A Survey on Recent Trends in High Dimensional Data Visualization

Alexander Kiefer, M. Khaledur Rahman
Indiana University Bloomington Department of Computer Science

Introduction

As technology has progress through the 21st century, the importance of the data it produces has become increasingly essential. With business entities paying and selling data sets and information with an ever-increasing fervor, technology has become dependent upon the data it produces in order to progress. Underlying this argument is the crux of why the field of data science has grown at such a quick rate, and that is the difficulty associated with the interpretation and understanding of the raw data produced by technology. Data and graph visualizations can provide a great depth of information with high density, making them crucial tools in conveying the underlying meaning of data to people.

Abstract

Data visualization is the process by which data of any size or dimensionality is processed to produce an understandable set of data in a lower dimensionality, allowing it to be manipulated and understood more easily by people. The dimensionality reduction algorithms which we will focus on typically fall into two groups, those which attempt to maintain the global structure of the data and those which maintain local distances over global distance. In this paper, of the algorithms discussed, TriMap falls into the first category, while UMAP, LargeVis, and t-SNE fall into the second. The focus of our paper will be on the effectiveness and applications of each of these algorithms.

Methodology

For our experiment, we will be analyzing the runtime, memory performance, and aesthetic qualities of all four algorithms. In order to measure runtime, we use thetimefunction1available in the bash shell, where the output "real" time is measured. To measure the memory usage of each algorithm over time, we will utilize thememoryprofiler2package available for Python 3 .For t-SNE, we will be using the original implementation distributed in the scikit-learn package3for Python3. For LargeVis, we also use its original implementation in C++4. For UMAP, we use the distribution available on conda-forge, which is a redistribution of the original algorithm's GitHub5. Finally, we will be using the TriMap implementation distributed on the Python Packed Index, which is sourced from the algorithms GitHubpage6.We will be running all tests on a system running RedHat Linux with an Intel(R) Xeon(R) CPU E5-2670 v3CPU running at 2.30GHz. All the tests will be run using the default parameters for each respective algorithm. For our datasets, we will be using the MNIST Digits dataset7, a standard dataset used in a large variety of machine learning applications, composed of labeled images of handwritten numbers, and the Fashion MNISTdataset8, a more modern and difficult dataset based on labeled images of clothing items. For the analysis of our results, we will be measuring the runtime and memory usage of each algorithm. We will also consider the quality of the produced embedding, considering the quality of the clustering, which is the tendency for similar points to group together, and the delineation between clusters, which is tendency for clusters to be separate from other clusters (ie. not overlap).

Results

To begin, we can see that all actual runtimes correspond with their respective theoretical runtimes, indicated in the bottom column of Table (1) for each algorithm. This tells us that the experiment ran as expected by the authors of all algorithms and is an accurate predictor of which algorithm will execute the fastest. We will analyze each individual algorithm according to their runtime, going from slowest to fastest.

With the serial version of t-SNE taking the longest out of all the algorithms, with a runtime of more than2 hours on both the MNIST Digits and Fashion MNIST data-sets, it does not provide a realistic runtime for data-sets over 100000. Though the version of t-SNE used in this experiment was the slowest of all, there are various other implementations of the algorithm that significantly improve its runtime, such as a multi-core Barnes-Hut t-SNE, CUDA t-SNE, and anchored t-SNE, decreasing runtime by up to 700 times. Looking at the memory used for t-SNE, we can see that for both data-sets, the memory usage holds at around2 gigabytes for most of the runtime, only dropping down to 1.5 gigabytes in the last fifth of the runtime. With this memory usage, t-SNE has the highest memory consumption of all algorithms tested, making it less useful in systems with a more limited capacity, though this, as before, can be mitigated by using other versions of the algorithm. Though this is true, it also has the most predictable and stable memory usage, as compared to the other algorithms. Looking at the quality of the embeddings, for the MNIST digits data, the data is clustered very well, with the points being distributed more widely over the graph, as compared to the other embeddings, while still maintaining clear delineation between clusters. In the Fashion MNIST embeddings, we see similar qualities, with slightly less delineation among clusters than in the Digits data-sets. With the goal of mitigating the crowding-out problem of dimensional reduction was successful for t-SNE.

The next fastest algorithm for both data sets was LargeVis, with a runtime of approximately 10 minutes. Though this algorithm did not provide the quickest results, its runtime show that it can scale reasonably to larger data-sets. With a peak memory usage of around 2 gigabytes occurring at the very start of the algorithm, memory usage quickly drops down and normalizes to a consistent value slightly less than 1 gigabyte. With a constant memory usage for much of its runtime, LargeVis is the second most stable algorithm according to memory usage. Finally, looking at the quality of the embeddings, we can see that for the MNIST Digits data-set, LargeVis provides very clear clustering among similar data points and a great amount of delineation between clusters. In the Fashion MNIST embedding, we also still see good clustering, however, with much less delineation between clusters, leading to a fair amount of overlap between data points.

UMAP was the second quickest algorithm tested, with its unique approach of applying topology mathematics to dimensional reduction resulting in a runtime of less than 6 minutes for both data-sets. With this computational efficiency, UMAP can easily run on datasets over 70,000 data points in a realistic amount of time. With a peak memory usage over 2 gigabytes, UMAP has the highest peak memory usage out of all algorithms tested. Furthermore, its memory usage is quite unstable, which several drops in usage paired with a series of sporadic climbs, only somewhat stabilizing at around halfway through its runtime. Finally, assessing the quality of the embeddings generated by UMAP, we can see that on the MNIST Digits dataset that its output is comparable to that of LargeVis, with high quality clustering and delineation. Moving to the Fashion MNIST dataset, we see that, just like LargeVis, the quality of the embedding has decreased, with significantly more overlap and less delineation between clusters.

