CS168: The Modern Algorithmic Toolbox
Lecture #9: The Singular Value Decomposition (SVD) and Low-Rank Matrix Approximations

Tim Roughgarden & Gregory Valiant

April 27, 2015

1 Low-Rank Matrix Approximations: Motivation

Consider an $n \times d$ matrix $\mathbf{A}$. Perhaps $\mathbf{A}$ represents a bunch of data points (one per row), or perhaps $\mathbf{A}$ represents a single object, like a rectangular image (with entries = pixel intensities). We've discussed "dimensionality reduction" for vectors — re-representing vectors in d dimensions as vectors in k dimensions, with $k \ll d$ — what's a good notion of dimensionality reduction for matrices?

One good answer, explored in this lecture, is to reduce the *rank* of the matrix. Recall from your linear algebra class that the following are equivalent definitions for the rank of a matrix $\mathbf{B}$ to be k (any one of the conditions implies the other two):

1. The largest linearly independent subset of columns of $\mathbf{B}$ has size k. That is, all d columns of $\mathbf{B}$ arise as linear combinations of only k different n-vectors.

2. The largest linearly independent subset of rows of $\mathbf{B}$ has size k. That is, all n rows of $\mathbf{B}$ arise as linear combinations of only k different d-vectors.

3. $\mathbf{B}$ can written as, or "factored into," the product of long and skinny $(n \times k)$ matrix $\mathbf{Y}_k$ and a short and long $(k \times d)$ matrix $\mathbf{Z}_k^T$ (Figure 1). Observe that when $\mathbf{B}$ can be written this way, all of its columns are linear combinations of the k columns of $\mathbf{Y}_k$, and all of its rows are linear combinations of the k rows of $\mathbf{Z}_k^T$. (It's also true that a rank-k matrix in the first two senses can always be written this way; see your linear algebra course for a proof.)

The primary goal of this lecture is to identify the "best" way to approximate a given matrix $\mathbf{A}$ with a rank-k matrix, for a target rank k. Why might you want to do this?

1. *Compression.* A low-rank approximation provides a (lossy) compressed version of the matrix. The original matrix $\mathbf{A}$ is described by nd numbers, while describing $\mathbf{Y}_k$ and $\mathbf{Z}_k^T$ requires only $k(n+d)$ numbers. When k is small relative to n and d, replacing the

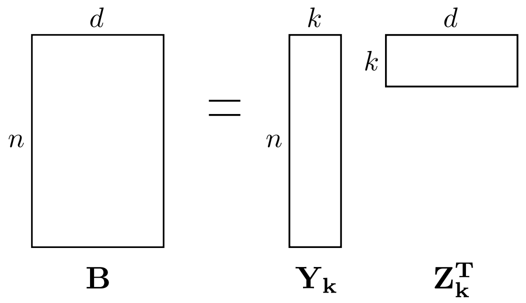

Figure 1: Any matrix $\mathbf{B}$ of rank k can be decomposed into a long and skinny matrix times a short and long one.

product of n and d by their sum is a big win. (With an image, n and d are typically in the 100s. In other applications, n and d might well be in the tens of thousands or more.)

2. *De-noising.* If $\mathbf{A}$ is a noisy version of some "ground truth" signal that is approximately low-rank, then passing to a low-rank approximation of the raw data $\mathbf{A}$ might throw out lots of noise and little signal, resulting in a matrix that is actually more informative than the original.

2 Low-Rank Approximations from PCA

The techniques covered last week can be used to produce low-rank matrix approximations. Recall the silly example at the beginning of Lecture #7, with a data set of n d-dimensional vectors $\mathbf{x}_i$ that turn out to all be multiples of each other. The corresponding matrix $\mathbf{A}$, with one $\mathbf{x}_i$ per row, has rank 1. The factorization in Figure 1 just involves taking $\mathbf{Z}_k^T$ to be the first vector (say), and $\mathbf{Y}_k$ a list describing what multiple of this vector each other vector is. That is, the data set can be re-represented, with no loss, by a single d-dimensional vector and one scalar per data point. When the data points are approximately multiples of the same vector, they can still be described with high accuracy using such a re-representation. More generally, for a target rank k, we can ask about how to best approximate a data set as a linear combinations of a set of k vectors.

