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

Tim Roughgarden & Gregory Valiant*

April 26, 2021

1 What Are The Missing Entries?

Here's a quiz for you: consider the following 5×3 matrix, with 7 entries shown and 8 entries missing:

$$\begin{bmatrix} 7 & ? & ? \\ ? & 8 & ? \\ ? & 12 & 6 \\ ? & ? & 2 \\ 21 & 6 & ? \end{bmatrix}.$$

What are the missing entries?

This *matrix completion* problem seems a bit unfair, no? After all, each of the unknown entries could be anything, and there's no way to know what they are. But what if I told you the additional hint that the complete matrix has "nice structure?" This could mean many things, but for the example let's use an extreme assumption: that *all rows are multiples of each other.*

Now, it is possible to recover all of the missing entries! For example, if the third row is a multiple of the second one, then each entry in the latter must be $\frac{3}{2}$ times the corresponding entry in the former (because of the "12" and "8" in the middle column). Thus we can conclude that the third entry of the second row must be a "4." Similarly, the middle entry of the fourth row is a "4," the last entry of the final row is a "3," and so on. Here's the

completed matrix:

$$\begin{bmatrix} 7 & 2 & 1 \\ 28 & 8 & 4 \\ 42 & 12 & 6 \\ 14 & 4 & 2 \\ 21 & 6 & 3 \end{bmatrix}. \tag{1}$$

The point of the example is that when you know something about the "structure" of a partially known matrix, then sometimes it's possible to recover all of the "lost" information. The assumption that all rows are multiples of each other is pretty extreme — what would "matrix structure" mean more generally? One natural and useful definition, explained in the next section, is that of having low rank.

2 Matrix Rank

You have probably seen the notion of matrix rank in previous courses, but let's take a moment to page back in the relevant concepts.

Rank-0 Matrices. There is only one rank-zero matrix of a given size, namely the all-zero matrix.

Rank-1 Matrices. A rank-one matrix is precisely a non-zero matrix of the type assumed in the example above — all rows are (not necessarily integral) multiples of each other. In the example in (1), all columns are also multiples of each other; this is not an accident.

An equivalent definition of a rank-1 $m \times n$ matrix is as the *outer product* $\mathbf{u}\mathbf{v}^\top$ of an m-vector $\mathbf{u}$ and an n-vector $\mathbf{v}$:[1]

$$\mathbf{A} = \mathbf{u}\mathbf{v}^\top = \begin{bmatrix} — & u_1\mathbf{v}^\top & — \\ — & u_2\mathbf{v}^\top & — \\ & \vdots & \\ — & u_m\mathbf{v}^\top & — \end{bmatrix} = \begin{bmatrix} | & | & & | \\ v_1\mathbf{u} & v_2\mathbf{u} & \cdots & v_n\mathbf{u} \\ | & | & & | \end{bmatrix}.$$

Note that each row is a multiple of $\mathbf{v}^\top$, and each column is multiple of $\mathbf{u}$.

Rank-2 Matrices. A rank-two matrix is just a superposition (i.e., sum) of two rank-1 matrices:

$$\mathbf{A} = \mathbf{u}\mathbf{v}^\top + \mathbf{w}\mathbf{z}^\top = \begin{bmatrix} — & u_1\mathbf{v}^\top + w_1\mathbf{z}^\top & — \\ — & u_2\mathbf{v}^\top + w_2\mathbf{z}^\top & — \\ & \vdots & \\ — & u_m\mathbf{v}^\top + w_m\mathbf{z}^\top & — \end{bmatrix} = \begin{bmatrix} | & | \\ \mathbf{u} & \mathbf{w} \\ | & | \end{bmatrix} \cdot \begin{bmatrix} — & \mathbf{v}^\top & — \\ — & \mathbf{z}^\top & — \end{bmatrix}. \tag{2}$$

[1] Contrast this with the *inner* (a.k.a. *dot*) product $\mathbf{u}^\top\mathbf{v} = \sum_{i=1}^{n} u_i v_i$, which is only defined for two vectors of the same dimension and results in a scalar.

