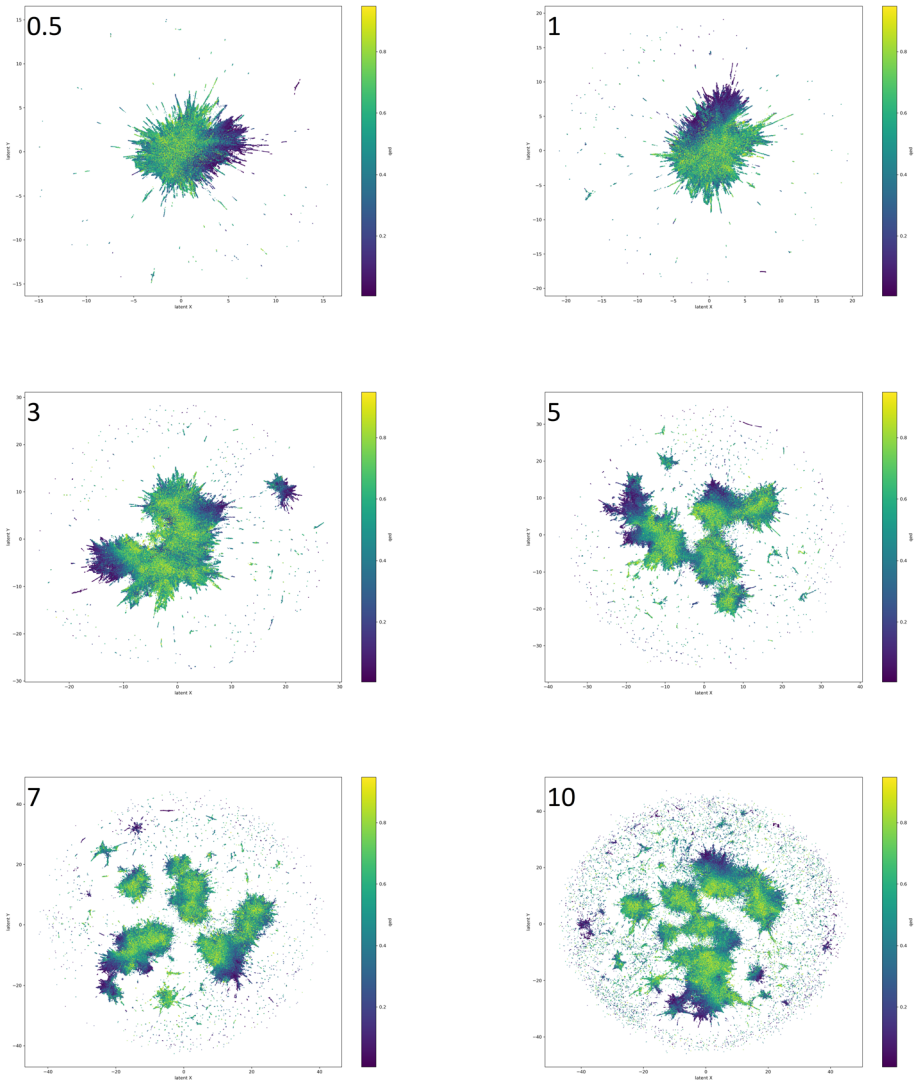

Figure 4.8: PaCMAP embeddings using various FP_ratio parameters (0.5, 1, 3, 5, 7 and 10). Higher FP_ratio resulted in more spread out clustering with less similar molecules close together. Tuning this parameter showed the most change out of all the tested non-GD parameters.

4.5 Overall results of hyperparameter optimization

Overall, it can be said that these algorithms behave very differently when their parameters are tuned. While t-SNE does not significantly change its output with different values of perplexity, UMAP and PaCMAP both show major changes with different parametrization. The potential of TriMAP is regrettably not known from this experiment.

The most potential comes with PaCMAP, as it has shifted the greatest amount from its default state. This, however, comes with a great increase in runtime. As for UMAP, while it has not shown as much tunability as PaCMAP, the fact that it runs much faster (with even GPU acceleration in certain implementations) can make it a more desirable method for common use. At this point, it seems like UMAP and PaCMAP are the most useful algorithms for *de novo* molecule design.

4.6 Linear interpolation test

So far, we have seen that each algorithm transformed the higher-dimensional latent space in a way such that it formed clusters in which molecules shared similar properties. This does imply a smoothness over chemical structure in the output space. However, it does not mean that different clusters contain dissimilar molecules. The property of the output space that ensures the aggregation of all similar molecules into one cluster, while not essential, is very useful in *de novo* molecule design, particularly in targeted searching.

In order to test the results of each algorithm for this property, I proposed a linear interpolation test, or LERP, for short. This technique consists of selecting a fixed number of points that fit on a straight line between two selected observations in high dimension. These points are then added to the end of the dataset, and the new extended dataset is run through each algorithm. This results in a new embedding with those additional points that can be traced to see how the transition between data points is mapped to the output space.

There are many considerations that need to be made with this kind of test. Firstly, the number of additional points may seem unimportant. However, it is one of the most important parameters. By choosing too many points relative to the size of the original dataset, the relations between the newly added points may overpower the underlying structure of the original dataset, transforming the embedding vastly. Choosing too few points, however, renders the test less useful, as the finer details of the mapping are not seen. In my case, I chose the number of points to be 100, which is sufficiently small for the size of the dataset (1.6 million molecules) as to not change the overall topology of the embedding while also enabling a finely detailed view of the mapping.