Recall that principal components analysis (PCA) proposes a solution for choosing the k vectors that "best" represent a data set, namely the eigenvectors of the covariance matrix $\mathbf{A}^\mathbf{T}\mathbf{A}$. In more detail, here's how we'd use PCA techniques to product a rank-k approximation to a matrix $\mathbf{A}$:

1. Preprocess $\mathbf{A}$ so that the rows sum to the all-zero vector and, optionally, normalize each column (like last week).

2. Form the covariance matrix $\mathbf{A}^\mathbf{T}\mathbf{A}$.

3. In the notation of Figure 1, take the k rows of $\mathbf{Z}_k^T$ to be the top k principal components of $\mathbf{A}$ — the k eigenvectors $\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_k$ of $\mathbf{A^TA}$ that have the largest eigenvalues. (These can be computed using the Power Iteration method from last lecture, or other methods discussed below.)

4. For $i = 1, 2, \ldots, n$, the ith row of the matrix $\mathbf{Y}_k$ is defined as the projections $(\langle \mathbf{x}_i, \mathbf{w}_1 \rangle, \ldots, \langle \mathbf{x}_i, \mathbf{x}_k \rangle)$ of $\mathbf{x}_i$ onto the vectors $\mathbf{w}_1, \ldots, \mathbf{w}_k$. This is the best approximation, in terms of Euclidean distance from $\mathbf{x}_i$, of $\mathbf{x}_i$ as a linear combination of $\mathbf{w}_1, \ldots, \mathbf{w}_k$.[1]

The above four steps certainly produce a matrix

$$\mathbf{Y}_k \cdot \mathbf{Z}_k^T \tag{1}$$

that has rank only k. But how well does it approximate the original matrix $\mathbf{A}$? Also, it's unsatisfying and possibly wasteful to do so many computations involving the covariance matrix $\mathbf{A^TA}$, when all we really care about is the original matrix $\mathbf{A}$. Is there a better way? Next we discuss a fundamental matrix operation that provides answers to both of these questions.

3 The Singular Value Decomposition (SVD)

3.1 Definitions

We'll start with the formal definitions, and then discuss interpretations, applications, and connections to concepts in previous lectures. A *singular value decomposition (SVD)* of an $n \times d$ matrix $\mathbf{A}$ expresses the matrix as the product of three "simple" matrices:

$$\mathbf{A} = \mathbf{USV}^T, \tag{2}$$

where:

1. $\mathbf{U}$ is an $n \times n$ orthogonal matrix;[2]

2. $\mathbf{V}$ is a $d \times d$ orthogonal matrix;

3. $\mathbf{S}$ is an $n \times d$ diagonal matrix with nonnegative entries, and with the diagonal entries sorted from high to low (as one goes "northwest" to "southeast")."[3]

[1]For example, with $k = 2$, these values $(\langle \mathbf{x}_i, \mathbf{w}_1 \rangle, \langle \mathbf{x}_i, \mathbf{w}_2 \rangle)$ are the values that you plotted in Mini-Project #4.

[2]Recall that a matrix is *orthogonal* if its columns (or equivalently, its rows) are orthonormal vectors, meaning they all have norm 1 and the inner product of any distinct pair of them is 0.

[3]When we say that a (not necessarily square) matrix is diagonal, we mean what you'd think: only the entries of the form (i, i) are allowed to be non-zero.

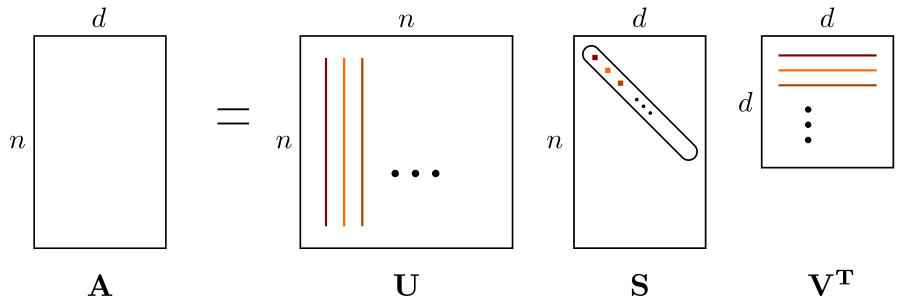

Figure 2: The singular value decomposition (SVD). Each singular value in $\mathbf{S}$ has an associated left singular vector in $\mathbf{U}$, and right singular vector in $\mathbf{V}$.