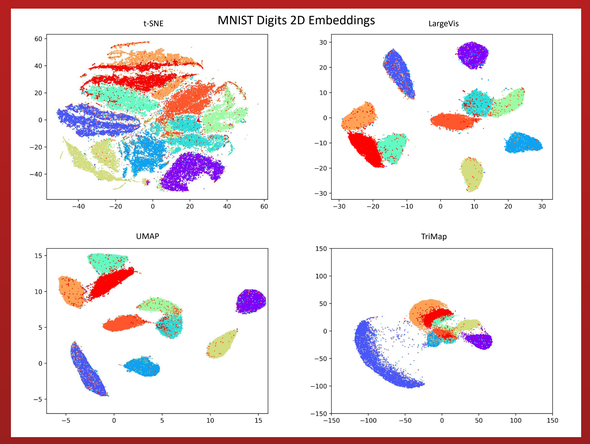
TriMap was the fastest of all algorithms by a significant margin, with it outperforming all other algorithms in terms of runtime by at least 2 times and, when compared to t-SNE, up to 60 times. With runtimes of around 2 minutes on both data-sets and a linear computational complexity, it can certainly scale far beyond the tested 70,000-member data-sets it was tested on without any trouble. With a peak memory usage of about 1.2 gigabytes, TriMap is also the most memory efficient algorithm, allowing it to scale up to larger datasets without the need for a significantly larger amount of memory. However, the stability of its memory usage is among the worst of the algorithms tested, with it being comparable to that of UMAP with its frequent spikes in usage.

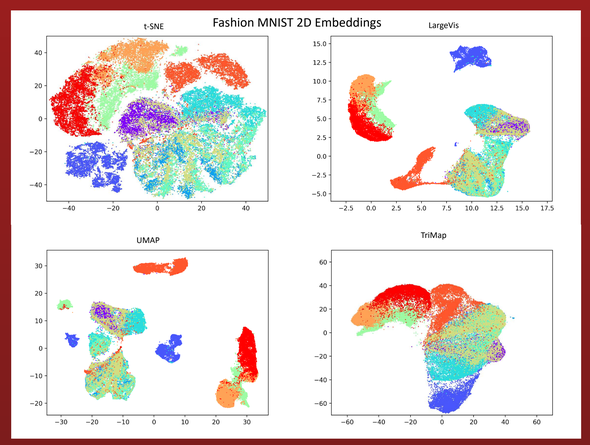
Finally, viewing the output embeddings, we can see that TriMap produces embeddings in both the MNIST Digits and Fashion MNIST data-sets that are quite different from all the others. This is most likely due to the fact that TriMap's main focus is maintaining the global structure of the data, as compared to LargeVis and t-SNE maintaining local structure and UMAP attempting to do find a medium between local and global structure. That being said, TriMap produces embeddings for both data-sets which show clear clustering, with much less delineation between clusters than other algorithms and less overlap between data points.

Visualization Program Runtimes

	TSNE	UMAP	LargeVis	TriMap
MNIST Digits Actual	129:40.967	5:06.650	9:35.643	1:39.526
Fashion MNIST Actual	132:09.443	5:42.680	9:57.080	2:08.569
Theoretical	$O(N^2)$	$O(N^{1.14})$	$O(smN)$	$O(N)$

Table (1)





Conclusion

Considering all the data and analysis, we can see that all of the algorithms tested have certain strengths and weaknesses. With this, we can find a use case for all the algorithms. With its focus on solving the crowding-out problem, t-SNE is the best choice for reducing data-sets where a more uniformly distributed embedding is desired. Furthermore, with the most stable memory usage among algorithms, it is well applied to use cases where predictable memory usage is favorable. However, we recommend that the original serial version of t-SNE is not used, as it fails to scale to large data-sets, with its high memory consumption and high computational complexity. Therefore it is recommended that if it is used, that one of its more optimized versions be used. Being based upon t-SNE with the same KNN tree construction, LargeVis shares with it certain characteristics, such as it more stable memory usage. However, LargeVis is significantly less computationally complex than t-SNE, allowing it to scale to data-sets with greater data points. With the focus of LargeVis being to improve runtime as compared to t-SNE, it certainly delivers. Therefore, we believe that LargeVis should be used for applications where stable memory usage, high scalability, and the preservation of the data's local structure are desired. Moving on to UMAP, we are given faster runtime than both t-SNE and LargeVis, but at the cost of less stable memory usage. Providing similar embedding quality as compare to LargeVis, UMAP should be used as an alternative for LargeVis use cases when the stability of memory is not a factor, as it provides similar results in nearly half the time. Finally, we are left with the fastest and most memory efficient of all algorithms tested, that being TriMap. With its significantly lower runtimes and memory usage TriMap certainly outclasses all other tested algorithms in all tests. However, TriMap is somewhat limited by its design specifications, as it is primarily geared towards maintaining the global structure of data, rather than local structure. With relatively unstable memory usage, the predictability of TriMap is less than that of algorithms like LargeVis and t-SNE. Therefore, we believe that the best use cases for TriMap are whenever maintaining the global structure of data is the goal or the size of the data-set is to great for any algorithm to realistically process.

Future Works

The extent of this survey is limited in certain aspects and can be improved in a few ways. One of these aspects is the scale of testing. Having only tested the algorithms on the MNIST Digits and Fashion MNIST datasets, we would like to include more data sets with greater variety and explore how various factors, such as original dimensionality and the number of data points, affect algorithm metrics. Some future aspects for analysis include the effect of algorithms' hyper-parameter settings on the quality and metrics of embeddings and the improvements and drawbacks of algorithm reimplementations, such as those of t-SNE.

Link to Paper