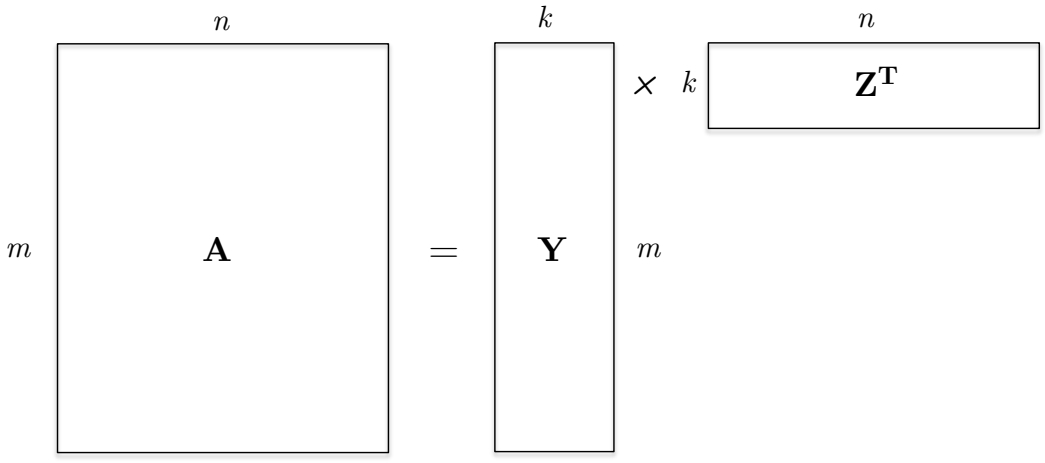

Figure 1: Any matrix $\mathbf{A}$ of rank k can be decomposed into a long and skinny matrix times a short and long one.

It's worth spending some time checking and internalizing the equalities in (2).

OK not quite: a rank-2 matrix is one that can be written as the sum of two rank-1 matrices and is not itself a rank-0 or rank-1 matrix.

Rank-k Matrices. The general definition of matrix rank should now be clear: a matrix $\mathbf{A}$ has rank k if it can be written as the sum of k rank-one matrices, and cannot be written as the sum of $k-1$ or fewer rank-one matrices.

Rephrased in terms of matrix multiplication, an equivalent definition is that $\mathbf{A}$ can written as, or "factored into," the product of a long and skinny $(m \times k)$ matrix $\mathbf{Y}$ and a short and long $(k \times n)$ matrix $\mathbf{Z}^\top$ (Figure 1). (And that $\mathbf{A}$ cannot be likewise factored into the product of $m \times (k-1)$ and $(k-1) \times n$ matrices.)

There are many equivalent definitions of the rank of a matrix $\mathbf{A}$. The following two conditions are equivalent to each other and to the definition above (any one of the three conditions implies the other two):

1. The largest linearly independent subset of columns of $\mathbf{A}$ has size k. That is, all n columns of $\mathbf{A}$ arise as linear combinations of only k of them.

2. The largest linearly independent subset of rows of $\mathbf{A}$ has size k. That is, all m rows of $\mathbf{A}$ arise as linear combinations of only k of them.

It is clear that the first condition implies the second and third: if $\mathbf{A} = \mathbf{Y}\mathbf{Z}^\top$, then all of $\mathbf{A}$'s columns are linear combinations of the k columns of $\mathbf{Y}$, and all of $\mathbf{A}$'s rows are linear combinations of the k rows of $\mathbf{Z}^\top$. See your linear algebra course for proofs of the other implications (they are not difficult).

3 Low-Rank Matrix Approximations: Motivation

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. Such a matrix is called a *low-rank approximation*. 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 mn numbers, while describing $\mathbf{Y}$ and $\mathbf{Z}^\top$ requires only $k(m+n)$ numbers. When k is small relative to m and n, replacing the product of m and n by their sum is a big win. For example, when $\mathbf{A}$ represents an image (with entries = pixel intensities), m and n are typically in the 100s. In other applications, m and n might well be in the tens of thousands or more. With images, a modest value of k (say 100 or 150) is usually enough to achieve approximations that look a lot like the original image.

