FastMap, MetricMap, and Landmark MDS are all Nyström Algorithms

John C. Platt
Microsoft Research
1 Microsoft Way
jplatt@microsoft.com

Abstract

This paper unifies the mathematical foundation of three multidimensional scaling algorithms: FastMap, MetricMap, and Landmark MDS (LMDS). All three algorithms are based on the Nyström approximation of the eigenvectors and eigenvalues of a matrix. LMDS is applies the basic Nyström approximation, while FastMap and MetricMap use generalizations of Nyström, including deflation and using more points to establish an embedding. Empirical experiments on the Reuters and Corel Image Features data sets show that the basic Nyström approximation outperforms these generalizations: LMDS is more accurate than FastMap and MetricMap with roughly the same computation and can become even more accurate if allowed to be slower.

1 INTRODUCTION

Multidimensional Scaling (MDS) [4] is an important method for visualizing and processing high-dimensional or graphical data. MDS takes as input a distance matrix between items. It produces a coordinate vector for each item in a Euclidean space whose dimension is user-specifiable. This process is known as *embedding*.

MDS is applicable to two different tasks: 1) dimensionality reduction, which measures a set of distances between items, then applies MDS to the resulting (perhaps sparse) distance matrix; and 2) converting graphs to vectors, where a data set is expressed as a set of similarity relationships between items that is then converted into a simple table of vectors.

The original MDS algorithms from the 1960s [4] are not appropriate for large scale applications because they require an entire $N \times N$ distance matrix to be stored in memory and may have $O(N^3)$ complexity. However, in the last 10 years, several scalable MDS algorithms have been proposed. For example, Falutsos and Lin [7] proposed FastMap, which is an MDS method that determines one coordinate at a time by examining a constant number of rows of the distance matrix. Wang, et. al [12] proposed an improvement on FastMap, called MetricMap, which attempts to do the entire projection at once. de Silva and Tennenbaum [5] proposed Landmark MDS (LMDS) as another attempt at scalable MDS.

FastMap, MetricMap, and LMDS are all classical MDS algorithms that start with a distance matrix assumed to have been computed from points in a low-dimensional space. They then (approximately) minimize a quadratic cost function between the resulting embedding coordinates and the original (hidden) coordinates that created the distance matrix. Another set of MDS algorithms are based on spring models [3, 14]. These spring models minimize a cost function that is non-quadratic in the coordinates: the squared difference between embedded and observed distances between items. Recent work has also accelerated these spring models, but they are prone to local minima [3]. Spring models are not considered in this paper.

The proliferation of MDS algorithms may lead to frustration amongst practitioners because it is unclear how the algorithms relate to one another or how they compare against each other. This paper attempts to clarify the situation in two ways. First, this paper shows that FastMap, MetricMap, and LMDS are all algorithms based on the Nyström approximation of the eigenvectors of a large matrix [1, 13], based only on a rectangular sub-matrix of the large matrix. LMDS uses the basic Nyström approximation, FastMap uses Nyström to find one eigenvector at a time, and MetricMap uses an irregular sub-matrix to find the eigenvectors. The paper presents empirical comparisons between FastMap, MetricMap, and LMDS, to determine which variation

of Nyström is optimal.

2 MATHEMATICAL BACKGROUND

This section presents a step-by-step introduction to Classical MDS and the Nyström eigenvector approximation. The LMDS algorithm is then derived based on the combination of the two, as first described in [2].

2.1 REVIEW OF CLASSICAL MDS

FastMap, MetricMap, and LMDS are all multi-dimensional scaling (MDS) algorithms [4] that map a matrix of dissimilarities $\mathbf{D}$ between N items to a k-dimensional coordinate vector for each item ($\vec{x}_i$). This paper's derivation for all three of these algorithms starts with metric MDS, which assumes that the entries in the dissimilarities matrix are Euclidean distances. These three algorithms then utilize classical MDS, which chooses the k-dimensional coordinates to minimize the squared difference between the distances in the embedded and the original spaces.

Classical MDS proceeds in two steps. First, the distance matrix $\mathbf{D}$ undergoes "double-centering" to convert it from a distance matrix to a new matrix $\mathbf{K}$:

$$K_{ij} = -\frac{1}{2}\left(D_{ij}^2 - e_j \sum_i c_i D_{ij}^2 - e_i \sum_j c_j D_{ij}^2 + \sum_{i,j} c_i c_j D_{ij}^2\right), \quad (1)$$

where $\sum_i c_i = 1$ and e_i is the vector of all ones. If the original distance matrix $\mathbf{D}$ is a Euclidean distance matrix in d-dimensional space, then $\mathbf{K}$ is a matrix of dot products between coordinate vectors in that same space [11]. This is also known as a Gram or kernel matrix. If the original distance matrix is Euclidean, then $\mathbf{K}$ is symmetric and positive semi-definite. The parameters c_i in (1) determine the origin of the coordinate vectors (which is not constrained by the distance matrix $\mathbf{D}$).