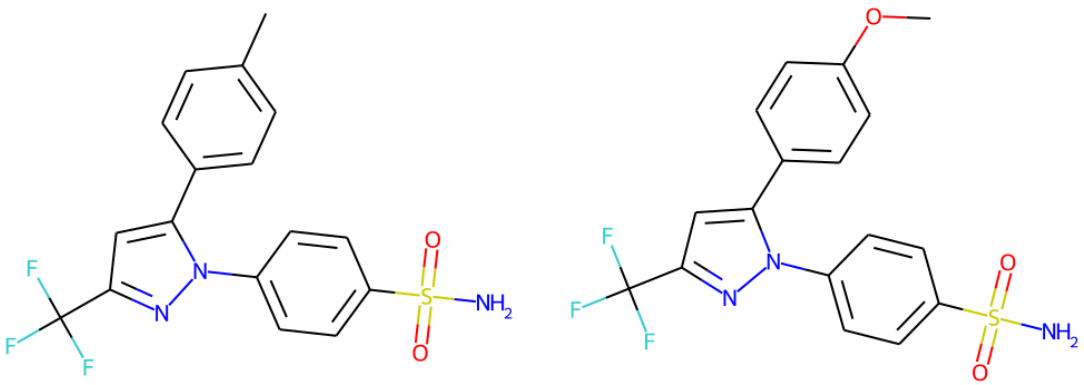

Figure 4.9: Two similar molecules (0.742 Tanimoto similarity) used in the linear interpolation test. The one on the left is celecoxib, a well-known COX-2 inhibitor.

The second parameter to consider is the choice of endpoints. Different choices of points lead to different insights learned by performing the test. By deliberately choosing similar points, one can examine whether similar points in the input space are truly mapped near each other in the output space. With a random selection, a more general trend can be observed. It shows the transition that the algorithm constructs between data points. The same can be said about deliberately choosing distant points, but that choice offers no more

insight than the random point variant. In my test, I opted to try both similar molecules (figure (4.9)) and random ones (figure (4.10)).

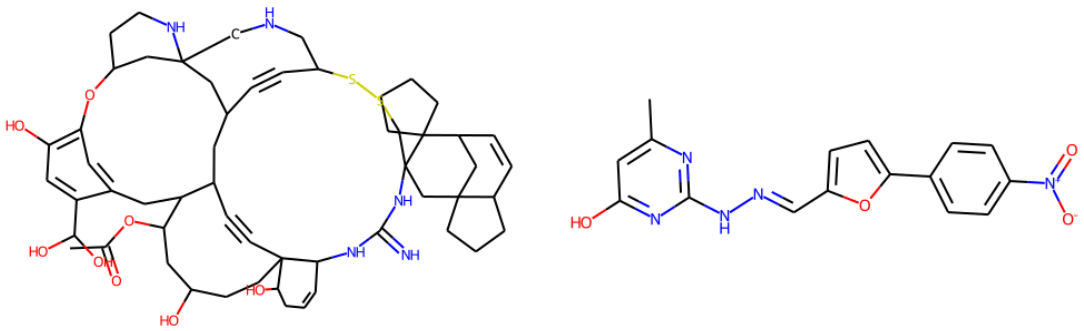

Figure 4.10: Two randomly selected molecules that were used in the linear interpolation test. The two molecules have very distinctly different structures and, as such, belong in different clusters.

Finally, while I presented this test as a LERP, there are many other ways to sample points between two observations. On datasets with different topologies, different interpolation techniques work best. In particular, *spherical linear interpolation* (SLERP) can be used as another useful method. In fact, during the training of the original autoencoder, SLERP was observed to give better results in novel molecule generation. [21] I have chosen to use LERP instead for conducting this test, as my use-case was not novel molecule sampling, and the mapping of a straight line is more interpretable than that of a circular arc.

If the mappings are not only smooth, but also relatively continuous, a curve of some kind should be drawn between the endpoints in two dimensions. With the similar molecules, it is also possible that the algorithms do not preserve the line topology, but instead transform the molecules into a cluster – perhaps along with many other molecules.

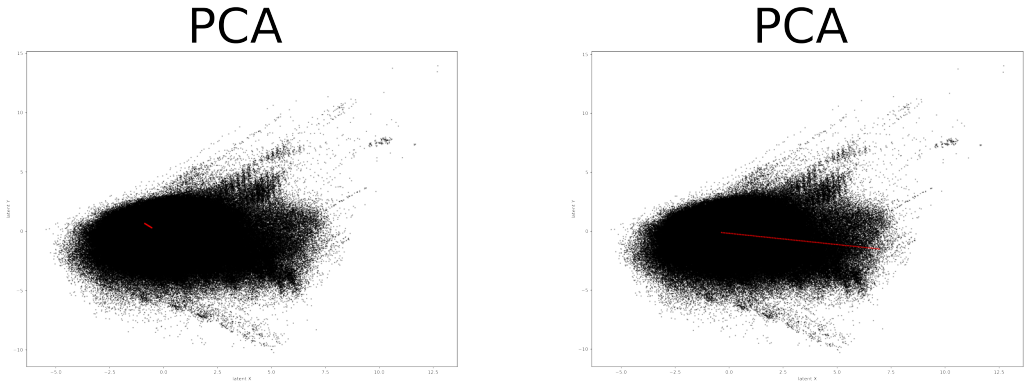

Figure 4.11: Result of linear interpolation test of PCA with similar molecules (left) and random molecules (right). The shape of the trace line is straight.

The first algorithm, PCA, transformed the interpolations into straight lines, as seen in figure (4.11). This is not surprising as PCA performs a linear transformation that maps straight lines into straight lines. It can be clearly seen that the similar molecules were transformed into points much closer on the plane than the random ones.

