Some uses of spectral methods

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 ith row gives the coordinates of the ith 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 jth pixel in the ith image in a database. Or points might be documents from a database with a_{ij} denoting whether the ith document contains the jth 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 ith student in the jth 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 which we will see shortly, 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. It would seem natural, that identifying ths subspace would be an important first step in any non-trivial processing of A. Principal Component Analysis (or the linear algebraic technique of Singular Value Decomposition) does precisely this. 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 ith student. Suppose further that the jth 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 a_{ij} obtained by the ith student in the jth subject are given as:

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

If our hypothesis is correct (unlikely) the student marks will be as above, if they are only approximately correct (more likely), then the student marks will be *close* to above. It should be clear that matrix A obtained as will have rank 3, independent of the number of students and subjects. More specifically, defining an $n \times 3$ 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 $t_{3j} = V_j$, we get

$$A = PT$$

Since P is $n \times 3$ this demonstrates that A has rank 3. So if our hypotheses are reasonably accurate, the marks will lie reasonably close to a rank 3 matrix. Statistical studies show that student mark matrices can indeed be approximte nicely by low rank matrices, though not necessarily rank 3.

It is useful to note why the rank was low in the above example. The first key idea is the hypothesis that although there were many observed quantities, they were governed by a small number (in this case 3) of *latent* factors. Latent, in that the factors themselves (quantitative, logical and verbal abilities) didnt get observed directly. The second key idea is that the factor effect was *bilinear*, in the sense that equation 1 is a sum of products.

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, customarily denoted as v_1, is defined as a unit vector that maximizes $||Av_1||_2$. In other words, the first singular vector is the one that is stretched most under the action of A. Notice that if the rows of A lie in a subspace, then a most stretched vector will also come from this subspace, and thus will characterize the subspace.

The first left singular vector could be defined as the right singular vector of the matrix A^T. However, there is an elegant joint definition of both left and right singular vectors:

$$u_1, v_1 = \text{unit vectors that maximize } u_1^T A v_1$$

Note that this immediately shows that the direction of Av_1 must be the same as that of u_1, and that of $u_1^T A$ the same as that v_1^T. Further, we define $\sigma_1 = ||Av_1||_2 = ||u_1^T A||_2$ as the first singular value.

An important observation is that u_1 can be thought of as a "rough" estimate of the directions of the rows of A. In particular, noting that Av_1 gives the projections of the rows of A on v, the rank 1 matrix $A_1 = Av_1v_1^T = u_1v_1^T$ can be considered to be a rank 1 approximation to A. In fact we can prove that A_1 is the best such matrix in the sense of the Frobenius norm:

$$||A - A_1||_F = \min_B ||A - B||_F$$

where B is any rank 1 matrix. Note that the Frobenius norm A_F of a matrix is simply the sum of the squares of all its entries.

Writing $A' = A - A_1$, we simply repeat the above procedure on A' to get u_2, v_2, σ_2 and so on. Clearly the rank of A' will be 1 less than that of A, and hence the process will terminate after r steps, where r is the rank of A. Let u_1, v_i, σ_i, for $i = 1$ to r be obtained in this manner. These respectively give the ith left singular vector, the ith right singular vector, and the ith singular value. A_k is defined as $\sum_{i=1}^{k} \sigma_i u_i v_i^T$. Obviously $A_r = A$.

Since in $A - A_1$ we are removing from each row its projection on v_1, we know that the rows of A' must be orthogonal to v_1. Thus it follows that v_2 is orthogonal to v_1 and so on. Thus the vectors v_i are orthogonal to each other, and similarly u_i amongst themselves.

It is customary to define U_k as consisting of the matrix made up by using $u_1, \ldots, u_k$ as its columns. Likewise V_k. Σ_k is defined to be a $k \times k$ diagonal matrix with $\sigma_1, \ldots, \sigma_k$ along the diagonal. Then clearly, $AV_r = U_r\Sigma_r$. Let U be any orthogonal matrix obtained by extending U_r, i.e. by adding any columns that are orthogonal to the columns in U_r and to each other. Likewise V. Also extend Σ_r into an $n \times m$ matrix by adding 0s. Then we have $AV = U\Sigma$. Alternately, we can write this as a decomposition of A, noting that $V^T = V^{-1}$.

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

This is the Singular Value Decomposition (SVD) of A and it can be computed in time $O(mn^2 + m^2n)$, see [11].

We note that it is easily proved that A_k is the best rank k approximation to A:

$$||A - A_k||_F = \min_B ||A - B||_F$$

where B is any rank k matrix. Further the error $||A - A_k||_F^2 = \sigma_{k+1}^2 + \ldots + \sigma_r^2$.