The second step of classical MDS is to extract the coordinate vectors from the kernel matrix $\mathbf{K}$ through eigenvector decomposition. If the matrix $\mathbf{D}$ is symmetric, then $\mathbf{K}$ is symmetric and can be decomposed into

$$\mathbf{K} = \mathbf{Q}\mathbf{\Lambda}\mathbf{Q}^T \quad (2)$$

where $\mathbf{Q}$ is a matrix whose columns are orthonormal eigenvectors and $\mathbf{\Lambda}$ is a diagonal matrix of eigenvalues. The k-dimensional coordinate vectors that would give rise to the kernel matrix $\mathbf{K}$ are the scaled rows of $\mathbf{Q}$. In order to minimize the difference between the embedded and original distances with a fixed number of dimensions k, the eigenvectors with the top k eigenvalues are retained. Assuming that the eigenvalues are ordered by decreasing eigenvalue, the jth component of point i's coordinate vector is:

$$x_{ij} = \sqrt{\lambda_j} Q_{ij}, \quad (3)$$

where λ_j is the jth eigenvalue and here the index j runs only from 1 to k, rather than to N (the size of $\mathbf{K}$).

2.2 THE NYSTRÖM APPROXIMATION

There are numerous ways of performing the decomposition (2). If only the top k eigenvectors are needed, then orthogonal iteration or Lanczos iteration can be applied [8].

However, the three MDS algorithms that are the subject of this paper use an approximation method from physics, called the Nyström approximation [1, 13]. To use Nyström, first choose m items in the distance and kernel matrix at random. Without loss of generality, permute the m items to be the first rows and columns of these matrices. The $\mathbf{K}$ and $\mathbf{D}$ can then be partitioned into submatrices:

$$\mathbf{K} = \begin{array}{|c|c|}\hline \mathbf{A} & \mathbf{B} \\ \hline \mathbf{B}^T & \mathbf{C} \\ \hline \end{array}, \quad \mathbf{D} = \begin{array}{|c|c|}\hline \mathbf{E} & \mathbf{F} \\ \hline \mathbf{F}^T & \mathbf{G} \\ \hline \end{array}, \quad (4)$$

where $\mathbf{A}$ and $\mathbf{E}$ have dimension $m \times m$; $\mathbf{B}$ and $\mathbf{F}$ have dimension $m \times (N-m)$; and $\mathbf{C}$ and $\mathbf{G}$ have dimension $(N-m) \times (N-m)$.

The Nyström approximation permits the computation of the coordinates $\vec{x}_i$ using only the information in matrices $\mathbf{A}$ and $\mathbf{B}$. Nyström assumes that $\mathbf{K}$ is positive semi-definite, and hence a Gram matrix. Thus, $\mathbf{K}$ should be expressible in terms of dot products between columns of matrices $\mathbf{X}$ and $\mathbf{Y}$ [1]:

$$\mathbf{K} = \begin{bmatrix} \mathbf{X}^T\mathbf{X} & \mathbf{X}^T\mathbf{Y} \\ \mathbf{Y}^T\mathbf{X} & \mathbf{Y}^T\mathbf{Y} \end{bmatrix}. \quad (5)$$

Identifying the submatrices in (5) with those in $\mathbf{K}$ in (4) yields

$$\begin{aligned} \mathbf{A} &= \mathbf{X}^T\mathbf{X}, \\ \mathbf{B} &= \mathbf{X}^T\mathbf{Y}. \end{aligned} \quad (6)$$

The first equation in (6) is standard in classical MDS: the solution for $\mathbf{X}$ is to eigendecompose $\mathbf{A}$:

$$\mathbf{A} = \mathbf{U}\mathbf{\Gamma}\mathbf{U}^T, \quad (7)$$

and then assign the coordinates

$$\mathbf{X} = \mathbf{\Gamma}_{[k]}^{1/2}\mathbf{U}_{[k]}^T, \quad (8)$$

where the subscript $[k]$ indicates the submatrices corresponding to the eigenvectors with the k largest positive eigenvalues. The coordinates corresponding to $\mathbf{B}$ can be derived by solving the linear system:

$$\mathbf{Y} = \mathbf{X}^{-T}\mathbf{B} = \mathbf{\Gamma}_{[k]}^{-1/2}\mathbf{U}_{[k]}^{T}\mathbf{B}. \tag{9}$$