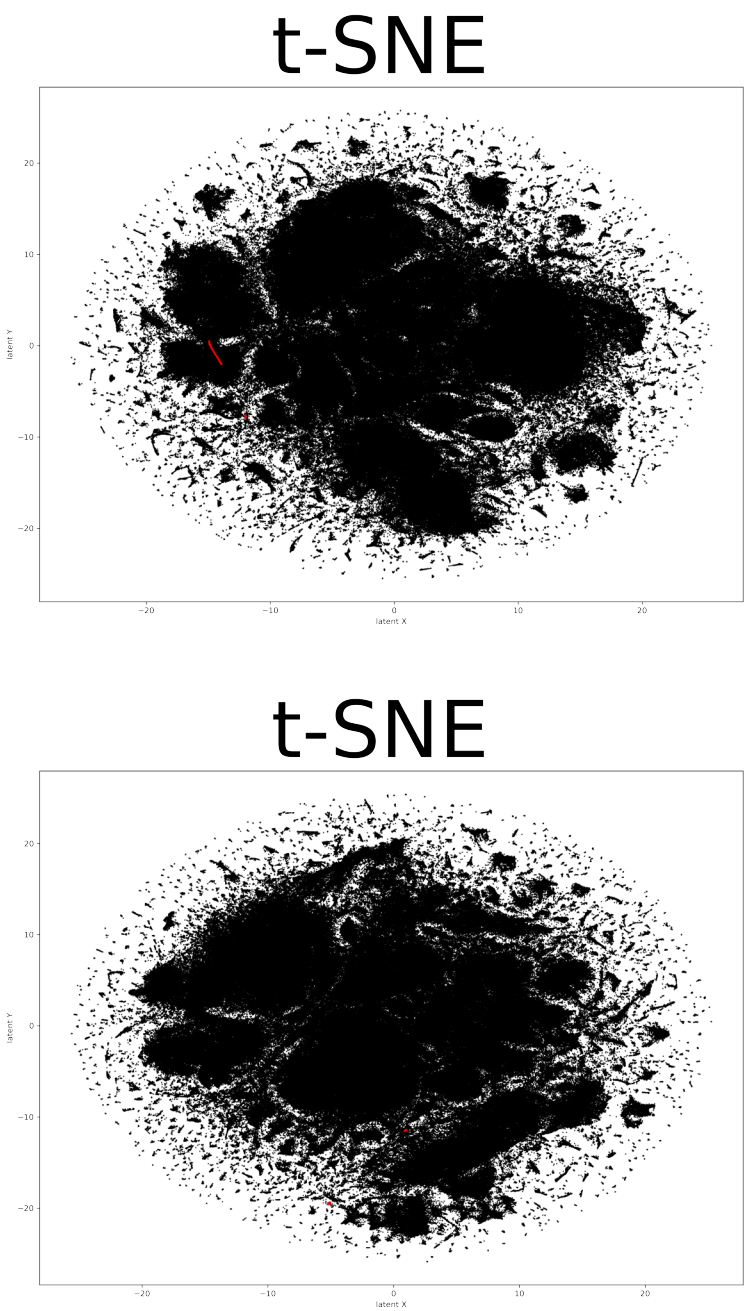

Figure 4.12: Result of linear interpolation test of t-SNE with similar molecules (upper picture) and random molecules (lower picture). In both cases, two distinct clusters were formed, but in the similar endpoints case, there was some path-tracing structure.

The results of t-SNE are significantly different from those of PCA. As can be seen in figure (4.12), t-SNE did not place the coxib molecules into the same cluster. Furthermore, the interpolated molecules are not placed in a curve between the two randomly chosen

endpoints, but rather into the clusters themselves. This results in a space where each point is surrounded by points that were close to it in the higher dimensional space. However, for targeted search, it is suboptimal, since many good candidate molecules are located in regions of the output space that are very far from the reference point. In the case of similar endpoints, while some path-tracing structure is present, the algorithm ultimately places them in two distinct clusters.

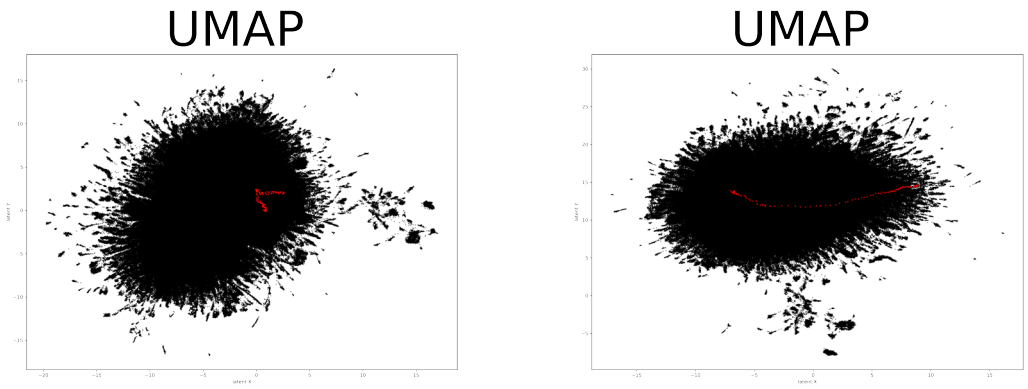

Figure 4.13: Results of UMAP LERP test. The interpolated points form a path between both the similar molecules (left) and random ones (right).

UMAP behaves very similarly when the interpolated points are between similar molecules and between random points. As can be seen in figure (4.13), in both cases, the algorithm placed the selected endpoints rather far away, and the interpolated points formed a path between them. This is a desirable property of the UMAP since the transformation from high to low dimension remains continuous, resulting in a nicer transformation. The fact that the two similar points were placed far away is not as advantageous, however. Similarly with the t-SNE embedding, if similar molecules occupy different regions in the output space, the efficacy of targeted searching decreases, as there will be some candidate molecules that cannot be found, as they are not near the target of the search.