Note that in contrast to the decompositions discussed last week, the orthogonal matrices $\mathbf{U}$ and $\mathbf{V}$ are *not* the same — since $\mathbf{A}$ need not be square, $\mathbf{U}$ and $\mathbf{V}$ need not even have the same dimensions.[4]

The columns of $\mathbf{U}$ are *left singular vectors* of $\mathbf{A}$. The columns of $\mathbf{V}$ (that is, the rows of $\mathbf{V}^T$) are *right singular vectors* of $\mathbf{A}$. The entries of $\mathbf{S}$ are the *singular values* of $\mathbf{A}$. Thus with each singular vector (left or right) there is an associated singular value. The "first" or "top" singular vector refers to one associated with the largest singular value, and so on. See Figure 2.

Every matrix $\mathbf{A}$ has a SVD. The proof is not deep, but is better covered in a linear algebra course than here. Geometrically, this fact is kind of amazing: every matrix $\mathbf{A}$, no matter how weird, is only performing a rotation (multiplication by $\mathbf{V}^T$), scaling plus adding or deleting dimensions (multiplication by $\mathbf{S}$), followed by a rotation in the range (multiplication by $\mathbf{U}$). Along the lines of last lecture's discussion, the SVD is "more or less unique." The singular values of a matrix are unique. When a singular value appears multiple times, the subspaces spanned by the corresponding left and right singular vectors are uniquely defined, but arbitrary orthonormal bases can be chosen for each.[5]

3.2 PCA Reduces to SVD

There is an interesting relationship between the SVD and the decompositions we discussed last week. Recall in previous lectures we used the fact that $\mathbf{A^TA}$, as a symmetric $d \times d$ matrix, can be written as $\mathbf{A^TA} = \mathbf{QDQ}^T$, where $\mathbf{Q}$ is a $d \times d$ orthogonal matrix and $\mathbf{D}$ is a $d \times d$ diagonal matrix.[6] Consider the SVD $\mathbf{A} = \mathbf{USV}^T$ and what its existence means for

[4]Even small numerical examples are tedious to do in detail — the orthogonality constraint on singular vectors ensures that most of the numbers are messy. The easiest way to get a feel for what SVDs look like is to feed a few small matrices into the SVD subroutine supported by your favorite environment (Matlab, python's numpy library, etc.).

[5]Also, one can always multiply the ith left and right singular vectors by -1 to get another SVD.

[6]Actually, last week we wrote $\mathbf{A^TA} = \mathbf{Q}^T\mathbf{DQ}$. It doesn't really matter, but writing $\mathbf{A^TA} = \mathbf{QDQ}^T$ is much more common, and we do this from now.

$\mathbf{A^TA}$:

$$\mathbf{A^TA} = (\mathbf{USV}^T)^T(\mathbf{USV}^T) = \mathbf{VS}^T\underbrace{\mathbf{U}^T\mathbf{U}}_{=I}\mathbf{SV}^T = \mathbf{VDV}^T, \tag{3}$$

where $\mathbf{D}$ is a diagonal matrix with diagonal entries equal to the squares of the diagonal entries of $\mathbf{S}$ (if $n < d$ then the remaining $d - n$ diagonal entries of $\mathbf{D}$ are 0).

Recall from last lecture that if you decompose $\mathbf{A^TA}$ as $\mathbf{QDQ}^T$, then the rows of $\mathbf{Q}^T$ are eigenvectors of $\mathbf{A^TA}$. The computation in (3) therefore shows that the rows of $\mathbf{V}^T$ are the eigenvectors of $\mathbf{A^TA}$. Thus, *the right singular vectors of* $\mathbf{A}$ *are the same as the eigenvectors of* $\mathbf{A^TA}$. Similarly, the eigenvalues of $\mathbf{A^TA}$ are the squares of the singular values of $\mathbf{A}$.

