A Fast Approximation to Multidimensional Scaling

Tynia Yang[1], Jinze Liu[1], Leonard McMillan[1], and Wei Wang[1]

University of Chapel Hill at North Carolina, Chapel Hill NC 27599, USA
{tynia, liuj, mcmillan, weiwang}@cs.unc.ed

Abstract. We present an approximation algorithm for Multidimensional Scaling (MDS) for use with large datasets and interactive applications. MDS describes a class of dimensionality reduction techniques that takes a dissimilarity matrix as input. It is often used as a tool for understanding relative measurements when absolute measurements are not available. MDS is also used for visualizing high-dimensional datasets. At the core of MDS is an eigendecomposition on an $n \times n$ symmetric matrix. For large n, this eigendecomposition becomes unwieldy. Our method employs a divide-and-conquer approach, dividing the matrix into submatrices of reasonable size to perform MDS, and then stitching the subproblem solutions back together for a complete solution for the $n \times n$ matrix. It requires $\Theta(n \lg n)$ steps and is easily parallelized.

1 Introduction

The objective of this work is to describe a fast approximation to classical multidimensional scaling (MDS) [11] that enables it to be applied to large datasets at interactive speeds. MDS describes a class of dimensionality reduction techniques that operate on pairwise dissimilarities between points and generates a low-dimensional embedding. In this paper, we focus on classical metric MDS, which assumes a Euclidean distance as the dissimilarity measure. MDS is commonly used as a subroutine in feature selection methods [9], face recognition applications [1], and non-linear dimensionality reduction tools such as Isomap [10]. MDS is also an effective tool for visualizing and exploring the structure of high-dimensional datasets. However, the applications of MDS are limited by its poor scalability with regard to the dataset's size.

Several approximations to MDS have been developed to support large datasets. Before we present these other MDS approximation approaches in the next section, and compare them to our approach, we provide a brief overview of classical MDS.

1.1 Classical Multidimensional Scaling

Multidimensional scaling (MDS) is a well-known statistical method for mapping pairwise relationships to coordinates. The coordinates that MDS generates are

an optimal linear fit to the given dissimilarities between points, in a least squares sense, assuming the distance used is metric. An MDS solution is unique down to a rigid-body transformation, with a possible reflection. MDS takes as input an $n \times n$ matrix $\mathbf{D}$ containing pairwise dissimilarities between all n data objects. A valid dissimilarity matrix must satisfy both of the following constraints: (i) self-similarity ($d_{ii} = 0$) and (ii) symmetry ($d_{ij} = d_{ji}$). A dissimilarity matrix is metric if (i) and (ii) hold and the dissimilarities obey the triangle inequality: $d_{ij} \leq d_{ik} + d_{ki}$ for all k.

The objective of MDS is to find coordinates for each point that preserve the given pairwise dissimilarities as faithfully as possible. However, MDS is also useful when coordinates are given for high-dimensional datasets. Since MDS only takes a scalar dissimilarity matrix as input, its performance is independent of the dataset's dimensionality. Its performance depends only on the number of objects. In the case where the dissimilarity is Euclidean distance, MDS gives a solution that is identical to principal component analysis (PCA).

There are two stages in computing classical MDS. The first is to convert the input matrix $\mathbf{D}$ into a matrix of dot products, or a Gram matrix $\mathbf{B}$. This is done by multiplying $\mathbf{D}^2$ on both sides with a "centering matrix" $\mathbf{H}$, which subtracts out the row and column average of each entry and adds back the overall matrix average.

$$\begin{aligned} \mathbf{B} &= \frac{-\mathbf{H}\mathbf{D}^2\mathbf{H}}{2} \\ h_{ij} &= \delta_{ij} - \frac{1}{n} \end{aligned} \tag{1}$$

The second stage is the bottleneck in MDS. Since $\mathbf{B}$ is symmetric, it can be eigendecomposed into $\mathbf{U}\mathbf{S}\mathbf{U}^T$, where $\mathbf{U}$ is a matrix of eigenvectors and $\mathbf{S}$ is a diagonal matrix containing the corresponding eigenvalues. MDS derives its lower-dimensional coordinates by taking successive columns from $\mathbf{U}\sqrt{\mathbf{S}}$. A complete eigendecomposition of $\mathbf{B}$ using QR decomposition takes $\mathcal{O}(n^3)$ time, resulting in an $\mathcal{O}(n^3)$ time for MDS.