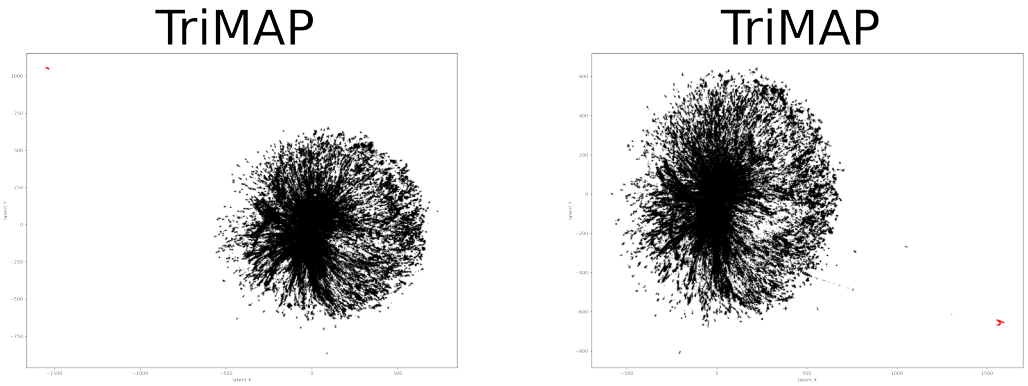

Figure 4.14: Linear interpolation test results made by TriMAP. In both cases (LERP between similar molecules on the left, random on the right), the algorithm separated the interpolated points very far from the rest of the dataset.

TriMAP resulted in the most unexpected embedding of all tested algorithms. The nature of the interpolated points was rather different from the others taken from embedding SMILES strings into a 64-dimensional latent space. This qualitative difference was picked

up by TriMAP in both the tests. The algorithm placed the added points very far from all other points in two dimensions, as figure (4.14) shows. The rest of the input data was then embedded almost identically as before, without any added points. This shows how "far-sighted" TriMAP is. Even the difference between one hundred points on a line and 1.6 million points selected from a database embedded into a latent 64-dimensional space was enough to result in the total and perfect separation of the two classes.

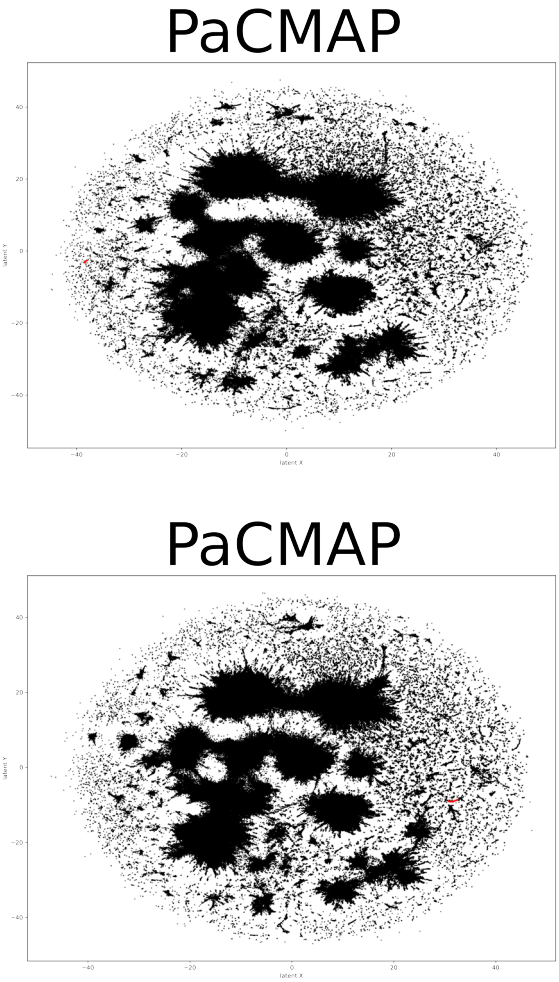

Figure 4.15: PaCMAP embedding of the LERP tests. In the first case, the coxib-like points were all placed on the left of the space in one cluster. The randomly chosen LERP resulted on two clusters being connected on the right. No change to the topology of the whole embedding is observable.

PaCMAP produced a result that could be described as expected. With an interpolation between two similar coxib molecules, the resulting embedding placed those extra points in one cluster, around the two endpoints. With random molecules, however, the interpolation points were mapped to a line connecting two distinct clusters. These clusters were not too far apart, even though the two selected molecules possessed majorly different structures. This is possibly because the interpolated points connected the clusters, forcing PaCMAP to bring them closer to each other. This, however, did not result in the topology of the whole embedding to be different from normal runs.

Chapter 5

Conclusion

The main goal of this semester's work was to investigate the legitimacy of a number of different algorithms in the field of *de novo* molecule design for reducing the dimension of a dataset in a meaningful way, allowing for targeted searching. I have learned much about each algorithm, and can confidently say which ones are particularly useful for this use-case and which ones are less so.

I have found that – as many experts already know – PCA is fundamentally unfit for such a task, as it has severe limitations in the correlations it can capture. t-SNE, which to this day is one of, if not the most popular dimension reduction algorithms, suffers from near-sightedness and performance issues both in terms of memory usage and runtime (the latter of which was not an issue because of the GPU implementation). UMAP seems to be among the most useful algorithms in this regard, transforming the chemical space in such a way as to be able to smoothly transition from one molecule to the next. In the tests, PaCMAP showed the most potential as the *de facto* go-to algorithm for pharmaceutical research. Unfortunately, the potential of TriMAP was not explored well enough for any definitive statement. However, educated guesses can be made to suggest that it is suboptimal for allowing targeted searching in its output space.