Combining equations (8) and (9) together and writing the coordinate as rows in a matrix results in

$$x_{ij} = \begin{cases} \sqrt{\gamma_j}U_{ij} & \text{if } i \le m; \\ \sum_p B_{pi}U_{pj}/\sqrt{\gamma_j} & \text{otherwise,} \end{cases} \tag{10}$$

where U_{ij} is the ith component of the jth eigenvector of $\mathbf{A}$ and γ_j is the jth eigenvalue of $\mathbf{A}$. As in (3) the j index only runs from 1 to k, in order to make a k-dimensional embedding.

The Nyström approximation can be understood by plugging (8) and (9) back into (6). Nyström approximates the full matrix $\mathbf{K}$ by

$$\tilde{\mathbf{K}} = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{B}^T\mathbf{A}^{-1}\mathbf{B} \end{bmatrix}. \tag{11}$$

This approximation is exact when $\mathbf{K}$ is of rank m or less. The quality of the approximation is proportional to $||\mathbf{C} - \mathbf{B}^T\mathbf{A}^{-1}\mathbf{B}||$.

To turn Nyström into an MDS method, matrices $\mathbf{A}$ and $\mathbf{B}$ must be derived only from submatrices $\mathbf{E}$ and $\mathbf{F}$ (from $\mathbf{D}$). This can be done by choosing the centering coefficients in equation (1) to be

$$c_i = \begin{cases} 1/m & \text{if } i \le m; \\ 0 & \text{otherwise,} \end{cases} \tag{12}$$

which yields the centering formulas

$$\begin{aligned} A_{ij} \;=\; & -\frac{1}{2}\left(E_{ij}^2 - e_i\frac{1}{m}\sum_p E_{pj}^2 \right. \\ & \left. -e_j\frac{1}{m}\sum_q E_{iq}^2 + \frac{1}{m^2}\sum_{p,q} E_{pq}^2 \right) \end{aligned} \tag{13}$$

$$B_{ij} = -\frac{1}{2}\left(F_{ij}^2 - e_i\frac{1}{m}\sum_p F_{qj}^2 - e_j\frac{1}{m}\sum_p E_{ip}^2 \right), \tag{14}$$

where the constant centering term in (14) is dropped, because it introduces an irrelevant shift of origin.

Note that Nyström can also be used to approximately solve spectral clustering problems [1], by using a normalized graph Laplacian, instead of a centering matrix, to transform a data matrix into a kernel matrix.

2.3 LMDS

The combination of equations (13) and (14), followed by (7), then (10) is almost the LMDS algorithm [5]. LMDS is so named because Classical MDS is first applied to a subset of the points (in $\mathbf{A}$), called "landmarks."

The only difference between the algorithm derived here and LMDS is the centering formula (14). In [5], the centering formula

$$B_{ij} = -\frac{1}{2}\left(F_{ij}^2 - e_i\frac{1}{m}\sum_p E_{ip}^2 \right) \tag{15}$$

is used. This simplification does not affect the results, because it adds a vector proportional to e_i to every column of $\mathbf{B}$. Equation (9) shows that the columns of $\mathbf{B}$ are only involved in dot products with the eigenvectors of $\mathbf{A}$. It is easy to show that e_i is an eigenvector of $\mathbf{A}$ with eigenvalue zero. Therefore, all other eigenvectors must be orthogonal to e_i. Therefore, adding a vector proportional to e_i to the columns of $\mathbf{B}$ does not change the embedding results.

The computational complexity of LMDS is $O(Nmk + m^3)$. For large N, the computation time is dominated by computing the input distances from raw input data and projecting the distances in (10).

3 UNIFICATION OF THE THREE ALGORITHMS

While LMDS, FastMap, and MetricMap were all proposed independently and appear to be distinct algorithms, we will see in the following section that they are all, in fact, applications of the Nyström approximation.