Thus *PCA reduces to computing the SVD of* $\mathbf{A}$ *(without forming* $\mathbf{A^TA}$*)*. Recall that the output of PCA, given a target k, is simply the top k eigenvectors of the covariance matrix $\mathbf{A^TA}$. The SVD $\mathbf{USV}^T$ of $\mathbf{A}$ hands you these eigenvectors on a silver platter — they are simply the first k rows of $\mathbf{V}^T$. This is an alternative to the Power Iteration method discussed last lecture. So which is better? There is no clear answer; in many cases, either should work fine, and if performance is critical you'll want to experiment with both. Certainly the Power Iteration method, which finds the eigenvectors of $\mathbf{A^TA}$ one-by-one, looks like a good idea when you only want the top few eigenvectors. If you want many or all of them, then the SVD — which gives you all of the eigenvectors, whether you want them or not — is probably the first thing to try. The running time of typical SVD implementations is $O(n^2d)$ or $O(d^2n)$, whichever is smaller.[7] Such implementations have been heavily optimized in most of the standard libraries.

3.3 More on PCA vs. SVD

PCA and SVD are closely related, and in data analysis circles you should be ready for the terms to be used almost interchangeably. There are differences, however. First, PCA refers to data analysis technique, while the SVD is a general operation defined on all matrices. For example, it doesn't really make sense to talk about "applying PCA" to a matrix $\mathbf{A}$ unless the rows of $\mathbf{A}$ have clear semantics — typically, as data points $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in $\mathbb{R}^d$. By contrast, the SVD (2) is well defined for every matrix $\mathbf{A}$, whatever the semantics for $\mathbf{A}$. In the particular case where $\mathbf{A}$ is a matrix where the rows represent data points, the SVD can be interpreted as performing the calculations required by PCA.

We can also make more of an "apples vs. apples" comparison in the following way. Let's define the "PCA operation" as taking an $n \times d$ matrix as input, and possibly a parameter k, and outputting all (or the top k) eigenvectors of the covariance matrix $\mathbf{A^TA}$. The "SVD operation" takes as input an $n \times d$ matrix $\mathbf{A}$ and outputs $\mathbf{U}$, $\mathbf{S}$, and $\mathbf{V}^T$, where the rows of $\mathbf{V}^T$ are the eigenvectors of $\mathbf{A^TA}$. Thus *the SVD gives strictly more information than PCA*, namely the matrix $\mathbf{U}$.

Is the additional information $\mathbf{U}$ provided by SVD useful? In applications where you want to understand the *column* structure of $\mathbf{A}$, in addition to the row structure, the answer is

[7]We won't discuss how this is done, instead taking the SVD as a readily available "black box." Implementation details are covered in any course on numerical analysis.

"yes." To see this, let's review some interpretations of the SVD (2). On the one hand, the decomposition expresses every row of $\mathbf{A}$ as a linear combinations of the rows of $\mathbf{V}^T$, with the rows of $\mathbf{US}$ providing the coefficients of these linear combinations. That is, we can interpret the rows of $\mathbf{A}$ in terms of the rows of $\mathbf{V}^T$, which is useful when the rows of $\mathbf{V}^T$ have interesting semantics. Analogously, the decomposition in (2) expresses the *columns* of $\mathbf{A}$ as linear combinations of the columns of $\mathbf{U}$, with the coefficients given by the columns of $\mathbf{SV}^T$. So when the columns of $\mathbf{U}$ are interpretable, the decomposition gives us a way to understand the columns of $\mathbf{A}$.