3.1 Examples

Our first example decomposition is

$$A = \begin{pmatrix} 1 & 1 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 0.85 & -0.53 \\ 0.53 & 0.85 \end{pmatrix} \begin{pmatrix} 1.62 & 0 \\ 0 & 0.62 \end{pmatrix} \begin{pmatrix} 0.53 & 0.85 \\ -0.85 & 0.53 \end{pmatrix}$$

Note here that direction of the first right singular vector $\begin{pmatrix} 0.53 \\ 0.85 \end{pmatrix}$ is somewhere between the directions of the rows of A.

Our second decomposition example is

$$B = \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

In this case note that the first right singular vector $\begin{pmatrix} 1 \\ 0 \end{pmatrix}$ snaps to the direction of the first row, does not point somewhere in between the directions of the two rows. In some sense it can be seen to identify the *dominant* direction. This happens when the rows are orthogonal, unlike the case for matrix A earlier.

Our third decomposition example is

$$B = \begin{pmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 3 & 0 & 0 \\ 0 & 3 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

In this case the top two singular values are both 3. Thus, the largest singular value is said to have multiplicity 2. In this case, the largest and second largest singular vectors are together determined only to within a subspace of the same dimension as the multiplicity. In this case the expression above gives $\begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix}$ and $\begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$ as the first two singular vectors. But indeed, any two orthogonal vectors in the subspace spanned by these would also work.

Our final example consists of the matrrix

$$C = \begin{pmatrix} 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix}$$

C has a block diagonal structure, with the blocks being of size 3×3 and 3×4. In this case it turns out that the matrix inherits the singular values of the blocks. The singular vectors are also suitably padded by zeros and inherited. In this case the largest singular value 2.36 comes from the second block, and the second largest, 2.00 from the first block. Both these are singular values of C. The first singular vector of the second block is $(.56\ .44\ .44\ .56)^T$, and so the first singular vector of C will be $(0\ 0\ 0\ .56\ .44\ .44\ .56)^T$. The next singular value 2 comes from block 1. The singular vector in that block is $(0.58\ 0.58\ 0.58)^T$. So the second singular vector for C will be $(0.58\ 0.58\ 0.58\ 0\ 0\ 0\ 0)^T$.

3.2 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.

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. Singular values are real for all matrices by definition, while eigenvalues may in general be imaginary. An important relationship to note is that the left singular vectors of A are the left eigenvectors of AA^T, and similarly of A^TA for the right. Eigenvalues are squares of corresponding singular values.

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. The required translation is typically found by aligning the centroids. The proper alignment is found by rotating the objects so that the principal axes of rotation match the coordinate axes. This is where SVD is useful.

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. For the two models, let matrices A and A' respectively be such that each row is a triple giving the coordinates of (unit) masses obtained by digitizing the models. Next to orient them, we compute the singular value decompositions $A = U\Sigma V^T$ and $A' = U'\Sigma' V'^T$ respectively. Now, AV and $A'V'$ respectively give the positions of masses oriented so that rotation axes align with the coordinate axes.

It is easily seen that the first singular vector gives the direction of the axis about which the moment of inertia is the smallest: 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. Now minimizing the perpendicular distance squared is equivalent to maximizing the projected distance squared along the axis – which is precisely the definition of the first singular vector.

Note that the axes are uniquely defined only if the first 3 singular values have multiplicity 1.

4.2 Document/image retrieval

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]/image[1]. What constitutes “important features” is

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

not specified but is left to the algorithm designer. Finally, the items (i.e. images/documents or whatever) in the database might contain noise.

A manifestation of this problem in the context of document retrieval is that of *synonymy*. Suppose we have a collection of documents from which we need to retrieve those concerning cars. A simple minded way of doing this would be to return those documents which explicitly contain the term "car". However, this procedure would miss documents which contain "automobile" but not "car", though clearly such documents should also be retrieved. You might suspect that this problem could be solved by using dictionaries which contain synonyms. However, consider the problem of retrieving documents about "Marathas" – clearly this should return documents containing "Bajirao" or "Peshwa" even though these documents might not explicitly contain the term "Maratha". This problem, which might be called the extended synonymy problem, is one of the problems elegantly solved using the spectral approach.

It is customary to represent the document database by matrix in which the ijth entry indicates whether the jth term/word is present in the ith 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. This is explained in a manner very similar to that in the students-marks example. The latent factors in this case are what might be called *topics*[8] (anologous to the skills in the students example). Each document contains different amounts of discussion of each topic, just as each student posseses different proficiency in each skill. Each term has different probability of occuring when different topics are being discussed, just as each skill is tested to different extents by the different subjects. The rank of the term document matrix is expected to be equal to the number of topics, except for the noise that is inevitable. The noise will arise 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.

Consider now the process of 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. This may be thought of as eliminating noise (since we ignore the dimensions labelled irrelevant/noisy). Because of it we can effectively recognize extended synonymy – and use it in the matching process, while having no knowledge about the semantic relationships amongst the terms!