3.1 FASTMAP

Consider LMDS for $k = 1$ and $m = 2$. In this case, we are trying to find a one-dimensional embedding using two landmark points. If the distance between the two landmarks is d_{12} and the distance from each landmark to all other points is d_{1i} or d_{2i}, then LMDS produces:

$$\mathbf{A} = \frac{1}{4}\begin{bmatrix} d_{12}^2 & -d_{12}^2 \\ -d_{12}^2 & d_{12}^2 \end{bmatrix}. \tag{16}$$

The largest eigenvalue for $\mathbf{A}$ is $d_{12}^2/2$, and its corresponding eigenvector is $[1\ -1]^T/\sqrt{2}$. After centering,

$$\mathbf{B} = \frac{1}{2}\begin{bmatrix} d_{1i}^2 - d_{12}^2/2 \\ d_{2i}^2 - d_{12}^2/2 \end{bmatrix}. \tag{17}$$

Using this in equation (10) yields a coordinate

$$x_i = \frac{\frac{-1}{2\sqrt{2}}\left(d_{1i}^2 - d_{2i}^2\right)}{d_{12}/\sqrt{2}} = \frac{d_{2i}^2 - d_{1i}^2}{2d_{12}}, \tag{18}$$

which is exactly one iteration of FastMap [7], except for an unimportant shift in the origin.

The $k = 1, m = 2$ case assumes that the principal eigenvector of the data lies on the line connecting points 1 and 2. This can be a poor estimate of this eigenvector. So, FastMap proposes a heuristic that scans a number of rows of the distance matrix $\mathbf{D}$, looking for two points that are far away from one another, assuming that the principal eigenvector is more likely to lie along the line connecting two points that are distant. Notice that if FastMap is going to generate m rows of a distance matrix per iteration, those rows can easily be added to the Nyström approximation to increase the accuracy of the eigenvalue and eigenvector estimate, instead of being used in a heuristic.

One iteration of FastMap generates only one embedding dimension at a time. Because FastMap is estimating the leading eigenvectors of a kernel matrix, it is legitimate to perform *deflation*. That is, each subsequent call to FastMap operates in a subspace that is orthogonal to previous dimensions, with previous eigenvectors projected away. Typically, deflation is performed by deducting a rank-one matrix from the kernel matrix. However, we cannot manipulate the entire kernel matrix. Therefore, FastMap computes the distances in the deflated space by deducting the squared distance between embedded coordinates from the original squared distance:

$$D_{ij}^2 = D_{i,j,\text{original}}^2 - \sum_n (x_{in} - x_{jn})^2. \qquad (19)$$

This is legitimate, because the deflated space is orthogonal to any embedded dimensions. Notice that this deflation can result in negative squared distances when the original distance matrix is not Euclidean. LMDS has a similar problem with negative eigenvalues of $\mathbf{A}$. FastMap produced negative distances for the Corel Features dataset in Section 4, while on the same data set, LMDS did not produce negative eigenvalues. These negative eigenvalues can be compensated by adding a small amount to the diagonal of $\mathbf{A}$.

The computational complexity of FastMap is $O(Nk^2)$, because the computation is dominated by k deflation iterations, each operating on N data samples, each of which takes $O(k)$ operations, due to (19).

3.2 METRICMAP

MetricMap [12] can be understood as a Nyström approximation by revisiting Landmark MDS. In LMDS, m rows of the distance matrix $\mathbf{D}$ are used to compute coordinates for every item. This rectangular slice is used both to find the coordinates of the landmarks (the m items corresponding to the m rows) at equation (8) and the remainder of the rows (equation (9)).

MetricMap uses a generalization of the Nyström approximation with different sized submatrices $\mathbf{A}$ and $\mathbf{B}$ in (8) and (9). MetricMap prescribes that $\mathbf{A}$ have dimension $2k \times 2k$, while $\mathbf{B}$ have dimension $k \times k$ [12]. Because the dimensions of $\mathbf{A}$ and $\mathbf{B}$ no longer match, the solution to the linear system in (9) is no longer correct. Instead, MetricMap derives embedding coordinates $\mathbf{X}$ from the $2k \times 2k$ matrix $\mathbf{A}$, using classical MDS from the k largest (in absolute value) eigenvalues of $\mathbf{A}$. Only k landmarks from the $2k$ embedded points are used:

$$\hat{\mathbf{X}} = \mathbf{\Gamma}_{[k,k]}^{1/2} \mathbf{U}_{[k,k]}, \qquad (20)$$

where the subscript $[k, k]$ indicates the sub-matrices indexed by the k largest eigenvalues and the k selected landmarks.

The k landmark rows of $\mathbf{A}$ that are selected by this procedure are also the rows used to generate $\mathbf{B}$. Thus, equation (9) becomes

$$\mathbf{Y} = \hat{\mathbf{X}}^{-T} \mathbf{B} = \mathbf{\Gamma}_{[k,k]}^{-1/2} \mathbf{U}_{[k,k]}^{-T} \mathbf{B}. \qquad (21)$$

The submatrix $\mathbf{U}_{[k,k]}$ must be inverted because it is no longer an orthogonal matrix. For speed, $\mathbf{U}_{[k,k]}$ undergoes LU decomposition, and each column of $\mathbf{B}$ is backsubstituted.