2 Related Work

Several approximations to classical MDS have been proposed to address its poor scalability. One class of algorithms is based on a spring-mass model [2][6]. These methods calculate lower-dimensional coordinates by iteratively minimizing a cost, or stress, function that is proportional to the distance between the current coordinates and the given dissimilarities. Chalmers [2] has developed a fast $\Theta(n^2)$ approximation to the spring-based approach that considers a set of *near* neighbors to each point and a second set of *far* points that are selected at random on each iteration. Morrison et al. [6] introduced a sampling-based variant to Chalmers' algorithm that achieves $\Theta(n \lg n)$ performance, but is limited to embeddings in two-dimensions. Williams et al. have suggested an improvement to Chalmers' approach, and added user control to speed up performance [13]. A disadvantage of spring-based models, in general, is that they are subject to local

minima, and that they require an *a priori* assumption of the dataset's underlying dimensionality.

FastMap [5], MetricMap [12], and Landmark MDS (LMDS) [4] approximate classical MDS by solving MDS for a subset of the data and fit the remainder to the solution. Platt, in [8], shows how all three algorithms belong to a class of methods called Nyström algorithms, which approximates the solution to the eigenproblem of a large matrix and concludes that LMDS is the fastest and most accurate of the three.

Landmark MDS designates a set of m points as "landmarks", where $m << n$. It then extracts the m rows from $\mathbf{D}$ that contain the distances from the m landmarks to every other point, resulting in a submatrix $\mathbf{D}_{m \times n}$ of size $m \times n$. LMDS then applies MDS to the $m \times m$ matrix $\mathbf{D}_{landmarks}$, which contains the pairwise distances between just the landmarks. The result is a set of coordinates in $\mathbb{R}^{d_{low}}$ for just the landmarks. The algorithm then uses a distance-based triangulation, taking the distances from $\mathbf{D}_{m \times n}$, to determine coordinates for the remaining points. LMDS runs in $\mathcal{O}(Cmn + d_{low}mn + m^3)$ where C is the cost of computing and accessing each entry of $\mathbf{D}_{m \times n}$. In practice, m is $\Theta(\sqrt{n})$ to get an acceptable approximation.

FastMap [5] approximates classical MDS by constructing a d_{low}-dimensional embedding one dimension at a time. It iteratively selects the two farthest points in the data to be the axis, and uses the distances from the n-2 remaining points to the two chosen points to compute the embedding coordinates. As pointed out in [4], FastMap essentially is an iterated form of LMDS in the simplest case of two landmarks.

In this paper, we propose an algorithm that solves MDS for all of the input, but in a piecewise manner. We then take a sampling-based approach to fit the pieces back together.

3 Sampling-based FastMDS

Algorithm Overview. Our FastMDS approach is based on the observation that a submatrix along the diagonal of a dissimilarity matrix is itself a dissimilarity matrix. Instead of running MDS on the full $n \times n$ matrix $\mathbf{D}$, we partition $\mathbf{D}$ along the diagonal into p submatrices $D_1, D_2, \ldots D_p$, each of size $\frac{n}{p} \times \frac{n}{p}$. Throughout this paper, we will refer to a submatrix as $\mathbf{D}_i$, where $1 \leq i \leq p$.

We then compute the MDS solution for each submatrix $\mathbf{D}_i$. We stitch these individual MDS solutions together by sampling s points from each submatrix $\mathbf{D}_i$ and putting them into an alignment matrix $\mathbf{M}_{align}$ of size $sp \times sp$. In principle, s should be at least 1 + the estimated dimensionality of the dataset. In practice, we oversample by a factor of 2 or more, to ensure that we capture the data's inherent dimensionality. We run MDS on $\mathbf{M}_{align}$ to get d_{low}-dimensional coordinates for the sampled points. We now have two MDS solutions for each of the sampled points; one from performing MDS on $\mathbf{D}_i$ and one from performing MDS on $\mathbf{M}_{align}$. The next step is to compute an affine mapping $\mathbf{A}_i$ between these two

sets of solutions to line them up in a common coordinate system. This is a linear least squares problem:

$$\mathbf{A}_i dMDS_i = mMDS_i \tag{2}$$

where $dMDS_i$ is the MDS solution for the sample points from $\mathbf{D}_i$ and $mMDS_i$ is the solution from $\mathbf{M}_{align}$ that corresponds to sampled points from $\mathbf{D}_i$. Solving for $\mathbf{A}_i$ gives us a mapping between $\mathbf{D}_i$ and $\mathbf{M}_{align}$, which we apply to the rest of $\mathbf{D}_i$ to get d_{low}-dimensional coordinates for all $\frac{n}{p}$ points. In solving for $\mathbf{A}_i$, we minimize the least squares error between our approximation of MDS and the real MDS solution.