In the future, I would like to dive deeper into the optimal parametrization of each method. With more parameters to test, I believe a more desirable output space can be achieved with each method.

I would particularly like to test TriMAP more with the right parameters, because no conclusion can be made from the parameter optimization I performed.

While the LERP test was an important and fruitful examination of each algorithm's embedding quality, an arguably more important property of the implementation is the possibility of inverse transformation. That is, by sampling points from the output space, can a model return their respective positions in the original space. This would make targeted searching even more powerful, as one can simply search in the output space – an operation that is faster because of the lowered dimensionality – and produce the corresponding molecules by inverse transforming the sampled dataset.

Overall, the work I have done this semester answers quite a few questions about the validity of such dimension reduction algorithms in *de novo* molecule design, but there are still more to investigate.

Bibliography

[1] Ehsan Amid and Manfred K. Warmuth. TriMap: Large-scale Dimensionality Reduction Using Triplets. *arXiv preprint arXiv:1910.00204*, 2019.

[2] Elangannan Arunan, Gautam Desiraju, R. Klein, Joanna Sadlej, Steve Scheiner, Ibon Alkorta, David Clary, Robert Crabtree, J. Duannenberg, P. Hobza, H. Kiaergaard, Anthony Legon, Benedetta Mennucci, and David Nesbitt. Definition of the hydrogen bond. *Chemistry and Biochemistry Faculty Publications*, 83, 01 2010.

[3] Jerry Avorn. The $2.6 billion pill — methodologic and policy considerations. *New England Journal of Medicine*, 372(20):1877–1879, 2015. DOI: 10.1056/NEJMp1500848. URL https://doi.org/10.1056/NEJMp1500848. PMID: 25970049.

[4] Josh Barnes and Piet Hut. A hierarchical o(n log n) force-calculation algorithm. *Nature*, 324(6096):446–449, Dec 1986. ISSN 1476-4687. DOI: 10.1038/324446a0. URL https://doi.org/10.1038/324446a0.

[5] Mikhail Belkin and Partha Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation. *Neural Computation*, 15(6):1373–1396, 2003. DOI: 10.1162/089976603321780317.

[6] Richard Bickerton, Gaia Paolini, Jérémy Besnard, Sorel Muresan, and Andrew Hopkins. Quantifying the chemical beauty of drugs. *Nature chemistry*, 4:90–8, 02 2012. DOI: 10.1038/nchem.1243.

[7] Regine S. Bohacek, Colin McMartin, and Wayne C. Guida. The art and practice of structure-based drug design: A molecular modeling perspective. *Medicinal Research Reviews*, 16(1):3–50, 1996. DOI: https://doi.org/10.1002/(SICI)1098-1128(199601)16:1<3::AID-MED1>3.0.CO;2-6. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291098-1128%28199601%2916%3A1%3C3%3A%3AAID-MED1%3E3.0.CO%3B2-6.

[8] Léon Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998. URL http://leon.bottou.org/papers/bottou-98x. revised, oct 2012.

[9] Peter Ertl and Ansgar Schuffenhauer. Estimation of synthetic accessibility score of drug-like molecules based on molecular complexity and fragment contributions. *Journal of Cheminformatics*, 1(1):8, Jun 2009. ISSN 1758-2946. DOI: 10.1186/1758-2946-1-8. URL https://doi.org/10.1186/1758-2946-1-8.

[10] Karl Pearson F.R.S. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, 1901. DOI: 10.1080/14786440109462720. URL https://doi.org/10.1080/14786440109462720.

[11] Rafael Gómez-Bombarelli, Jennifer N. Wei, David Duvenaud, José Miguel Hernández-Lobato, Benjamín Sánchez-Lengeling, Dennis Sheberla, Jorge Aguilera-Iparraguirre, Timothy D. Hirzel, Ryan P. Adams, and Alán Aspuru-Guzik. Automatic chemical design using a data-driven continuous representation of molecules. *ACS Central Science*, 4(2):268–276, Jan 2018. ISSN 2374-7951. DOI: 10.1021/acscentsci.7b00572. URL http://dx.doi.org/10.1021/acscentsci.7b00572.

[12] 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. MIT Press, 2003. URL https://proceedings.neurips.cc/paper/2002/file/6150ccc6069bea6b5716254057a194ef-Paper.pdf.

[13] Jacob Israelachvili. *Van der Waals Forces between Particles and Surfaces*, pages 253–289. 12 2011. ISBN 9780123751829. DOI: 10.1016/B978-0-12-375182-9.10013-2.

[14] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[15] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

[16] S. Kullback and R. A. Leibler. On Information and Sufficiency. *The Annals of Mathematical Statistics*, 22(1):79 – 86, 1951. DOI: 10.1214/aoms/1177729694. URL https://doi.org/10.1214/aoms/1177729694.

[17] George C. Linderman, Manas Rachh, Jeremy G. Hoskins, Stefan Steinerberger, and Yuval Kluger. Fast interpolation-based t-sne for improved visualization of single-cell rna-seq data. *Nature Methods*, 16(3):243–245, Mar 2019. ISSN 1548-7105. DOI: 10.1038/s41592-018-0308-4. URL https://doi.org/10.1038/s41592-018-0308-4.

[18] Mariëlle Linting, Jacqueline J. Meulman, Patrick J. F. Groenen, and Anita J van der Koojj. Nonlinear principal components analysis: introduction and application. *Psychological methods*, 12 3:336–58, 2007.