In some applications, we really only care about understanding the rows of $\mathbf{A}$, and the extra information $\mathbf{U}$ provided by the SVD over PCA is irrelevant. In other applications, both the rows and the columns of $\mathbf{A}$ are interesting in their own right. For example:

1. Suppose rows of $\mathbf{A}$ are indexed by customers, and the columns by products, with the matrix entries indicating who likes what. We are interested in understanding the rows, and in the best-case scenario, the right singular vectors (rows of $\mathbf{V}^T$) are interpretable as "customer types" or "canonical customers" and the SVD expresses each customer as a mixture of customer types. For example, perhaps one or both of your instructors can be understood simply as a mixture of a "math customer," a "music customer," and a "sports customer." In the ideal case, the left singular vectors (columns of $\mathbf{U}$) can be interpreted as "product types," where the "types" are the same as for customers, and the SVD expresses each product as a mixture of product types (the extent to which a product appeals to a "math customer," a "music customer," etc.).

2. Suppose the matrix represents data about drug interactions, with the rows of $\mathbf{A}$ indexed by proteins or pathways, and the columns by chemicals or drugs. We're interested in understanding both proteins and drugs in their own right, as mixtures of a small set of "basic types."

In the above two examples, what we really care about is the relationships between two groups of objects — customers and products, or proteins and drugs — the labeling of one group as the "rows" of a matrix and the other as the "columns" is arbitrary. In such cases, you should immediately think of the SVD as a potential tool for better understanding the data. When the columns of $\mathbf{A}$ are not interesting in their own right, PCA already provides the relevant information.

4 Low-Rank Approximations from the SVD

If we want to best approximate a matrix $\mathbf{A}$ by a rank-k matrix, how should we do it? The SVD gives an elegant and rigorously justified solution. Recall from Section 1 what it means for a matrix to have rank k — all of the rows are linear combinations of a set of merely k rows, and all of the columns are linear combinations of merely k columns. Thus choosing a rank-k matrix boils down to choosing sets of k vectors. What's a principled way to choose these? If only we had a representation of the data matrix $\mathbf{A}$ as linear combinations of sets

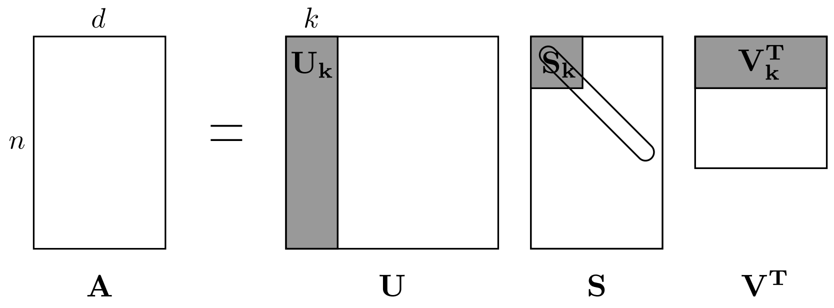

Figure 3: Low rank approximation via SVD. Recall that $\mathbf{S}$ is non-zero only on its diagonal, and the diagonal entries of S are sorted from high to low. Our low rank approximation is $\mathbf{A}_k = \mathbf{U}_k\mathbf{S}_k\mathbf{V}_k^T$.

of vectors, with these vectors ordered by "importance," then we could just keep the k "most important" vectors. But wait, the SVD gives us exactly such a representation!

Formally, given an $n \times d$ matrix $\mathbf{A}$ and a target rank $k \geq 1$, we produce a rank-k approximation of $\mathbf{A}$ as follows. See also Figure 3.

1. Compute the SVD $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^T$, where $\mathbf{U}$ is an $n \times n$ orthogonal matrix, $\mathbf{S}$ is a nonnegative $n \times d$ diagonal matrix with diagonal entries sorted from high to low, and $\mathbf{V}^T$ is a $d \times d$ orthogonal matrix.

2. Keep only the top k right singular vectors: set $\mathbf{V}_k^T$ equal to the first k rows of $\mathbf{V}^T$ (a $k \times d$ matrix).

3. Keep only the top k left singular vectors: set $\mathbf{U}_k$ equal to the first k columns of $\mathbf{U}$ (a $n \times k$ matrix).

4. Keep only the top k singular values: set $\mathbf{S}_k$ equal to the first k rows and columns of $\mathbf{S}$ (a $k \times k$ matrix), corresponding to the k largest singular values of $\mathbf{A}$.