We apply this process recursively, until the size of $\mathbf{D}_i$ is optimal to run MDS on. We find this stopping condition as follows. Let $l \times l$ be the largest matrix that allows MDS to be executed efficiently. There are two issues that impact the performance of FastMDS on an $n \times n$ matrix. (i) the size of $\mathbf{D}_i$ after subdivision and (ii) p, the number of submatrices that we stitch together at each conquer step. Ideally, the size of each submatrix after division should be as large as possible without exceeding $l \times l$. By the same token, the size of the alignment matrix $\mathbf{M}_{align}$ should also be bounded by $l \times l$. The number of submatrices to be stitched together, p, should be the largest number such that $sp \leq l$.

In our examples, we define our peak error ε to be the maximum distance between the FastMDS and the real MDS solutions for two corresponding points, after aligning the two solutions, as shown in (3). Because MDS returns a unique solution only down to a translation, rotation, and possible reflection, we first need to align the FastMDS solution with the MDS solution before evaluating ε. We compute an affine mapping $\mathbf{A}$ between $\mathbf{M} = (m_1 \dots m_n)^T$ and $\mathbf{F} = (f_1 \dots f_n)^T$. We then apply $\mathbf{A}$ to $\mathbf{F}$, in order to line the two solutions up, and set ε equal to the maximum distance between two corresponding points. $\mathbf{M}$ contains the d_{low}-dimensional coordinates from MDS and $\mathbf{F}$ contains the d_{low}-dimensional coordinates from FastMDS.

Assuming $\mathbf{D}$ can be embedded in d_{low} dimensions, regular MDS guarantees an optimal embedding in $\mathbb{R}^{d_{low}}$ space. So there exists a d_{low}-dimensional coordinate for each point represented in $\mathbf{D}$. We use a matrix of size sp ($\mathbf{M}_{align}$) to find the d_{low}-dimensional coordinates for a subset of points, which we then use to align the remaining points. Assuming we sampled enough points from each submatrix $\mathbf{D}_i$, the subset of points in $\mathbf{M}_{align}$ should be enough to capture the structure in the original point set.

$$\varepsilon \equiv \max_{i=1}^{n} \|m_i - \mathbf{A} f_i\| \tag{3}$$

Computational Complexity. We can characterize the steps in our FastMDS algorithm with the following recurrence relation:

$$\mathcal{T}(n) = p\mathcal{T}(\frac{n}{p}) + \mathcal{MDS}(sp) + p\mathcal{AFF}(s) + p\mathcal{Z}(\frac{n}{p}) \tag{4}$$

where p is the number of partitions, and s is the number of samples per submatrix. $\mathcal{MDS}(sp)$ is the cost of running MDS on $\mathbf{M}_{align}$. Because both s and p are constant with respect to n, we treat this as a constant-time operation. $\mathcal{AFF}(s)$ is the process of computing the affine map $\mathbf{A}_i$ between the s sample points in $\mathbf{D}_i$ and $\mathbf{M}_{align}$. Solving a linear least squares problem for an $s \times s$ matrix involves calculating a pseudoinverse and takes $\mathcal{O}(s^2)$. Because s is invariant with respect to n, and is always small compared to n, we treat $\mathcal{AFF}(s)$ as a constant-time operation as well. $\mathcal{Z}(\frac{n}{p})$ is a function that applies $\mathbf{A}_i$ to a matrix of size $\frac{n}{p} \times \frac{n}{p}$. This amounts to a simple matrix multiplication, which is a linear-time operation and takes $\mathcal{O}(n)$ time. So we can simplify (4) into (5) and solve to get (6):

$$\mathcal{T}(n) = p\mathcal{T}(\frac{n}{p}) + \mathcal{Z}(n) \tag{5}$$

$$\mathcal{T}(n) = \Theta(n \lg n) \tag{6}$$

4 Examples

To demonstrate FastMDS and how it can be applied to real data, we ran it on images taken from the MNIST database of handwritten digits [7], which contains a training set of 60,000 examples and a test set of 10,000 examples of handwritten digits from 0-9. All images are 28×28 in size and are in grayscale. We ran our algorithm on a set of 974 "8"s from the test set (Figure 1) and projected the data down to two dimensions.

We can see in Figure 1 that the "8"s separate into round, full shapes in the lower left corner of the figure, and narrow, spindly shapes towards the upper right corner. They also vary in the slant direction, with the top left corner showing a strong slant towards the right, but decreasing the amount of slant as we move towards the non-slanted "8"s in the lower left corner of the figure.