[19] L. McInnes, J. Healy, and J. Melville. UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction. *ArXiv e-prints*, February 2018.

[20] S. M. Paul, D. S. Mytelka, C. T. Dunwiddie, C. C. Persinger, B. H. Munos, S. R. Lindborg, and A. L. Schacht. How to improve R&D productivity: the pharmaceutical industry's grand challenge. *Nat Rev Drug Discov*, 9(3):203–214, 03 2010.

[21] Domonkos Pogány. Adott célpontra történő gyógyszerhatóanyag generálás hatóanyag-molekulák látens teréből, 2020.

[22] S Prasanna and Robert Doerksen. Topological polar surface area: A useful descriptor in 2d-qsar. *Current medicinal chemistry*, 16:21–41, 02 2009. DOI: 10.2174/092986709787002817.

[23] RDKit, online. RDKit: Open-source cheminformatics. http://www.rdkit.org, 2013. [Online; accessed 11-April-2013].

[24] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[25] Marwin H. S. Segler, Thierry Kogej, Christian Tyrchan, and Mark P. Waller. Generating focussed molecule libraries for drug discovery with recurrent neural networks. *CoRR*, abs/1701.01329, 2017. URL http://arxiv.org/abs/1701.01329.

[26] Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000. DOI: 10.1126/science.290.5500.2319. URL https://www.science.org/doi/abs/10.1126/science.290.5500.2319.

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

[28] Yingfan Wang, Haiyang Huang, Cynthia Rudin, and Yaron Shaposhnik. Understanding how dimension reduction tools work: An empirical approach to deciphering t-sne, umap, trimap, and pacmap for data visualization. *Journal of Machine Learning Research*, 22(201):1–73, 2021. URL http://jmlr.org/papers/v22/20-1061.html.

[29] Martin Wattenberg, Fernanda Viégas, and Ian Johnson. How to use t-sne effectively. *Distill*, 2016. DOI: 10.23915/distill.00002. URL http://distill.pub/2016/misread-tsne.

[30] David Weininger. Smiles, a chemical language and information system. 1. introduction to methodology and encoding rules. *Journal of Chemical Information and Computer Sciences*, 28(1):31–36, 1988. DOI: 10.1021/ci00057a005. URL https://pubs.acs.org/doi/abs/10.1021/ci00057a005.

Appendix

A.1 Exact packages used

Name	Version	Build	Channel
_libgcc_mutex	0.1	conda_forge	conda-forge
_openmp_mutex	4.5	1_gnu	conda-forge
_tflow_select	2.1.0	gpu	
absl-py	0.11.0	py37h89c1867_0	conda-forge
annoy	1.17.0	pypi_0	pypi
argon2-cffi	20.1.0	pypi_0	pypi
astor	0.8.1	pyh9f0ad1d_0	conda-forge
async-generator	1.10	pypi_0	pypi
attrs	20.3.0	pypi_0	pypi
backcall	0.2.0	pypi_0	pypi
binutils_impl_linux-64	2.35.1	h193b22a_2	conda-forge
binutils_linux-64	2.35	h67ddf6f_30	conda-forge
blas	1.0	openblas	
bleach	3.3.0	pypi_0	pypi
boost	1.74.0	py37h6dcda5c_3	conda-forge
boost-cpp	1.74.0	hc6e9bd1_2	conda-forge
bzip2	1.0.8	h7f98852_4	conda-forge
c-ares	1.17.1	h36c2ea0_0	conda-forge
ca-certificates	2020.12.5	ha878542_0	conda-forge
cached-property	1.5.1	py_0	conda-forge
cairo	1.16.0	h7979940_1007	conda-forge
certifi	2020.12.5	py37h89c1867_1	conda-forge
cffi	1.14.5	pypi_0	pypi
cuda100	1.0	hb0b9368_0	cannylab
cudatoolkit	10.0.130	hf841e97_8	conda-forge
cudnn	7.6.5.32	ha8d7eb6_1	conda-forge
cupti	10.0.130	0	
cycler	0.10.0	py_2	conda-forge
dbus	1.13.6	hfdff14a_1	conda-forge
decorator	4.4.2	pypi_0	pypi
defusedxml	0.6.0	pypi_0	pypi
entrypoints	0.3	pypi_0	pypi
expat	2.3.0	h9c3ff4c_0	conda-forge
fcd	1.1	pyh9f0ad1d_0	conda-forge