The computed low-rank approximation is then

$$\mathbf{A}_k = \mathbf{U}_k\mathbf{S}_k\mathbf{V}_k^T. \tag{4}$$

Storing the matrices on the right-hand side of (4) takes $O(k(n+d))$ space, in contrast to the $O(nd)$ space required to store the original matrix $\mathbf{A}$. This is a big win when k is relatively small and n and d are relatively large (as in many applications).

In the matrix $\mathbf{A}_k$ defined in (4), all of the rows are linear combinations of the top k right singular vectors of $\mathbf{A}$ (with coefficients given by the rows of $\mathbf{U}_k\mathbf{S}_k$), and all of the columns are linear combinations of the top k left singular vectors of $\mathbf{A}$ (with coefficients given by the columns of $\mathbf{S}_k\mathbf{V}_k^T$). Thus $\mathbf{A}_k$ clearly has rank k. It is natural to interpret (4) as approximating the raw data $\mathbf{A}$ in terms of k "concepts" (e.g., "math," "music," and "sports"), where the singular values of $\mathbf{S}_k$ express the signal strengths of these concepts, the rows of $\mathbf{V}^T$ and columns of $\mathbf{U}$ express the "canonical row/column" associated with each concept (e.g., a customer that likes only music products, or a product liked only by music customers), and the rows of $\mathbf{U}$ (respectively, columns of $\mathbf{V}^T$) approximately express each row (respectively, column) of $\mathbf{A}$ as a linear combination (scaled by $\mathbf{S}_k$) of the "canonical rows" (respectively, canonical columns).

Conceptually, this method of producing a low-rank approximation is as clean as could be imagined: we re-represent $\mathbf{A}$ using the SVD, which provides a list of $\mathbf{A}$'s "ingredients," ordered by "importance," and we retain only the k most important ingredients. But is the result of this elegant computation any good? Also, how does it compare to our previous method of producing a low-rank approximation via PCA (Section 2)?

The first fact is that the two methods discussed for producing a low-rank approximation are exactly the same.[8]

Fact 4.1 *The matrix $\mathbf{A}_k$ defined in (1) and the matrix $\mathbf{A}_k$ defined in (4) are identical.*

We won't prove Fact 4.1, but pause to note its plausibility. Recall that in the PCA-based solution defined in (1), we defined $\mathbf{Z}_k^T$ to be the top k principal components of $\mathbf{A}$ — the first k eigenvectors of the covariance matrix $\mathbf{A^TA}$. As noted in Section 3.2, the right singular vector of $\mathbf{A}$ (i.e., the rows of $\mathbf{V}^T$) are also the eigenvectors of $\mathbf{A^TA}$. Thus, the matrices $\mathbf{Z}_k^T$ and $\mathbf{V}_k^T$ are identical, both equal to the top k eigenvectors of $\mathbf{A^TA}$/top k right singular vectors of $\mathbf{A}$. Given this, it is not surprising that the two definitions of $\mathbf{A}_k$ are the same: both the matrix $\mathbf{Y}_k$ in (1) and the matrix $\mathbf{U}_k\mathbf{S}_k$ in (4) are intuitively defining the linear combinations of the rows of $\mathbf{Z}_k^T$ and $\mathbf{V}_k^T$ that give the best approximation to $\mathbf{A}$. In the PCA-based solution in Section 2, this is explicitly how $\mathbf{Y}_k$ is defined; the SVD encodes the same linear combinations in the form $\mathbf{U}_k\mathbf{S}_k$.

Our second fact justifies our methods by stating that the low-rank approximations they produce are *optimal* in a natural sense. The guarantee is in terms of the "Frobenius norm" of a matrix $\mathbf{M}$, which just means applying the ℓ_2 norm to the matrix as if it were a vector: $\|\mathbf{M}\|_F = \sqrt{\sum_{i,j} m_{ij}^2}$.

Fact 4.2 *For every $n \times d$ matrix $\mathbf{A}$, rank target $k \geq 1$, and rank-k $n \times d$ matrix $\mathbf{B}$,*

$$\|\mathbf{A} - \mathbf{A}_k\|_F \leq \|\mathbf{A} - \mathbf{B}\|_F,$$

where $\mathbf{A}_k$ is the rank-k approximation (4) derived from the SVD of $\mathbf{A}$.