We also demonstrate FastMDS's accuracy in approximating real MDS. We ran both MDS and FastMDS on random sets of 10-dimensional points, ranging in size from 1000 to 5000 points. The results are displayed below in Table 4. For the first four experiments, as we double the input size, computation time for MDS increases by a factor of eight. In contrast, FastMDS roughly triples its computation time, but returns a set of coordinates that is no more than ε away from MDS's. For 5000 points, MDS runs out of memory and crashes.

To illustrate the similarity between solutions, we plot the points from the first experiment (1000 points) in Figure 2. The MDS solution is displayed as blue squares ($\square$), and FastMDS is displayed as red Xs (x). As shown in the figure, the two sets of points overlap, with $\varepsilon = 1.26$x10-7.

5 Conclusion and Future Work

We have developed a fast approximation to MDS that is well suited for large datasets. Our method has the same asymptotic performance as the best previous

Table 1. Timing results for MDS vs. FastMDS on random points. After 4000 points, MDS runs out of memory and we are unable to compute a solution

No. of points	**MDS run time (sec)**	**FastMDS run time (sec)**	ε
1000	6.28	0.45	1.2665e-007
2000	47.67	1.125	1.5944e-007
3000	158.62	1.875	1.9542e-007
4000	377.50	3.141	1.8128e-007
5000	n/a	4.515	n/a

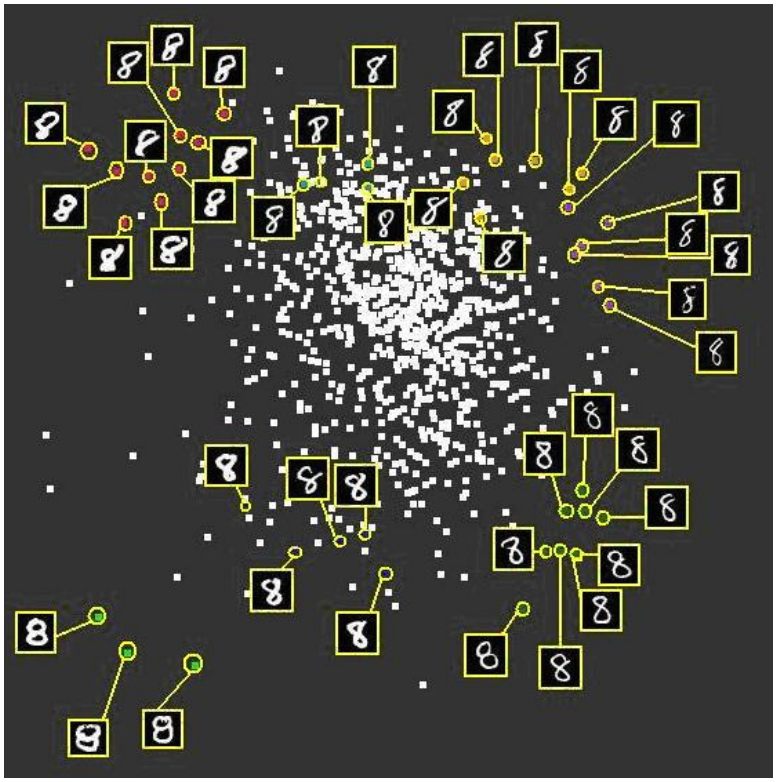

Fig. 1. FastMDS plot of 974 handwritten "8"s from the MNIST database, projected onto 2 dimensions. We can see noticeable trends in the patterns, from round, non-slanted "8"s in the bottom left corner to the thinner, more stretched-out "8"s in the top right corner. Most of the "8"s on the top of the figure are slanted towards the right, while they straighten out towards the bottom of the figure.

