

Some uses of spectral methods in Computer Science

Abhiram G. Ranade

Indian Institute of Technology, Mumbai, India,
ranade@cse.iitb.ac.in,
WWW home page: <http://www.cse.iitb.ac.in/~ranade>

Abstract Methods based on the analysis of eigenvalues or singular values of a matrix, often called *spectral methods* are very popular for many applications including graph partitioning, clustering, recognition, compression. This paper will survey some of these applications and present the basic underlying ideas.

1 Introduction

The purpose of this tutorial is to give a short informal introduction to some of the uses of spectral methods. Spectral methods are applicable to a wide range of problems; we will concentrate on those uses related to Principal Component Analysis (PCA)[12]. Principal component analysis, or the underlying Singular Value Decomposition[11] have been successfully used in a variety of problems, some of which are as follows:

1. Search. Given a database consisting of documents or images, identify those documents that closely match a given query (which might itself be a document or an image).
2. Clustering. This is related to the above problem: divide the documents or images into clusters such that documents/images in the same cluster are similar, while those in distinct clusters are dissimilar.
3. Compression. How to compactly store an image.
4. Finding the principal axes (related to rotation) in a solid model. These axes might be useful for ascertaining mechanical properties, obviously, but they are often used to canonically orient the solid for the purposes of matching with other solid models, say from a database of solid models.
5. Summarization. Given an English language paragraph, identify those sentences which are the most representative of the paragraph.
6. Given a graph, partition it into subgraphs which are sparsely connected with one another, with each subgraph being possibly densely connected internally. For example the problem of partitioning a circuit amongst circuit boards connected by as few wires as possible can be directly seen as a graph partitioning problem. More generally, graph partitioning is the key to employing divide and conquer techniques on graphs, and arises in many areas.

This document is not intended to be comprehensive, and even for some of the basic definitions the reader might need to refer to standard texts on the subject. Our goal is more to provide intuition, for this purpose we discuss several elementary as well as complex applications in which PCA/SVD has been used. It is hoped that this document together with basic texts will give a good introduction to enable creative use of these methods in Computer Science problem solving.

We begin by defining a generic context in which SVD is applicable. We then present the basic mathematical ideas of SVD. We then discuss several applications.

2 Generic Problem

The input to our problem is a set of n points in m -dimensional space. We will assume throughout that these are given to us as an $n \times m$ matrix A , in which the i th row gives the coordinates of the i th point. The main question of interest is: does this cloud of points have an interesting shape. Often, the point cloud will be a representation of some real phenomenon or object, and the shape of the cloud will be indicative of some important property of the phenomenon. Hence our interest.

Here are some of the ways in which the point cloud might have arisen. For example, each point might represent an image with a_{ij} giving the grayscale value of the j th pixel in the i th image in a database. Or points might be documents from a database with a_{ij} denoting whether the i th document contains the j th term (as per some numbering of the terms appearing in the entire database). Or the points might correspond to students, with a_{ij} denoting the marks obtained by the i th student in the j th subject. In each such scenario, we expect the entries of A to be correlated. For example, very likely a document that contains the term “fiscal” also contains the term “deficit”, or that if a student gets high marks in physics, the marks in mathematics could not be too low. These correlations will get reflected in the shape of the corresponding point cloud, and vice versa.

Discovering such correlations/patterns in A is important for answering queries of the kind discussed in the introduction, and in general for understanding the phenomenon/process from which A has arisen.

In many interesting practical situations the pattern which we expect to find in A is that A is a low rank matrix with some added noise. In other words, the rows interpreted as points in m dimensional space are not scattered randomly, but in fact sit in a low dimensional subspace. We will discuss at some length why this happens, but note first that it is precisely under these circumstances that Principal Component Analysis (or the linear algebraic technique of Singular Value Decomposition) is useful. In fact, SVD allows us to identify the relevant subspace even in presence of noise – and this is its main power.