Note also that, as described in [12], MetricMap uses centering coefficients $c_i = \delta_{i1}$ (the first is one, the rest zero). This causes an unimportant shift in the origin of the coordinate system.

The computational complexity of MetricMap is $O(Nk^2 + k^3)$. In Landmark MDS, if m scales as k, then the computational complexity of MetricMap is the same as LMDS. LMDS may have an advantage in only requiring $O(k^2)$ work for every point in $\mathbf{B}$, rather than $O(km)$. However, because MetricMap uses different dimensions for $\mathbf{A}$ and $\mathbf{B}$, it is not known when MetricMap will yield an exact answer.

The unification of LMDS with MetricMap illustrates that the Nyström approximation can be generalized by allowing rows($\mathbf{A}$)$\geq$rows($\mathbf{B}$). Unlike the basic Nyström approximation, there are three free parameters: the final embedding dimension k, the number of rows in $\mathbf{B}$ and the number of rows in $\mathbf{A}$. The only required relationship between these is that rows($\mathbf{A}$)$\geq$rows($\mathbf{B}$)$\geq d$. Thus, the restriction in [12] of rows($\mathbf{A}$)$=2$ rows($\mathbf{B}$) is not required.

4 ACCURACY AND SPEED COMPARISONS

The main point of Section 3 is that FastMap, MetricMap, and LMDS are all applications of the Nyström

approximation. One important difference is that FastMap performs deflation, while MetricMap and LMDS require solving an eigensystem. Empirical testing can determine whether deflation is better than solving an eigensystem: does the eigensystem cause noticeable slowing? Does it yield extra accuracy?

Another difference is that MetricMap uses a matrix $\mathbf{A}$ that has more rows than $\mathbf{B}$, unlike FastMap and LMDS. Does using a larger matrix help the quality of the embedding?

To answer these questions, FastMap, MetricMap, and LMDS are compared on two medium-to-large data sets: the Reuters collection (a UCI KDD dataset [9]) and the Corel Image Features data set (also from UCI KDD).

4.1 REUTERS

The Reuters data set is a collection of Reuters news articles, stored in SGML. The algorithms are tested on the ModApte training set of Reuters, consisting of 9603 labeled documents, categorized with 115 labels, with multiple labels per document allowed.

The documents are converted into features vectors in a standard way. If the news article has an identified title and body, words in those are used. Otherwise, words in the text of the news article are used. All words are folded to lower case and then stemmed. Common words are removed from the list, and the remaining unique stemmed words in the corpus became features. The feature dimensionality is 22226, which is extremely high: larger than the number of examples in the set.

Each document i is represented by the standard tf-idf vector representation, denoted by t_{ik}, which is normalized to unit length. The distance squared matrix is then computed via

$$D_{ij}^2 = 1 - \sum_k t_{ik} t_{jk} \tag{22}$$

where the ith document is the ith row in t_{ik}.

FastMap, LMDS, and MetricMap are applied to the resulting distance matrix. The quality of the algorithms is measured in three ways: how much RMS relative distance error is introduced by the embedding, the F_1 retrieval quality score of the nearest neighbor in the embedded space, and CPU time. Two different settings of m are chosen for the LMDS experiments. First, $m = 3k$ is run. FastMap uses a heuristic to find the two farthest points that requires $3k$ distance rows to be computed. In order to match the computation of FastMap, LMDS is run with the same number of rows. The second experiment uses LMDS run with $m = 600$, which measures LMDS in a regime of high quality. MetricMap is run with rows($\mathbf{A}$) = 2 rows($\mathbf{B}$), as suggested by [12].

4.1.1 Relative Distance Error

The RMS relative distance error is measured by taking 100 documents at random from the set and measuring the 100×100 distance matrix, both in the original unembedded space, and in the embedded space, while varying the dimension k of the embedding. The RMS relative distance error is defined to be

$$\text{Error} = \sqrt{\frac{1}{10000} \sum_i (sE_i/t_i - 1)^2} \tag{23}$$

where t_i is the true (unembedded) distance, E_i is the estimated distance (in the embedded space), and s is a scaling factor. The lower this quantity, the better the embedding.