   Thus low-rank approximations are a matrix analog to notions of dimensionality reduction for vectors (Lectures #4 and #7).

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.

3. *Matrix completion.* Low-rank approximations offer a first-cut approach to the matrix completion problem introduced in Section 1. (We'll see more advanced approaches in Week 9.) Given a matrix $\mathbf{A}$ with missing entries, the first step is to obtain a full matrix $\hat{\mathbf{A}}$ by filling in the missing entries with "default" values. Exactly what these default values should be requires trial and error, and the success of the method is often highly dependent on this choice. Natural things to try for default values include: 0, the average of the known entries in the same column, the average of the known entries in the same row, and the average of the known entries of the matrix. The second step is to compute the best rank-k approximation to $\hat{\mathbf{A}}$. This approach works reasonably well when the unknown matrix is close to a rank-k matrix and there are not too many missing entries.

Our high-level plan for computing a rank-k approximation of a matrix $\mathbf{A}$ is: (i) express $\mathbf{A}$ as a list of its ingredients, ordered by "importance;" (ii) keep only the k most important ingredients. The non-trivial step (i) is made easy by the singular value decomposition, a general matrix operation discussed in the next section.

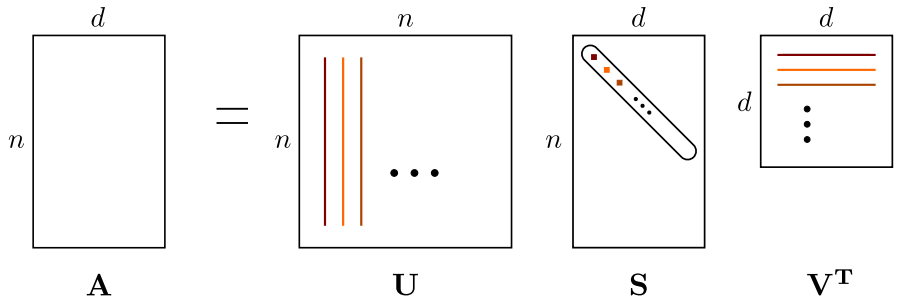

Figure 2: The singular value decomposition (SVD). Each singular value in $\mathbf{S}$ has an associated left singular vector in $\mathbf{U}$, and right singular vector in $\mathbf{V}$.

4 The Singular Value Decomposition (SVD)

4.1 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 $m \times n$ matrix $\mathbf{A}$ expresses the matrix as the product of three "simple" matrices:

$$\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^\top, \tag{3}$$

where:

1. $\mathbf{U}$ is an $m \times m$ orthogonal matrix;[2]
2. $\mathbf{V}$ is an $n \times n$ orthogonal matrix;
3. $\mathbf{S}$ is an $m \times n$ diagonal matrix with nonnegative entries, and with the diagonal entries sorted from high to low (as one goes "northwest" to "southeast")."[3]

Note that in contrast to the decomposition discussed in Lecture #8 ($\mathbf{A} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top$ when $\mathbf{A}$ has the form $\mathbf{X}^\top\mathbf{X}$), 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 called the *left singular vectors* of $\mathbf{A}$ (these are m-vectors). The columns of $\mathbf{V}$ (that is, the rows of $\mathbf{V}^\top$) are the *right singular vectors* of $\mathbf{A}$ (these are n-vectors). The entries of $\mathbf{S}$ are the *singular values* of $\mathbf{A}$. Thus with each singular vector (left

[2]Recall from last lecture 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.

[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.).

or right) there is an associated singular value. The "first" or "top" singular vector refers to the one associated with the largest singular value, and so on. See Figure 2.