Name	Version	Build	Channel
ffmpeg	4.3.1	hca11adc_2	conda-forge
fontconfig	2.13.1	hba837de_1004	conda-forge
freetype	2.10.4	h0708190_1	conda-forge
gast	0.2.2	py_0	conda-forge
gcc_impl_linux-64	9.3.0	h70c0ae5_18	conda-forge
gcc_linux-64	9.3.0	hf25ea35_30	conda-forge
gettext	0.19.8.1	h0b5b191_1005	conda-forge
glib	2.66.7	h9c3ff4c_0	conda-forge
glib-tools	2.66.7	h9c3ff4c_0	conda-forge
gmp	6.2.1	h58526e2_0	conda-forge
gnutls	3.6.13	h85f3911_1	conda-forge
google-pasta	0.2.0	pyh8c360ce_0	conda-forge
graphite2	1.3.13	h58526e2_1001	conda-forge
grpcio	1.36.0	py37hb27c1af_0	conda-forge
gst-plugins-base	1.18.4	h29181c9_0	conda-forge
gstreamer	1.18.4	h76c114f_0	conda-forge
guacamol	0.5.2	pyh9f0ad1d_0	conda-forge
gxx_impl_linux-64	9.3.0	hd87eabc_18	conda-forge
gxx_linux-64	9.3.0	h3fbe746_30	conda-forge
h5py	3.1.0	nompi_py37h1e651dc_100	conda-forge
harfbuzz	2.8.0	h83ec7ef_0	conda-forge
hdf5	1.10.6	hb1b8bf9_0	
icu	68.1	h58526e2_0	conda-forge
importlib-metadata	3.7.0	py37h89c1867_0	conda-forge
ipykernel	5.5.0	pypi_0	pypi
ipython	7.21.0	pypi_0	pypi
ipython-genutils	0.2.0	pypi_0	pypi
ipywidgets	7.6.3	pypi_0	pypi
jasper	1.900.1	h07fcdf6_1006	conda-forge
jedi	0.18.0	pypi_0	pypi
jinja2	2.11.3	pypi_0	pypi
joblib	1.0.1	pyhd8ed1ab_0	conda-forge
jpeg	9d	h36c2ea0_0	conda-forge
jsonschema	3.2.0	pypi_0	pypi
jupyter	1.0.0	pypi_0	pypi
jupyter-client	6.1.11	pypi_0	pypi
jupyter-console	6.2.0	pypi_0	pypi
jupyter-core	4.7.1	pypi_0	pypi
jupyterlab-pygments	0.1.2	pypi_0	pypi
jupyterlab-widgets	1.0.0	pypi_0	pypi
keras	2.3.1	py37_0	conda-forge
keras-applications	1.0.8	py_1	conda-forge
keras-preprocessing	1.1.2	pyhd8ed1ab_0	conda-forge
kernel-headers_linux-64	2.6.32	h77966d4_13	conda-forge
kiwisolver	1.3.1	py37h2527ec5_1	conda-forge
krb5	1.17.2	h926e7f8_0	conda-forge
lame	3.100	h7f98852_1001	conda-forge
lcms2	2.12	hddcbb42_0	conda-forge
ld_impl_linux-64	2.35.1	hea4e1c9_2	conda-forge

Name	Version	Build	Channel
libblas	3.8.0	17_openblas	conda-forge
libcblas	3.8.0	17_openblas	conda-forge
libclang	11.1.0	default_ha53f305_0	conda-forge
libcurl	7.71.1	hcdd3856_8	conda-forge
libedit	3.1.20191231	he28a2e2_2	conda-forge
libev	4.33	h516909a_1	conda-forge
libevent	2.1.10	hcdb4288_3	conda-forge
libffi	3.3	h58526e2_2	conda-forge
libgcc-devel_linux-64	9.3.0	h7864c58_18	conda-forge
libgcc-ng	9.3.0	h2828fa1_18	conda-forge
libgfortran-ng	7.3.0	hdf63c60_0	
libglib	2.66.7	h1f3bc88_0	conda-forge
libgomp	9.3.0	h2828fa1_18	conda-forge
libgpuarray	0.7.6	h7f98852_1003	conda-forge
libiconv	1.16	h516909a_0	conda-forge
liblapack	3.8.0	17_openblas	conda-forge
liblapacke	3.8.0	17_openblas	conda-forge
libllvm11	11.1.0	hf817b99_2	conda-forge
libnghttp2	1.43.0	h812cca2_0	conda-forge
libopenblas	0.3.10	h5a2b251_0	
libopencv	4.5.1	py37h90094e2_0	conda-forge
libpng	1.6.37	h21135ba_2	conda-forge
libpq	13.2	hfd2b0eb_2	conda-forge
libprotobuf	3.15.2	h780b84a_0	conda-forge
libssh2	1.9.0	hab1572f_5	conda-forge
libstdcxx-devel_linux-64	9.3.0	hb016644_18	conda-forge
libstdcxx-ng	9.3.0	h6de172a_18	conda-forge
libtiff	4.2.0	hdc55705_0	conda-forge
libuuid	2.32.1	h7f98852_1000	conda-forge
libwebp-base	1.2.0	h7f98852_0	conda-forge
libxcb	1.13	h7f98852_1003	conda-forge
libxkbcommon	1.0.3	he3ba5ed_0	conda-forge
libxml2	2.9.10	h72842e0_3	conda-forge
llvmlite	0.37.0	pypi_0	pypi
lz4-c	1.9.3	h9c3ff4c_0	conda-forge
mako	1.1.4	pyh44b312d_0	conda-forge
markdown	3.3.4	pyhd8ed1ab_0	conda-forge
markupsafe	1.1.1	py37h5e8e339_3	conda-forge
matplotlib-base	3.3.4	py37h0c9df89_0	conda-forge
mistune	0.8.4	pypi_0	pypi
mysql-common	8.0.23	ha770c72_1	conda-forge
mysql-libs	8.0.23	h935591d_1	conda-forge
nbclient	0.5.3	pypi_0	pypi
nbconvert	6.0.7	pypi_0	pypi
nbformat	5.1.2	pypi_0	pypi
ncurses	6.2	h58526e2_4	conda-forge
nest-asyncio	1.5.1	pypi_0	pypi
nettle	3.6	he412f7d_0	conda-forge
nomkl	3.0	0	
notebook	6.2.0	pypi_0	pypi
nspr	4.30	h9c3ff4c_0	conda-forge