Under what circumstances do we expect A to have low rank? We explain this using the example in which A gives marks obtained by students. We begin by hypothesizing that although a student is tested in several subjects, there really are 3 abilities that are of consequence in all subjects: quantitative, logical,

and verbal. Suppose that numbers q_i, l_i, v_i characterize these abilities of the i th student. Suppose further that the j th subject is characterized by numbers Q_j, L_j, V_j which denote the extent to which it tests these abilities. Finally we hypothesise that the marks obtained by the i th student in the j th subject are given as:

$$q_i Q_j + l_i L_j + v_i V_j \tag{1}$$

If in fact our hypotheses are correct, then it should be clear that any matrix A consistent with the above process will have rank 3 at most. There are two key ideas in all this:

1. Assumption of a small number of latent factors. Although the number of observables is m , the number of subjects, we are expecting that the marks scored only have three-fold variation, corresponding to the 3 hypothesized abilities.
2. Bilinear effect of the factors. The entries of A are assumed to be determined by expressions of the form (1). In fact defining a $n \times 3$ student matrix P in which $p_{i1} = q_i, p_{i2} = l_i$, and $p_{i3} = v_i$, and a subject matrix T with $t_{1j} = Q_j, t_{2j} = L_j$, and $t_{3j} = V_j$, the marks expected for the students simply are:

$$A = PT$$

This is only in absence of error and assuming our assumptions hold; otherwise the observed matrix A will include some error (hopefully small) as well.

We will soon see that such a model based on latent factors is useful for other kinds of collections, e.g. documents or images in a database. All such collections can be productively dealt with using Singular Value Decomposition, which we describe next.

3 Singular value decomposition

Given an $n \times m$ matrix A , its first (right) singular vector v is defined as that vector from the set of all unit vectors w such that $\|Aw\|_2$ is maximum. In other words, the first singular vector is the one that is stretched most under the action of A . Notice that since Aw is composed of dot products of the rows of A and w , this will be maximized by that w which is closest in direction to most of the rows of A . Intuitively, if we consider the i th row to be the coordinates of the i th point, then to a first approximation, all these points lie on the line through the origin in the direction of v – or this is the one dimensional subspace closest to the rows of A . The projection of the rows of A onto v , given by the matrix $(Av)v^T$ is indeed the best rank 1 approximation to A in the Frobenius norm, i.e.

$$\|A - Avv^T\|_F$$

is the smallest over $\|A - B\|_2$ where B is any rank 1 matrix.

Writing $A' = A - Avv^T$, we simply repeat the above procedure on A' to get vectors v' and so on. Clearly this process will terminate after r steps, where r is

the rank of A . Let v_1, v_2, \dots, v_r denote the vectors identified by this procedure (with $v_1 = v, v_2 = v'$ as we defined). These are the right singular vectors of A . Likewise define u_1, u_2, \dots, u_r to be the left singular vectors (working with A^T).

Let V denote the matrix formed by using v_i as columns. Then by construction V is orthogonal. But the interesting fact that emerges is that the columns of matrix AV are also orthogonal; not only that, the i th column is in the direction of u_i ! This may be stated as $AV = U\Sigma$, where Σ has entries only along the diagonal. The matrices U and V as defined above have only r columns. However, we can add $m - r$ columns to V to make it orthogonal (any columns that are mutually orthogonal and orthogonal to the columns originally present will do), and similarly we extend U also to be an orthogonal matrix. To keep the relation $AV = U\Sigma$ intact we extend Σ suitably with 0s. Then we can write this as a decomposition:

$$A = U\Sigma V^T$$

The i th entry on the diagonal of Σ is called the i th singular value, and is denoted by σ_i . We will abuse notation and call the singular vector associated with the largest singular value the largest singular vector and so on.

Define U_k, V_k to be matrices composed of the first k columns of U, V . Let Σ_k denote the top left $k \times k$ submatrix of Σ . It is also possible to show that $U_k \Sigma_k V_k^T$ is indeed the best rank k approximation to A in the Frobenius norm. Further the error $\|A - U_k \Sigma_k V_k^T\|_F^2$ of this approximation is $\sigma_{k+1}^2 + \dots + \sigma_r^2$ where $r = \text{rank}(A)$.

SVD can be computed in time $O(mn^2 + m^2n)$, see [11].

3.1 Singular values vs. Eigenvalues

Singular value decomposition is related to the more well known Eigenvalue decomposition, but in some sense singular value decomposition is intuitively easier.

It is easily seen that the left singular vectors of A are the left eigenvectors of AA^T , and similarly for the right. Eigenvalues are squares of corresponding singular values. Singular values are real for all matrices by definition, while eigenvalues may in general be imaginary. While singular vectors can be thought of as approximating the directions of the rows/columns of a matrix, such an interpretation is not interesting for eigenvectors.