All three algorithms tend to underestimate the true distance between objects. However, in most applications, absolute distance is not needed: mean relative distance between items is the important quantity. A linear rescaling of the distance is thus harmless. Such a rescaling is reflected in s, which is the optimal scaling that maps E_i into t_i. This scaling is chosen to minimize the cost function in (23):

$$s = \frac{\sum_i E_i/t_i}{\sum_i E_i^2/t_i^2}. \tag{24}$$

Number of Dimensions	LMDS $m = 600$	LMDS $m = 3k$	FastMap	MetricMap
5	0.659	0.611	0.694	0.841
10	0.536	0.577	0.659	0.652
20	0.458	0.466	0.564	0.863
50	0.386	0.418	0.441	0.725
100	0.338	0.339	0.413	0.573
200	0.298	0.298	0.348	0.587

Table 1: RMS relative distance error on Reuters (lower is better).

The RMS error for Reuters is shown in Table 1. Three conclusions can be reached from this table.

First, when LMDS is only allowed to access $m = 3k$ distance rows (the same as FastMap), it still outperforms FastMap and MetricMap. This is because the heuristic in FastMap does not work well on text feature vectors: there are many documents that have very little overlap with each other, because their words do not overlap. The heuristic picks two examples that have little overlap as pivot points. The FastMap algorithm

then assigns most documents to the center of the coordinate, because most documents have little overlap with either of the two pivot points. Thus, it requires roughly twice as many dimensions to reach the same RMS error.

Second, further accuracy gains for LMDS can usually be had by increasing m above $3k$, although those gains are slight. These gains are shown in more detail, below.

Third, the extra information in the $2k \times 2k$ matrix $\mathbf{A}$ in MetricMap does not help the quality of the embedding: in fact, it actively harms it.

4.1.2 F_1 Retrieval Metric

Another metric for the algorithms is how the dimensionality reduction affects text retrieval and categorization. This is tested by finding, for each document, the nearest other document in the mapped space. Then, for all nearest neighbor pairs and for all labels, a standard microaveraged F_1 retrieval score is computed. Let A = the number of labels that are shared by any nearest pair. Let B = the number of labels that appear on one of the pair, but not the other. The F_1 score is then $F_1 = A/(A + B/2)$. The higher this quantity, the better the mapping is for text retrieval.

Number of Dimensions	LMDS $m = 600$	LMDS $m = 3k$	Fast-Map	Metric-Map
5	0.520	0.458	0.467	0.396
10	0.625	0.581	0.521	0.412
20	0.714	0.653	0.629	0.489
50	0.754	0.717	0.709	0.430
100	0.768	0.757	0.724	0.434
200	0.768	0.768	0.753	0.346

Table 2: F_1 retrieval metric on Reuters (higher is better).

The F_1 metric for Reuters is shown in Table 2. There are several interesting results in this table. First, the F_1 retrieval metric when applied to the original unmapped distances is 0.719. Thus, above 100 dimensions, the F_1 score for the mapped distances can be better than the unmapped. These algorithms are implementing a form of Latent Semantic Analysis [6], which is known to improve retrieval. Second, the F_1 scores roughly track distance error: LMDS with $m = 600$ beats LMDS with $m = 3k$ beats FastMap beats MetricMap. Finally, for dimensions above 100, gains in F_1 performance start to asymptote.

Number of Dimensions	LMDS $m = 600$	LMDS $m = 3k$	Fast-Map	Metric-Map
5	51.6	0.2	0.2	0.1
10	51.7	0.4	0.4	0.1
20	51.9	0.8	0.7	0.3
50	52.3	2.5	2.3	0.9
100	53.0	8.4	6.6	2.9
200	55.7	55.7	20.0	14.9

Table 3: CPU time (in seconds) on Reuters (lower is better).

4.1.3 CPU Time versus Accuracy

The CPU time for the three algorithms is shown in Table 3. The experiments are run on an unloaded 2.4 GHz Xeon PC running Windows Server 2003 with 1.5 GB of RAM. The datasets are small enough to fit into memory. Profiling the code shows that the time is dominated by the computation of the distance matrix elements: the eigendecomposition and projection take very little extra time.

Comparing FastMap and LMDS with $m = 3k$ is illustrative. For dimensions less than 200, the timings are very comparable. With $m = 3k$, LMDS does not incur a significant time penalty, but provides increased performance over FastMap. MetricMap is fastest, but the poor accuracy results in Table 2 indicate that the algorithm should be avoided.

Algorithm	m	RMS dist error	F_1	CPU time
FastMap	(150)	0.441	0.709	2.3
LMDS	60	0.424	0.687	0.9
LMDS	100	0.428	0.704	1.5
LMDS	150	0.417	0.717	2.5
LMDS	200	0.403	0.730	4.0
LMDS	300	0.380	0.742	8.1
LMDS	600	0.386	0.753	52.2

Table 4: Comparing performance of LMDS and FastMap on Reuters for different m ($k = 50$).