Name	Version	Build	Channel
nss	3.63	hb5efdd6_0	conda-forge
numba	0.54.0	pypi_0	pypi
numpy	1.20.3	pypi_0	pypi
olefile	0.46	pyh9f0ad1d_1	conda-forge
openblas	0.3.10	0	
openblas-devel	0.3.10	0	
opencv	4.5.1	py37h89c1867_0	conda-forge
openh264	2.1.1	h780b84a_0	conda-forge
openssl	1.1.1k	h7f98852_0	conda-forge
opentsne	0.5.2	py37hda21425_0	conda-forge
opt_einsum	3.3.0	py_0	conda-forge
packaging	20.9	pypi_0	pypi
pacmap	0.3	pypi_0	pypi
pandas	1.2.2	py37hdc94413_0	conda-forge
pandocfilters	1.4.3	pypi_0	pypi
parso	0.8.1	pypi_0	pypi
pcre	8.44	he1b5a44_0	conda-forge
pexpect	4.8.0	pypi_0	pypi
pickleshare	0.7.5	pypi_0	pypi
pillow	8.1.0	py37h4600e1f_2	conda-forge
pip	21.0.1	pyhd8ed1ab_0	conda-forge
pixman	0.40.0	h36c2ea0_0	conda-forge
prometheus-client	0.9.0	pypi_0	pypi
prompt-toolkit	3.0.16	pypi_0	pypi
protobuf	3.15.2	py37hcd2ae1e_0	conda-forge
pthread-stubs	0.4	h36c2ea0_1001	conda-forge
ptyprocess	0.7.0	pypi_0	pypi
py-opencv	4.5.1	py37h888b3d9_0	conda-forge
pycairo	1.20.0	py37h01af8b0_1	conda-forge
pycparser	2.20	pypi_0	pypi
pygments	2.8.0	pypi_0	pypi
pygpu	0.7.6	py37h902c9e0_1002	conda-forge
pynndescent	0.5.4	pypi_0	pypi
pyparsing	2.4.7	pyh9f0ad1d_0	conda-forge
pyrsistent	0.17.3	pypi_0	pypi
python	3.7.10	hffdb5ce_100_cpython	conda-forge
python-dateutil	2.8.1	py_0	conda-forge
python_abi	3.7	1_cp37m	conda-forge
pytz	2021.1	pyhd8ed1ab_0	conda-forge
pyyaml	5.4.1	py37h5e8e339_0	conda-forge
pyzmq	22.0.3	pypi_0	pypi
qt	5.12.9	hda022c4_4	conda-forge
qtconsole	5.0.2	pypi_0	pypi
qtpy	1.9.0	pypi_0	pypi
rdkit	2020.09.4	py37he53b9e1_0	conda-forge
readline	8.0	he28a2e2_2	conda-forge
reportlab	3.5.60	py37h69800bb_0	conda-forge
scikit-learn	0.24.1	py37h69acf81_0	conda-forge
scipy	1.6.1	py37hf56f3a7_0	

Name	Version	Build	Channel
send2trash	1.5.0	pypi_0	pypi
setuptools	49.6.0	py37h89c1867_3	conda-forge
six	1.15.0	pyh9f0ad1d_0	conda-forge
sqlalchemy	1.3.23	py37h5e8e339_0	conda-forge
sqlite	3.34.0	h74cdb3f_0	conda-forge
sysroot_linux-64	2.12	h77966d4_13	conda-forge
tensorboard	1.15.0	py37_0	conda-forge
tensorflow	1.15.0	gpu_py37h0f0df58_0	
tensorflow-base	1.15.0	gpu_py37h9dcbed7_0	
tensorflow-estimator	1.15.1	pyh2649769_0	
tensorflow-gpu	1.15.0	h0d30ee6_0	
termcolor	1.1.0	py_2	conda-forge
terminado	0.9.2	pypi_0	pypi
testpath	0.4.4	pypi_0	pypi
theano	1.0.5	py37hcd2ae1e_1	conda-forge
threadpoolctl	2.1.0	pyh5ca1d4c_0	conda-forge
tk	8.6.10	h21135ba_1	conda-forge
tornado	6.1	py37h5e8e339_1	conda-forge
tqdm	4.58.0	pyhd8ed1ab_0	conda-forge
traitlets	5.0.5	pypi_0	pypi
trimap	1.0.15	pypi_0	pypi
tsnecuda	2.1.0	pypi_0	pypi
typing_extensions	3.7.4.3	py_0	conda-forge
umap-learn	0.5.1	pypi_0	pypi
wcwidth	0.2.5	pypi_0	pypi
webencodings	0.5.1	pypi_0	pypi
werkzeug	0.16.1	py_0	conda-forge
wheel	0.36.2	pyhd3deb0d_0	conda-forge
widgetsnbextension	3.5.1	pypi_0	pypi
wrapt	1.12.1	py37h5e8e339_3	conda-forge
x264	1!161.3030	h7f98852_0	conda-forge
xorg-kbproto	1.0.7	h7f98852_1002	conda-forge
xorg-libice	1.0.10	h516909a_0	conda-forge
xorg-libsm	1.2.3	h84519dc_1000	conda-forge
xorg-libx11	1.6.12	h516909a_0	conda-forge
xorg-libxau	1.0.9	h7f98852_0	conda-forge
xorg-libxdmcp	1.1.3	h7f98852_0	conda-forge
xorg-libxext	1.3.4	h516909a_0	conda-forge
xorg-libxrender	0.9.10	h516909a_1002	conda-forge
xorg-renderproto	0.11.1	h14c3975_1002	conda-forge
xorg-xextproto	7.3.0	h7f98852_1002	conda-forge
xorg-xproto	7.0.31	h7f98852_1007	conda-forge
xz	5.2.5	h516909a_1	conda-forge
yaml	0.2.5	h516909a_0	conda-forge
zipp	3.4.0	py_0	conda-forge
zlib	1.2.11	h516909a_1010	conda-forge
zstd	1.4.8	ha95c52a_1	conda-forge