4 Applications

The applications we have selected are not all necessarily (commercially or scientifically) important; our hope is that they will help build up intuition.

4.1 Matching solid models

Suppose we are given (suitable descriptions of) two solid models M and M' . We would like to know if one can be obtained by translating/rotating the other.

This check is easy if they are consistently aligned. Finding the alignment which reveals the similarity can be easily accomplished using SVD.

We start by computing the centroids (centers of mass) of the models and aligning those; assume without loss of generality that this has already been done. Next we generate matrices A and A' respectively in which each row is a triple giving the coordinates of (unit) masses obtained by digitizing the models. Next we compute the singular value decompositions $A = U\Sigma V^T$ and $A' = U'\Sigma'V'^T$ respectively. Now all we need to check is whether $U\Sigma \approx U'\Sigma'$ – if these products agree sufficiently, then the models must be identical! The proof is an easy exercise.

It might be worth noting that the columns of V^T and analogously of V'^T give the principal axes of rotation for the two models. The first singular vector in fact gives the direction of the axis about which the moment of inertia is the smallest. This should also be obvious given that the moment of inertia is defined as $\sum_j m_j r_j^2$ where r_j is the perpendicular distance to the axis of rotation of a mass m_j . This is consistent with the intuition that the moment of inertia should be large if the masses are far from the axis; since the first singular vector approximates A , the masses should be closest to it.

4.2 High dimensional matching

A classic problem is: Given a document or image, find the one (or several) which is close to it from a given database. This problem is difficult because it is expected that “close match” be defined with respect to “important features” for the document[8, 5, 14]/image¹. What constitutes “important features” is not specified but is left to the algorithm designer. Finally, the items (i.e. images/documents or whatever) in the database might contain noise.

The items in the database are described by a matrix A with a single item being described by a single row, and every column corresponding to a measurement – which could be the value of the corresponding pixel in every image, or whether or not a particular word is present in each document. The key hypothesis, as before, is that the rows interpreted as points will not be uniformly distributed, but will lie in a small dimensional subspace. For documents, this happens because although there might be many documents in the database and many words will be used in them, there are only a few topics about which the documents will be written. The topics will thus comprise the latent factors [8], while the probability that term j appears in document i will be proportional to the relatedness of the term to the topic as well as the extent to which the document concerns the topic (bilinearity property). The matrix A is expected to have noise partly because this model is only a heuristic, but also because the probability estimates must eventually get (randomly) rounded up/down to actual presence or absence of terms.

Each right singular vector of A is customarily interpreted as a topic. If the i th row has a large projection on the j th right singular vector, then the i th

¹ For this idea applied in the context of faces see [19].

documents is labelled as being related to the j th topic. Singular vectors with very low singular values are discarded, as being indicative of the noise in the model/process. It is very common to find that only a small number of singular values are large – and also that documents grouped as above do indeed correspond to human intuitions about “belonging to a topic”.

As to finding documents that match a query: we essentially take a dot product of the query and the documents in the database, but only after projection on the subspace defined by the important singular vectors. On one hand this will be seen as eliminating noise (since we ignore the dimensions labelled irrelevant/noisy), but it has a much more interesting benefit. Through such processing we can effectively recognize synonymy – and use it in the matching process, while having no knowledge about the semantic relationships amongst the terms!

Here is a brief explanation. Suppose we have a database in which there are documents about two broad topics, automobiles and religion. Assume for simplicity that each of these topics has its own distinct vocabulary. Thus the matrix A would have a block form if we knew all this and we ordered the terms/documents corresponding to religion before those corresponding to automobiles. Now the first singular vector is likely to be in the direction of the topic which has more documents, say automobiles. Suppose now that we enter a query consisting of the word “automobile”. In this case, the idea is to treat this query as a (very small) document, and retrieve documents that have a high dot product with it. If we perform this dot product in the original matrix, then no document would be retrieved unless it actually contains the given word in it. On the other hand, the first singular vector will have non-zero components in all terms relating to cars. Thus all car related documents including the query document will share a component in the direction of the first singular vector. Thus documents which contain the terms such as “car” will be returned even if they do not contain the term “automobile”.

4.3 Summarization

In the *text summarization* problem, we are given a set of sentences, and we are required to select a subset which could be considered to be a good summary of the entire set.