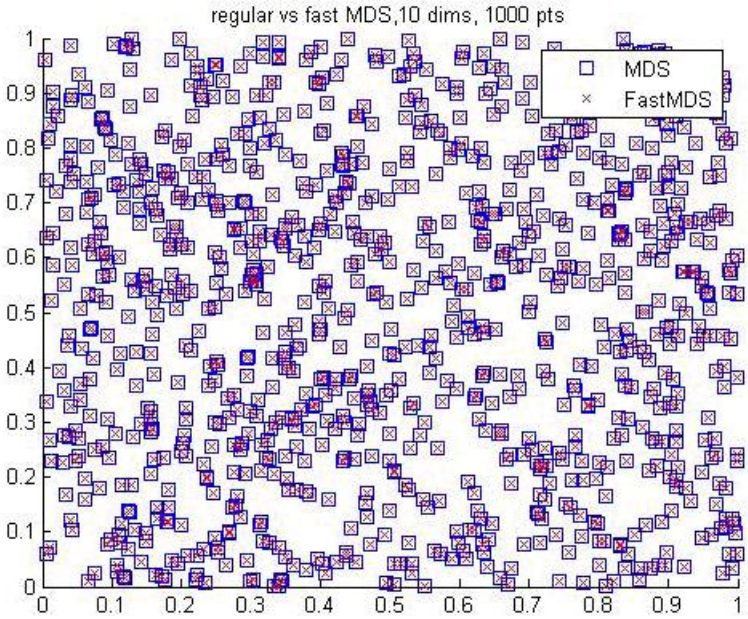

Fig. 2. 2-dimensional coordinates for 1,000 10-dimensional points. Our algorithm finds a solution with ε = 1.2665x10-7.

MDS approximation algorithm [6], and it allows for arbitrary low-dimensional solutions as long as $d_{low} << s$ and $d_{low} << \frac{n}{p}$. We have used our method in interactive applications and on large data sets. In practice, our method's solutions are in close agreement with classical MDS for low-dimensional embeddings.

There is still an element of art in the optimal selection of p, or more accurately $\frac{n}{p}$. We have observed, and attribute to cache effects, non-monotonic behaviors as the value of p is varied. This leads to sweet spots (i.e. certain submatrix sizes where the small MDS solutions are particularly fast). We suspect that these effects are likely to be implementation and platform dependent. Nonetheless, it is common for some numeric algorithms (FFTs in particular) to search for an optimal subproblem size as a precursor to solving large problems. We expect a similar idea could be used for FastMDS.

Finally, we would like to derive a bound on the expected errors of our FastMDS approximation. There is some hope for this based on the relationship of the MDS mapping from a dissimilarity to a Gram matrix. The Gram matrix is constructed by subtracting the appropriate row and column average from and adding the matrix average to each element of the dissimilarity matrix. If the submatrix dissimilarity matrices of FastMDS can maintain the same statistics as the full dissimilarity matrix, we expect identical solutions. In practice we randomize the points (or rows and columns of the given dissimilarity matrix) in an effort to achieve similar row, column, and matrix averages for all submatrices

References

1. Bronstein, A., Bronstein, M., Kimmel, R. "Expression-invariant 3D face recognition," in *Proc. Audio and Video-based Biometric Person Authentication*, pp. 62-69, 2003.
2. Chalmers, M. "A linear iteration time layout algorithm for visualizing high dimensional data," *Proc. IEEE Visualization*, pp. 127-132. 1996.
3. V. de Silva and J.B. Tenenbaum. "Global versus local methods in nonlinear dimensionality reduction," *Advances in Neural Information Processing Systems 15* S. Becker, S. Thrun, and K. Obermayer (eds). Cambridge, MIT Press, 705-712, 2002.
4. V. de Silva and J.B. Tenenbaum. "Sparse multidimensional scaling using landmark points" June 2004.
5. Faloutsos, C., Lin, K. "FastMap: a fast algorithm for indexing, data-mining, and visualization," *Proc. of ACM SIGMOD*, pp. 163-174, 1995.
6. Morrison, A., Ross, G., Chalmers, M. "Fast Multidimensional Scaling through Sampling, Springs, and Interpolation," *Information Visualization* 2(1), pp. 68-77, March 2003.
7. The MNIST Database of handwritten digits. http://yann.lecun.com/exdb/mnist/
8. Platt, J.C. "FastMap, MetricMap, and Landmark MDS are all Nyström Algorithms," 10^{th} *International Workshop on Artificial Intelligence and Statistics*, pp. 261-268, 2005
9. Shashua, A. Wolf, L. "Kernel Feature Selection with Side Data using a Spectral Approach," *Proc. of the European Conference on Computer Vision (ECCV)*, May 2004, Prague, Czech Republic
10. Tenenbaum, J.B., de Silva, V., Langford, J.C. "A global framework for nonlinear dimensionality reduction," *Science* 290 (5500), 2319-2323.
11. Torgerson, W.S. "Multidimensional Scaling: Theory and Method," *Psychometrika*, vol 17, pp. 401-419, 1952.
12. Wang, J.T-L., Wang, X., Lin, K-I., Shasha, D., Shapiro, B.A., Zhang, K. "Evaluating a class of distance-mapping algorithms for data mining and clustering," *Proc of ACM KDD*, pp. 307-311, 1999.
13. Williams, M., Munzner, T. "Steerable, Progressive Multidimensional Scaling," *INFOVIS* 2004, pp. 57-64, October 2004