To better understand the behavior of LMDS, the tests are run for a fixed dimensionality $k = 50$ and for different values of m. The accuracy and CPU results are shown in Table 4. This table illustrates that LMDS permits a time/accuracy trade-off by varying m. Increasing m above $3k$ will increase the accuracy, at the cost of increased CPU. For this problem, the analysis time is not burdensome, so the increase accuracy is worthwhile. Eventually, the eigenvectors and eigenvalues of $\mathbf{K}$ are accurately estimated, so increasing m further does not help the accuracy.

4.2 COREL IMAGE FEATURES

The Corel Image Features is a UCI KDD data set consisting of features extracted from 68,040 images. Four sets of features are extracted: 32 color histograms, 32 color layout histograms, 9 color moments, and 16 texture features. Each of these features is continuous. No labels are provided in the data set. Images with missing features are ignored, leaving 66,615 images. The Corel Image Features dataset probes the algorithms in a different way. The data set size is larger and more realistic.

We computed an overall distance between two image feature vectors by first computing the distance between each set of features. For the color histograms and color layout histograms, we used the chi-squared distance [10]:

$$D_{ij}^{\text{chi}} = \sum_n \frac{2(h_{in} - h_{jn})^2}{h_{in} + h_{jn}} \tag{25}$$

where h_{in} is the nth histogram bin for ith image. For color moments and texture features, we used Euclidean distance (not squared).

At this point, there are four distances for each pair of images. These distances are combined into a single distance by a weighted sum, where the weights are computed so that the average of each of the four distances is unity across the database:

$$D_{ij}^{\text{total}} = \frac{D_{ij}^{\text{colorHist}}}{2.457} + \frac{D_{ij}^{\text{layout}}}{2.006} + \frac{D_{ij}^{\text{moments}}}{4.295} + \frac{D_{ij}^{\text{texture}}}{7.212}. \tag{26}$$

Again, the Corel Features dataset stresses the algorithms more than Reuters: the distance is not a Euclidean distance, but a sum of heterogeneous distances, which can provoke negative eigenvalues.

4.2.1 Relative Distance Error

Number of Dimensions	LMDS $m = 150$	LMDS $m = 3k$	FastMap	MetricMap
1	0.516	0.529	0.545	0.540
2	0.326	0.377	0.384	0.482
5	0.142	0.174	0.254	0.373
10	0.075	0.086	0.147	0.390
20	0.069	0.082	0.124	0.546
50	0.108	0.108	N/A	0.530

Table 5: RMS relative distance error on Corel Image Features (lower is better).

Because there are no image labels provided, we test the effectiveness of each algorithm with RMS relative distance error, as computed in (23). These accuracy results are shown in Table 5. There are severable notable features of the data in this table. First, FastMap yielded negative pivot distances for $k = 50$. That is, even the two farthest points in the database have negative squared distance after 50 rounds of deflation: no result for $k = 50$ could be produced by FastMap. Second, LMDS provides a noticeable improvement in accuracy for $k > 1$, even when $m = 3k$. LMDS with $m = 150$ is even more accurate. Third, as in Reuters, MetricMap has far worse accuracy than either of the other algorithms. Finally, the innate dimensionality of this data set may be less than 50, because the results for $k = 50$ are worse than for $k = 20$. This can happen, because the original distance matrix is non-Euclidean.

4.2.2 CPU Time versus Accuracy

Number of Dimensions	LMDS $m = 150$	LMDS $m = 3k$	FastMap	MetricMap
1	13.3	0.2	0.2	0.1
2	13.3	0.5	0.5	0.2
5	13.4	1.3	1.4	0.5
10	13.5	2.6	2.7	0.9
20	13.7	5.3	5.6	1.9
50	14.4	14.4	N/A	5.5

Table 6: CPU time (in seconds) on Corel Image Features (lower is better).

CPU times for the algorithms on this dataset are shown in Table 6. None of the CPU times are onerous. As in Reuters, LMDS for $m = 150$ is dominated by the computation of the distance matrix. However, for $m = 3k$, LMDS is competitive with FastMap. MetricMap is substantially faster (because it computes the fewest distance matrix elements), but its poor accuracy is not worth the increased speed.