Bellare et al[6] present an interesting algorithm for this problem based on SVD. Their work contains a number of additional ideas, e.g. use of WordNet to understand relationships between words such as synonymy. We will only state the essence as it relates to SVD. Treating each sentence as an independent document, they first create a matrix A as described earlier, i.e. a_{ij} indicates presence of j th term in the i th sentence. Next they compute $U\Sigma V^T = A$. Noting that this can be written as $U\Sigma = AV$, they pick as small a subset of rows A' as possible such that $A'V_k \geq \alpha U_k \Sigma_k$, where α is some fixed constant smaller than 1, and k is suitably chosen. In the above, we write $R \geq S$ to mean each $r_{ij} \geq s_{ij}$ for all i, j . The idea is that the rows or A' adequately cover (to extent α) the important singular vectors of A , and hence can be considered to be a good summary. The

sentences corresponding to the subset A' are output. It appears that this method gives good results.

4.4 Compression

If $U_k \Sigma_k V_k^T$ provides a good enough approximation to A , then we could consider using U_k , Σ_k and V_k as a representation for A . A has mn entries, U_k, V_k have mk and nk respectively, and Σ_k has k . Thus the A can be represented using $k(m+n+1)$ numbers rather than mn . This can be a considerable saving if k is small. In many applications, such as when the matrix A consists of an $m \times n$ image, with a_{ij} denoting the grayscale value² of the ij th pixel, good enough approximation can indeed be obtained with small k [3, 10, 20].

5 Graph Partitioning

There are many ways to define the graph partitioning problem, we consider one of these. Suppose removal of a collection C of the edges in a graph $G = (V, E)$ partitions V into two sets V_1, V_2 such that no edge in $E - C$ connects vertices from V_1 to V_2 . Then we define the ratio of C to be

$$\frac{|C|}{\min(|V_1|, |V_2|)}$$

Then the ratio cut problem is to find the cut with the minimum ratio.

One way to relate this problem to SVDs is to observe that vertices of the graph can be likened to documents and edges to terms.³ Such an interpretation is more appropriate for hypergraphs, since a term may appear in several documents, and a hyper edge may also appear in several vertices rather than edges. Here we present another way to relate SVD to the partitioning problem.

Suppose a graph is embedded in some Euclidean space such that adjacent vertices are placed closer than non adjacent vertices, and in either case no two vertices are too close. Suppose further that such an embedding occupies a $d_1 \times d_2 \times \dots \times d_w$ volume of space. Now if we slice this volume by a hyperplane perpendicular to the i th dimension, then the geometric size of this cut will be $\prod_{k \neq i} d_k$. Clearly, this will be minimum when d_i is largest. The geometric cut will induce a partition of the embedded graph. While the geometric size of the cut can in general be very different from the number of edges crossing the cut, we could consider finding such minimum size cuts to be a reasonably good heuristic for finding small edge cuts of the graph. The problem of finding the largest d_i is in some sense similar to the problem of finding the direction in a solid body in which the moment of inertia is the smallest (Section 4.1). This is how we use the SVD.

² Similarly for colour.

³ And topics are in fact the desired partitions.

This heuristic is commonly implemented using embedding induced by the edge incidence matrix of the graph, i.e. the rows of this matrix give the embedding of the corresponding vertex in an m dimensional space, where m is the number of edges. Suppose for simplicity that the graph is k regular. Vertices connected by an edge are at a distance $\sqrt{2k-1}$, while those not connected are at a distance $\sqrt{2k}$. In other words, no two vertices are too close, however adjacent vertices are slightly closer than non adjacent vertices. So this simple embedding satisfies the naive requirements mentioned earlier. As we remarked, the naive requirements are very weak, and in general unlikely to give good partitions, but it turns out that this simplest, most natural embedding is useful! In fact the following algorithm is guaranteed to find a good ratio cut.

Algorithm:

1. Form the edge incidence matrix A of the graph. For this we number the edges in any order, and set $a_{ij} = 1$ if edge j is incident on vertex i and 0 otherwise.
2. Find $q = [q_1, q_2, \dots, q_n]$ the second left singular vector. This is equivalent to projecting the vertices on the line given by the second right singular vector.
3. For each q_j , consider the partition in which all vertices i with $q_i \leq q_j$ are on one side of the cut, and $q_i > q_j$ are on the other side. Of these return the partition with the best ratio.