To better see how the SVD expresses $\mathbf{A}$ as a "list of its ingredients," check that the factorization $\mathbf{A} = \mathbf{U}\mathbf{S}\mathbf{V}^\top$ is equivalent to the expression

$$\mathbf{A} = \sum_{i=1}^{\min\{m,n\}} s_i \cdot \mathbf{u}_i \mathbf{v}_i^\top, \tag{4}$$

where s_i is the ith singular value and $\mathbf{u}_i, \mathbf{v}_i$ are the corresponding left and right singular vectors. That is, the SVD expresses $\mathbf{A}$ as a nonnegative linear combination of $\min\{m, n\}$ rank-1 matrices, with the singular values providing the multipliers and the outer products of the left and right singular vectors providing the rank-1 matrices.

Every matrix $\mathbf{A}$ has a SVD. The proof is not deep, but is better covered in a linear algebra course than here. Geometrically, thinking of an $m \times n$ matrix as a mapping from $\mathbb{R}^n$ to $\mathbb{R}^m$, this fact is kind of amazing: every matrix $\mathbf{A}$, no matter how weird, is only performing a rotation in the domain (multiplication by $\mathbf{V}^\top$), followed by scaling plus adding or deleting dimensions (multiplication by $\mathbf{S}$) as needed, 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]

There are pretty good algorithms for computing the SVD of a matrix; details are covered in any numerical analysis course. It is unlikely that you will ever need to implement one of these yourself. For example, in Matlab, you literally just write `[U,S,V] = svd(A)` to compute the SVD of $\mathbf{A}$. The running time of the algorithm is the smaller of $O(m^2n)$ and $O(n^2m)$, and the standard implementations of it have been heavily optimized. A typical laptop should have no trouble computing the SVD of a 5000×5000 dense matrix, but might take a bit of time for a 10000×10000 matrix. As we remark at the end of these notes, if you just want to compute the largest k singular values, and their associated singular vectors, this can be computed significantly faster, in time roughly $O(kmn)$. (This last comment is quite relevant to this week's miniproject.)

5 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? If only we had a representation of the data matrix $\mathbf{A}$ as a sum of several ingredients, with these ingredients ordered by "importance," then we could just keep the k "most important" ones. But wait, the SVD gives us exactly such a representation! Recalling that the SVD expresses a matrix $\mathbf{A}$ as a sum of rank-1 matrices (weighted by the corresponding singular values), a natural idea is to keep only the first k terms on the right-hand side of (4). That is, for $\mathbf{A}$ as

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

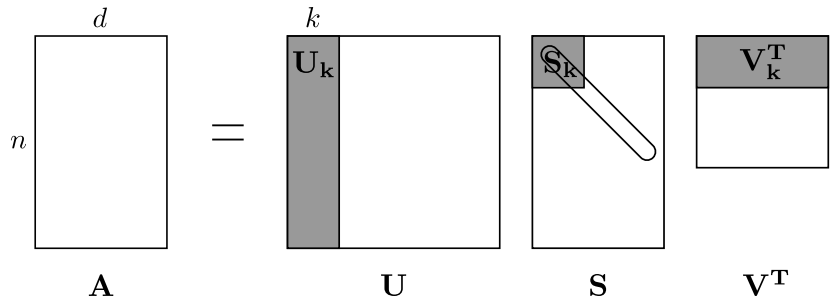

Figure 3: Low rank approximation via SVD. Recall that $\mathbf{S}$ is non-zero only on its diagonal, and the diagonal entries of S are sorted from high to low. Our low rank approximation is $\mathbf{A}_k = \mathbf{U}_k\mathbf{S}_k\mathbf{V}_k^\top$.

in (4) and a target rank k, the proposed rank-k approximation is

$$\hat{\mathbf{A}} = \sum_{i=1}^{k} s_i \cdot \mathbf{u}_i\mathbf{v}_i^\top, \tag{5}$$

where as usual we assume that the singular values have been sorted ($s_1 \geq s_2 \geq \cdots s_{\min\{m,n\}} \geq 0$), and $\mathbf{u}_i$ and $\mathbf{v}_i$ denote the ith left and right singular vectors. As the sum of k rank-1 matrices, $\hat{\mathbf{A}}$ clearly has rank (at most) k.