Algorithm	m	RMS dist error	CPU time
FastMap	(60)	0.124	5.6
LMDS	40	0.125	3.6
LMDS	60	0.082	5.4
LMDS	100	0.072	9.0
LMDS	150	0.069	13.7
LMDS	200	0.069	18.7
LMDS	300	0.067	30.5

Table 7: Comparing performance of LMDS and FastMap on Corel Image Features for different m ($k = 20$).

Finally, the performance of LMDS is examined as a function of m for fixed $k = 30$ in Table 7. As m increases, LMDS surpasses the performance of FastMap until $m = 100$, where the accuracy is reaches a plateau and further increases in m do not help.

5 CONCLUSIONS

This paper has shown that FastMap, MetricMap, and Landmark MDS (LMDS) all utilize the Nyström approximation to the eigenvectors of a large matrix. This unification highlights how the algorithms are related and can lead to future work.

These algorithms use different variations on the Nyström approximation. LMDS uses the basic Nyström approximation. FastMap uses deflation to embed items one dimension at a time. MetricMap uses an expanded matrix to fix the embedding of the landmark points. However, empirical evidence has shown that the variations of the Nyström algorithm are not beneficial for MDS.

The deflation in FastMap limits the time/accuracy tradeoff that is inherent in the basic all-dimensions-at-once Nyström approximation. By increasing m (the number of rows computed in the original distance matrix), LMDS becomes more accurate, but the computation of the distance matrix elements require more time. FastMap freezes this tradeoff by choosing a constant number of rows per dimension. Even with the same number of rows of the distance matrix, LMDS is more accurate. Thus, the all-dimensions-at-once LMDS algorithm is preferred.

MetricMap uses a larger matrix than basic Nyström to compute the embedding coordinates of the landmark points. Empirically, this causes a substantial decrease in accuracy. This decrease in accuracy is probably because increasing the size of the $\mathbf{A}$ matrix does not improve the fitting of the landmark points: the number of landmark points grows as the dimension of $\mathbf{A}$. Thus, the extra data in $\mathbf{A}$ is not used to eliminate noise in the position on the landmark points.

In conclusion, the basic Nystroḿ approximation in LMDS is faster and more accurate than the Nystroḿ variations proposed in FastMap and MetricMap. This superiority may carry over to other data sets and applications other than MDS.

Acknowledgements

I would like to thank Chris Burges and Dimitris Achlioptas for discussions and help with the paper.

References

[1] S. Belongie, C. Fowlkes, F. Chung, and J. Malik. Spectral partitioning with indefinite kernels using the Nyström extension. In *Proc. ECCV*, 2002.

[2] Y. Bengio, J.-F. Paiement, and P. Vincent. Out-of-sample extensions for LLE, Isomap, MDS, Eigenmaps and spectral clustering. In S. Thrun, L. Saul, and B. Schölkopf, editors, *Proc. NIPS*, volume 16, 2004.

[3] M. Chalmers. A linear iteration time layout algorithm for visualizing high-dimensional data. In *Proc. IEEE Information Visualization*, pages 127–132, 1996.

[4] T. Cox and M. Cox. *Multidimensional Scaling*. Number 59 in Monographs on Statistics and Applied Probability. Chapman & Hall, 1994.

[5] V. de Silva and J. B. Tenenbaum. Global versus local methods in nonlinear dimensionality reduction. In S. Becker, S. Thrun, and K. Obermayer, editors, *Proc. NIPS*, volume 15, pages 721–728, 2003.

[6] S. C. Deerwester, S. T. Dumais, T. K. Landauer, G. W. Furnas, and R. A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.

[7] C. Faloutsos and K. Lin. FastMap: a fast algorithm for indexing, data-mining, and visualization. In *Proc. ACM SIGMOD*, pages 163–174, 1995.

[8] G. Golub and C. V. Loan. *Matrix Computations*. Johns Hopkins University Press, 1983.

[9] S. Hettich and S. Bay. The UCI KDD archive. [http://kdd.ics.uci.edu] Irvine, CA: UCI, Dept. Information and Computer Science.

[10] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw Hill, 1991.

[11] B. Schölkopf. The kernel trick for distances. In *Proc. NIPS*, pages 301–307, 2000.

[12] J. T.-L. Wang, X. Wang, K.-I. Lin, D. Shasha, B. A. Shapiro, and K. Zhang. Evaluating a class of distance-mapping algorithms for data mining and clustering. In *Proc. ACM KDD*, pages 307–311, 1999.

[13] C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *Advances in Neural Information Processing Systems*, volume 13, pages 682–688, 2001.

[14] M. Williams and T. Munzner. Steerable, progressive multidimensional scaling. In *Proc. IEEE Information Visualization*, pages 57–64, 2004.