As should be clear, the above algorithm indeed uses a hyperplane perpendicular to the second right singular vector to form the partition – and of the $n-1$ possible hyperplanes, it selects the one which gives the best ratio. The reader might be curious as to why the second singular vector is to be used rather than the first. With the edge incidence embedding the points are not centered at the origin, and for regular graphs, the first right singular vector (which can be seen to be $v_1 = [\frac{1}{\sqrt{m}}, \frac{1}{\sqrt{m}}, \dots, \frac{1}{\sqrt{m}}]^T$) points in the direction of the center of mass of the vertices. Thus in $A - Av_1v_1^T$ the points have simply been shifted so that their center of mass aligns with the origin. Thus we should in fact be considering the first singular vector of the new matrix which is nothing but the second singular vector of the original matrix.

Proving that this ratio is reasonably good is somewhat hard and algebraically mysterious. We only state the main theorem.

Theorem 1 ([1, 17]). *Let ϕ denote the optimal cut ratio. Then the ratio ϕ_a obtained by the above algorithm satisfies: $\phi_a \leq \sqrt{2\phi}$*

Thus the ratio of the cut found by the algorithm can be a factor $\sqrt{\frac{2}{\phi}}$ larger than the best possible cut ratio. However, in practice the algorithm appears to give good cuts, and further it can be proved that the algorithm gives nearly optimal ratios for many interesting classes of graphs[18]. For example for planar graphs, the above algorithm is guaranteed to find partitions with ratio $\sqrt{1/n}$ which is existentially tight[18].

Better results are known for graph partitioning using more complex methods. Arora, Rao and Vazirani[4] show how to find cuts that are only $O(\sqrt{\log n})$ worse than the optimal cut. As the authors point out, this result subsumes the results for spectral partitioning, and in many senses builds up on spectral partitioning.⁴

5.1 The Laplacian

Suppose we arbitrarily orient and number the edges of the input graph. We now construct an $n \times m$ matrix B where $b_{ij} = 1$ if the j th edge originates at the i th vertex, $b_{ij} = -1$ if the j th edge terminates at the i th vertex, and 0 otherwise. The Laplacian L of the graph is defined as BB^T . Many traditional statements of the spectral partitioning algorithm use the second smallest eigenvector of the Laplacian of the graph, rather than the second largest singular vector of the edge incidence matrix.

This can be easily seen to be equivalent for regular graphs. For these we can see that $BB^T = L = 2D - AA^T$. Now it is easy to prove that B will have the same singular vectors as A , except that the ordering is reversed. Thus the i th largest singular vector of A will be the same as the i th smallest singular vector of B , which in turn is the same as the i th (largest) eigenvector of L .

5.2 Metricity of the edge incidence embedding

Although the edge incidence embedding is not metric, its projections often are. For example, consider the $d = \log n$ dimensional binary hypercube. Consider the projection of its edge incidence embedding on the subspace defined by the second through $d + 1$ th (largest) right singular vectors. It turns out that this projection is itself the natural embedding of the hypercube! Another interesting fact is that the second through $d + 1$ th singular values are all equal, with subsequent eigenvalues being smaller. An interesting question is, will a small number of dimensions always give (near) metric embeddings for all graphs.

5.3 Graph colouring

We note as an aside that partitioning using the smallest singular vector of A is useful for graph colouring[2]. While this is a deep result, we might mention that for colouring we want all neighbours to get separated, while in partitioning we want fewest neighbours to get separated. So in some sense it is no surprise that opposite ends of the spectrum get used for these two purposes.

⁴ The work in [4] and also the previous important work in [13] start with a graph embedding, but it is more carefully constructed than the edge incidence embedding. These algorithms can also be viewed as approximately using a hyperplane to partition the embedding.

6 Clustering

How do you partition the rows of a matrix A (or a set of points in m dimensional space) so that “similar” rows (points that are geometrically close) get put in the same partition or cluster? There are many ways of formalizing this question. We already mentioned one strategy for partitioning documents (use latent topics themselves), and here we consider another. This is the so called k-means clustering:

Given points $a_1, \dots, a_n \in R^m$, find $c_1, \dots, c_k \in R^m$ so as to minimize

$$\sum_i d(a_i, \{c_1, \dots, c_k\})^2$$

where $d(a, S)$ is the smallest distance from a point a to any of the points in S .