Intuitively, Fact 4.2 holds because: (i) minimizing the Frobenius norm $\|\mathbf{A}-\mathbf{B}\|_F$ is equivalent to minimizing the average (over i) of the squared Euclidean distances between the ith rows of $\mathbf{A}$ and $\mathbf{B}$; (ii) the SVD uses the same vectors to approximate the rows of $\mathbf{A}$ as PCA (the top eigenvectors of $\mathbf{A^TA}$/right singular vectors of $\mathbf{A}$); and (iii) PCA, by definition, chooses its k vectors to minimize the average squared Euclidean distance between the rows of $\mathbf{A}$ and the k-dimensional subspace of linear combinations of these vectors. The contribution of a

[8]We're assuming that identical preprocessing of $\mathbf{A}$, if any, is done in both cases.

row of $\mathbf{A} - \mathbf{A}_k$ to the Frobenius norm corresponds exactly to one of these squared Euclidean distances.

Remark 4.3 (How to Choose k) When producing a low-rank matrix approximation, we've been taking as a parameter the target rank k. But how should k be chosen? In a perfect world, the eigenvalues of $\mathbf{A}^T\mathbf{A}$/singular values of $\mathbf{A}$ give strong guidance: if the top few such values are big and the rest are small, then the obvious solution is to take k equal to the number of big values. In a less perfect world, one takes k as small as possible subject to obtaining a useful approximation — of course what "useful" means depends on the application. Rules of thumb often take the form: choose k such that the sum of the top k eigenvalues is at least c times as big as the sum of the other eigenvalues, where c is a domain-dependent constant (like 10, say).

Remark 4.4 (Lossy Compression via Truncated Decompositions) Using the SVD to produce low-rank matrix approximations introduces a useful paradigm for lossy compression that we'll exploit further in later lectures. The first step of the paradigm is to re-express the raw data exactly as a decomposition into several terms (as in (2)). The second step is to throw away all but the "most important" terms, yields an approximation of the original data. This paradigm works well when you can find a representation of the data such that most of the interesting information is concentrated in just a few components of the decomposition. The appropriate representation will depend on the data set — though some rules of thumb can be learned, as we'll discuss — and of course, messy enough data sets might not admit any nice representations at all.

5 Recovering Missing Entries via the SVD

This section briefly outlines how the SVD can be used to fill in missing entries of a matrix.[9] We'll also see some more sophisticated techniques for this problem in Week #7.

The input to the problem is an $n \times d$ matrix $\mathbf{A}$, except some (10%, say) of the entries are missing. You'd like to guess what the missing entries are. For a challenging example with lots of missing entries, if $\mathbf{A}$ is a matrix of movie ratings by Netflix customers — and of course, most people haven't rated most movies — you'd like to predict what a customer would rate a particular movie that he/she hasn't seen yet. Clearly, this is a relevant problem in the design of a recommendation system, among other applications.

This problem is clearly impossible unless we make assumptions about the "ground truth" matrix that we're supposed to recover — otherwise, the missing entries could be anything, and we have no information about them. A reasonable assumption that makes the problem more tractable is that the matrix to be recovered is well-approximated by a low-rank matrix. For intuition, think about the special case where you are given a rank-one matrix — so all rows are multiples of each other — except a few of the entries are missing. In this case, it is

[9]We didn't have time to cover this in lecture, but see Mini-Project #5.

clear that one can reconstruct the missing entries — if not too many are missing, there will be only one way of "filling in the blanks" that results in a rank-one matrix.

More generally, if there aren't too many missing entries, and if the matrix to be recovered is approximately low rank, then the following application of the SVD can yield a good guess as to the missing entries.

1. Fill in the missing entries with suitable default values to obtain a matrix $\hat{\mathbf{A}}$. Examples for default values include zero, the average value of an entry of the matrix, the average value of the row containing the entry, and the average value of the column containing the entry. The performance of the method improves with more accurate choices of default values.

2. Compute the best rank-k approximation to $\hat{\mathbf{A}}$. (The usual comments about how to choose k apply; see also Remark 4.3.)

References