Here is an equivalent way to think about the proposed rank-k approximation (see also Figure 3).

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

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

3. Keep only the top k left singular vectors: set $\mathbf{U}_k$ equal to the first k columns of $\mathbf{U}$ (an $m \times k$ 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 $\mathbf{A}$.

5. The rank-k approximation is then

$$\mathbf{A}_k = \mathbf{U}_k\mathbf{S}_k\mathbf{V}_k^\top. \tag{6}$$

Storing the matrices on the right-hand side of (6) takes $O(k(m+n))$ space, in contrast to the $O(mn)$ space required to store the original matrix $\mathbf{A}$. This is a big win when k is relatively small and m and n are relatively large (as in many applications).

It is natural to interpret (6) 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 $\mathbf{V}^\top$ 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 $\mathbf{V}^\top$) 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?

The next fact justifies this approach: this low-rank approximation is *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 5.1 *For every $m \times n$ matrix $\mathbf{A}$, rank target $k \geq 1$, and rank-k $m \times n$ matrix $\mathbf{B}$,*

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

where $\mathbf{A}_k$ is the rank-k approximation (6) *derived from the SVD of $\mathbf{A}$.*

We won't prove Fact 5.1 formally, but see Section 8 for a plausibility argument based on the properties we've already established about the closely related PCA method.

Remark 5.2 (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 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 singular values is at least c times as big as the sum of the other singular values, where c is a domain-dependent constant (like 10, say).

Remark 5.3 (Lossy Compression via Truncated Decompositions) Using the SVD to produce low-rank matrix approximations is another example of a useful paradigm for lossy compression. The first step of the paradigm is to re-express the raw data exactly as a decomposition into several terms (as in (3)). The second step is to throw away all but the "most important" terms, yielding 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 there are a few rules of thumb, as we'll discuss — and of course, messy enough data sets might not admit any nice representations at all.

6 PCA Reduces to SVD

There is an interesting relationship between the SVD and the decompositions we discussed last week. Recall in the last lecture we used the fact that $\mathbf{X}^\top\mathbf{X}$, as a symmetric $n \times n$ matrix, can be written as $\mathbf{X}^\top\mathbf{X} = \mathbf{Q}\mathbf{D}\mathbf{Q}^\top$, where $\mathbf{Q}$ is an $n \times n$ orthogonal matrix and $\mathbf{D}$ is an $n \times n$ diagonal matrix. (Here $\mathbf{X}$ is the data matrix, with each of the m rows representing a data point in $\mathbb{R}^n$.) Consider the SVD $\mathbf{X} = \mathbf{U}\mathbf{S}\mathbf{V}^\top$ and what its existence means for $\mathbf{X}^\top\mathbf{X}$:

$$\mathbf{X}^\top\mathbf{X} = (\mathbf{U}\mathbf{S}\mathbf{V}^\top)^\top(\mathbf{U}\mathbf{S}\mathbf{V}^\top) = \mathbf{V}\mathbf{S}^\top \underbrace{\mathbf{U}^\top\mathbf{U}}_{=I}\mathbf{S}\mathbf{V}^\top = \mathbf{V}\mathbf{D}\mathbf{V}^\top, \tag{7}$$

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

Recall from last lecture that if you decompose $\mathbf{X}^\top\mathbf{X}$ as $\mathbf{Q}\mathbf{D}\mathbf{Q}^\top$, then the rows of $\mathbf{Q}^\top$ are the eigenvectors of $\mathbf{X}^\top\mathbf{X}$. The computation in (7) therefore shows that the rows of $\mathbf{V}^\top$ are the eigenvectors of $\mathbf{X}^\top\mathbf{X}$. Thus, *the right singular vectors of* $\mathbf{X}$ *are the same as the eigenvectors of* $\mathbf{X}^\top\mathbf{X}$. Similarly, the eigenvalues of $\mathbf{X}^\top\mathbf{X}$ are the squares of the singular values of $\mathbf{X}$.