The c_i are the centers of the clusters; each point is put in the cluster whose center is closest to it. In this definition we minimize the sum square distance from the points to the associated centers.

We present here an algorithm due to Drineas et al[9] which gives a clustering whose sum square distance is at most 2 times the optimal. This requires k to be a constant. In this description assume $k = 2$. Note that even with this restriction, the problem remains NP-complete for arbitrary m .

The first observation is that if the points lie in a 2 dimensional subspace (in general k) of R^m , then the problem can be solved easily. We have two centers, and the the points belonging to the two clusters can be separated by the perpendicular bisector of the line joining the centers. This will always be true – some line will separate the clusters. Further, it is also easily proved that the center selected for a group of points must also be its centroid. Thus if we knew which line to choose, we would be done. The key point is that there are only a polynomial number of lines that need to be considered. Hence by trying out all possible lines, finding the centroids of the points on the sides of each choice, and evaluating the sum square distance for each, we can pick the optimal centers.

In general, the points need not be in a k dimensional subspace. So in this case, we find the best k dimensional subspace S' , i.e. that k dimensional subspace whose total square distance from the points is as small as possible. This is clearly the subspace defined by the largest k right singular vectors of A . The algorithm simply projects the points into this subspace, and returns the optimal centers of the projections. Suppose that the projected points are denoted as a'_1, \dots, a'_n , and their optimal centers are q_1 and q_2 . Notice that in some sense we are considering the points to “really” be in the k dimensional subspace S' , and their distance from this subspace to be the “error” which we minimized.

Now we must prove that the “error” does not hurt our answer much. Let c_1, c_2 be the actually optimal centers. Let S denote the k dimensional subspace containing c_1, c_2 . Let c'_1, c'_2 be the projections of c_1, c_2 into S' . Now note that

$$\sum_i d(a_i, \{q_1, q_2\})^2 \leq \sum_i d(a_i, \{c'_1, c'_2\})^2$$

$$= \sum_i d(a_i, a'_i)^2 + \sum_i d(a'_i, \{c'_1, c'_2\})^2$$

In this the first step follows because q_1, q_2 were optimal centers for the subspace S' . The second follows by the Pythagorean theorem – the square of the distance to a point in S' is the square of the perpendicular distance to S' plus the square of the distance within S' .

Next we note that the second second sum $\sum_i d(a'_i, \{c'_1, c'_2\})^2$ is a sum of term wise projection of the sums in $\sum_i d(a_i, \{c_1, c_2\})^2$, the optimal sum square distance. Likewise we will show that the first one is also similar:

$$\begin{aligned} \sum_i d(a_i, a'_i)^2 &= \sum_i d(a_i, S')^2 \\ &\leq \sum_i d(a_i, S)^2 \\ &\leq \sum_i d(a_i, \{c_1, c_2\})^2 \end{aligned}$$

In this the first step follows since a'_i is the projection of a_i into S' . The second step because S' minimized the sum of square distances from all points. The third because $c_1, c_2 \in S$.

Thus we have established that the sum square distance $\sum_i d(a_i, \{q_1, q_2\})^2$ for the centers calculated by the algorithm is at most twice the optimal sum square distance $d(a_i, \{c_1, c_2\})^2$, as required.

7 Concluding Remarks

There are many issues in using SVD that need to be addressed specially for each application. Some of these are as follows. Say we are dealing with documents: should we use frequencies of each term in each document, or should the entries just be bits indicating presence/absence? Should each row/column of the matrix be centered at 0? We saw in the case of regular graphs that centering happens automatically through the largest singular vector. Another idea used in PCA is to normalize the vector lengths – it turns out that this idea is useful for graph partitioning when the graphs are not regular[7]. When we work with images, say for face recognition[19], it is useful to eliminate the effects due to variations in lighting. Another interesting question concerns how to form the matrix A itself. In Section 4.4 we indicated how an $m \times n$ image can be compressed by treating the grayscale value of the ij th pixel as the ij th entry of the matrix. However, it turns out that by placing the grayscale value of pixel ij in matrix entry $i'j'$ where i' and j' are fixed functions (independent of the image itself) of i, j , the compression can be substantially improved [15].