Here is a brief explanation through an example simplified for ease of exposition. Suppose we have a database in which there are documents about two broad topics, religion and transportation. Assume for simplicity that each of these topics has its own distinct vocabulary. Thus the matrix would have a block form if we knew all this and we ordered the terms/documents corresponding to religion before those corresponding to transportation. Suppose that this matrix is matrix C of Section 3.1. Suppose the fourth and fith term in this matrix correspond to the words "automobile" and "car". Suppose we now have a query that requires documents about "car". A straightforward dot product with the C will return a positive match for rows 6 and 7 – corresponding to the documents in which the word acutally appears.

Suppose now that we consider do products in the subspace defined by the first two singular vectors. As discussed earlier, these are $v_1 = (0\ 0\ 0\ .56\ .44\ .44\ .56)^T$ and $v_2 = (.58\ .58\ .58\ 0\ 0\ 0\ 0)^T$. Now notice that if project the documents and the query into this subspace and only then take the dot product, then not only the 5th and 6th, but also the 4th document vector will have non-zero product with the query vector. Thus we will likely return document 4 also in the result.

This phenomenon is explained by noting that SVD effectively takes into account the *indirect* associations between the "car" query and document 4. This association arises through documents 5 and 6. It is this association which drags the singular vector midway between documents 4,5,6 – which in turn causes a match between document 4 and the "car" query.

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 jth term in the ith 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 value[2] of the ijth pixel, good enough approximation can indeed be obtained with small k [3, 10, 20].

[2] Similarly for colour.

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 $d_1 \times d_2 \times \cdots \times d_w$ volume of space. Now if we slice this volume by a hyperplane perpendicular to the ith dimension, then the geometric size of this cut will be $\prod_{k \neq j} d_k$. 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.

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 adjecent 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.

[3] And topics are in fact the desired partitions.

2. Find $q = [q_1, q_2, \ldots, q_n]$ = 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 $v_1 = [\frac{1}{\sqrt{m}} \frac{1}{\sqrt{m}} \cdots \frac{1}{\sqrt{m}}]^T$) 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 jth edge originates at the ith vertex, $b_{ij} = -1$ if the jth edge terminates at the ith 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

[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.

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 ith largest singular vector of A will be the same as the ith smallest singular vector of B, which in turn is the same as the ith (largest) eigenvector of L.

5.2 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.

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

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

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. Note that even with this restriction, the problem remains NP-complete for arbitrary m.

The algorithm is as follows. In the first step, we find the best k dimensional subspace that contains the points using SVD. By "best" k dimensional subspace we mean 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. Then we project the points onto that subspace. In the second step we find the k centers of the projected points in that subspace.

We will sketch how this can be done below. These centers, as it turns out, give a 2 approximation for the original space. 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.

We illustrate the second step of the algorithm by considering $k = 2$. We need to find two centers, for the points situated on some plane (as constructed in the first step of the algorithm). The key observations are: (1) There must exist a line (in fact the perpendicular bisector of the line joining the centers) that separates the two clusters, and (2) The optimal center for a cluster must be the centroid of the points in the cluster. 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. This idea can be extended to k dimensions.

Let S' denote the subspace found in the first step. Let $a'_1, \dots, a'_n$ denote the projections of the points into S'. Let the optimal centers for the projected points be $q_1, \dots, q_k$. Let $c_1, \dots, c_k$ be the actually optimal centers. Let S denote the k dimensional subspace containing $c_1, \dots, c_k$. Let $c'_1, \dots, c'_k$ be the projections of $c_1, \dots, c_k$ into S'. Now note that

$$\begin{aligned}\sum_i d(a_i, \{q_1, \dots, q_k\})^2 &= \sum_i d(a_i, a'_i)^2 + \sum_i d(a'_i, \{q_1, \dots, q_k\})^2 \\ &\leq \sum_i d(a_i, a'_i)^2 + \sum_i d(a'_i, \{c'_1, \dots, c'_k\})^2\end{aligned}$$

In this the first step (equality) follows from the Pythagorean theorem: the first term is the distance to the subspace S', while the second term is the distance within the subspace – these two are orthogonal and hence the Pythagorean theorem applies. The second step (inequality) follows because $q_1, \dots, q_k$ were optimal centers for the subspace S'.

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

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

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 (this is where we used the property of S', i.e. that it is the best rank k approximation). The third because $c_1, \dots, c_k \in S$.

Thus we have established that the sum square distance $\sum_i d(a_i, \{q_1, \ldots, q_k\})^2$ for the centers calculated by the algorithm is at most twice the optimal sum square distance $\sum_i d(a_i, \{c_1, \ldots, c_k\})^2$, 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 ijth pixel as the ijth 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 Loiwal, Vaibhav Mehta, Ganesh Ramakrishnan, and Pushpak Bhattacharya. Generic text summarization using wordnet. In *Internationational Conference on Language Resources and Evaluation (LREC)*, 2004.

[7] F. Chung. *Spectral Graph Theory.* Americal 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, S. M. Srikanth, and Satyen Kale. A variation on svd based image compression, 2006. To appear in Image and Vision Computing.

[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.