The core of most recognition/matching problems is the estimation of some latent parameters of the model which generates the given data, i.e. the matrix A considered in this paper. In general, this estimation is done by maximum likelihood techniques – that collection of model parameters is selected that is

most likely to generate the observed data. If the effects are bilinear (Section 2), and the noise is Gaussian, then the subspaces constructed by SVD are most likely to generate the given matrix A . Otherwise, the maximum likelihood estimation must be done with more ad hoc methods, e.g. using gradient descent which might return models which are local maxima[16]. So in some sense SVD represents a good tradeoff between simplicity of assumptions and the availability of a dependable algorithm.

References

- [1] N. Alon. Eigenvalues and expanders. *Combinatorica*, 6(2):83–96, 1986.
- [2] Noga Alon and Nabil Kahale. A spectral technique for coloring random 3-colorable graphs (preliminary version). In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 346–355. ACM Press, 1994.
- [3] Harry C. Andrews and Claude L. Patterson. Singular Value Decomposition (SVD) Image Coding. *IEEE Transactions on Communications*, 24:425–432, April 1976.
- [4] Sanjeev Arora, Satish Rao, and Umesh Vazirani. Expander flows, geometric embeddings and graph partitioning. In *STOC '04: Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pages 222–231. ACM Press, 2004.
- [5] Yossi Azar, Amos Fiat, Anna Karlin, Frank McSherry, and Jared Saia. Spectral analysis of data. In *STOC '01: Proceedings of the thirty-third annual ACM symposium on Theory of computing*, pages 619–626. ACM Press, 2001.
- [6] Kedar Bellare, Anish Das Sarma, Atish Das Sarma, Navneet Loival, Vaibhav Mehta, Ganesh Ramakrishnan, and Pushpak Bhattacharya. Generic text summarization using wordnet. In *International Conference on Language Resources and Evaluation (LREC)*, 2004.
- [7] F. Chung. *Spectral Graph Theory*. American Mathematical Society, 1997.
- [8] Scott C. Deerwester, Susan T. Dumais, Thomas K. Landauer, George W. Furnas, and Richard A. Harshman. Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, 41(6):391–407, 1990.
- [9] P. Drineas, A. Frieze, R. Kannan, S. Vempala, and V. Vinay. Clustering large graphs via the singular value decomposition. In *Tenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 291–299, 1999.
- [10] C. S. Mc Goldrick, W. J. Dowling, and A. Bury. Image coding using the singular value decomposition and vector quantization. In *Image Processing And Its Applications*, pages 296–300. IEE, 1995.
- [11] G. H. Golub and C. F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, 1996.
- [12] R. A. Johnson and D. W. Wichern. *Applied Multivariate Statistical Analysis*. Pearson Education Asia, 2002.
- [13] Tom Leighton and Satish Rao. Multicommodity max-flow min-cut theorems and their use in designing approximation algorithms. *J. ACM*, 46(6):787–832, 1999.
- [14] Christos H. Papadimitriou, Hisao Tamaki, Prabhakar Raghavan, and Santosh Vempala. Latent semantic indexing: A probabilistic analysis. In *ACM Conference on Principles of Database Systems*, pages 159–168, 1998.
- [15] A. Ranade and S. M. Srikanth. A variation on svd based image compression, 2004. Manuscript.

- [16] Mehran Sahami, Marti A. Hearst, and Eric Saund. Applying the multiple cause mixture model to text categorization. In Lorenza Saitta, editor, *Proceedings of ICML-96, 13th International Conference on Machine Learning*, pages 435–443, Bari, IT, 1996. Morgan Kaufmann Publishers, San Francisco, US.
- [17] A. Sinclair and M. Jerrum. Approximate counting, uniform generation and rapidly mixing markov chains. *Information and Computation*, 82(1):93–133, 1989.
- [18] Daniel A. Spielman and Shang-Hua Teng. Spectral partitioning works: Planar graphs and finite element meshes. In *IEEE Symposium on Foundations of Computer Science*, pages 96–105, 1996.
- [19] M. Turk and A. Pentland. Face recognition using eigenfaces. *Journal of Cognitive Neuroscience*, 3(1), 1991.
- [20] P. Waldemar and T. A. Ramstad. Image compression using singular value decomposition with bit allocation and scalar quantization. In *Proceedings of NORSIG Conference*, pages 83–86, 1996.