Thus *principal components analysis (PCA) reduces to computing the SVD of* $\mathbf{A}$ *(without forming* $\mathbf{X}^\top\mathbf{X}$*).* Recall that the output of PCA, given a target k, is simply the top k eigenvectors of the covariance matrix $\mathbf{X}^\top\mathbf{X}$. The SVD $\mathbf{U}\mathbf{S}\mathbf{V}^\top$ of $\mathbf{X}$ hands you these eigenvectors on a silver platter — they are simply the first k rows of $\mathbf{V}^\top$. This is an alternative to the Power Iteration method discussed last lecture. So which method for computing eigenvectors 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 $\mathbf{X}^\top\mathbf{X}$ one-by-one, looks like a good idea when you only want the top few eigenvectors (as in our data visualization use cases). 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.

Now that we understand the close connection between the SVD and the PCA method, let's return to Fact 5.1, which states that the SVD-based rank-k approximation is optimal (with respect to the Frobenius norm). Intuitively, this fact 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 $\mathbf{A}^\top\mathbf{A}$/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 row of $\mathbf{A} - \mathbf{A}_k$ to the Frobenius norm corresponds exactly to one of these squared Euclidean distances.

7 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 a data analysis approach — a choice of how to define the "best" way to approximate a bunch of data points as linear combinations of a small set of vectors. Meanwhile, 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 $\mathbf{x}_1, \ldots, \mathbf{x}_m$ in $\mathbb{R}^n$. By contrast, the SVD (3) 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. (The SVD is also useful for many other computational tasks.)

We can also make more of an "apples vs. apples" comparison in the following way. Let's define the "PCA operation" as taking an $m \times n$ data matrix $\mathbf{X}$ as input, and possibly a parameter k, and outputting all (or the top k) eigenvectors of the covariance matrix $\mathbf{X}^\top\mathbf{X}$. The "SVD operation" takes as input an $m \times n$ matrix $\mathbf{X}$ and outputs $\mathbf{U}$, $\mathbf{S}$, and $\mathbf{V}^\top$, where the rows of $\mathbf{V}^\top$ are the eigenvectors of $\mathbf{X}^\top\mathbf{X}$. Thus *the SVD gives strictly more information than required by PCA*, namely the matrix $\mathbf{U}$.

Is the additional information $\mathbf{U}$ provided by the SVD useful? In applications where you want to understand the *column* structure of $\mathbf{X}$, in addition to the row structure, the answer is "yes." To see this, let's review some interpretations of the SVD (3). On the one hand, the decomposition expresses every row of $\mathbf{X}$ as a linear combinations of the rows of $\mathbf{V}^\top$, with the rows of $\mathbf{US}$ providing the coefficients of these linear combinations. That is, we can interpret the rows of $\mathbf{X}$ in terms of the rows of $\mathbf{V}^\top$, which is useful when the rows of $\mathbf{V}^\top$ have interesting semantics. Analogously, the decomposition in (3) expresses the *columns* of $\mathbf{X}$ as linear combinations of the columns of $\mathbf{U}$, with the coefficients given by the columns of $\mathbf{SV}^\top$. So when the columns of $\mathbf{U}$ are interpretable, the decomposition gives us a way to understand the columns of $\mathbf{X}$.

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

1. Suppose rows of $\mathbf{X}$ 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 $\mathbf{V}^\top$) are interpretable as "customer types" or "canonical customers" and the SVD expresses each customer as a mixture of customer types. For example, perhaps each student's purchasing history can be understood simply as a mixture of a "CS customer," a "music customer," and a "clothing 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 "CS customer," a "music customer," etc.).

2. Suppose the matrix represents data about drug interactions, with the rows of $\mathbf{X}$ 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 $\mathbf{X}$ are not interesting in their own right, PCA already provides the relevant information.

8 PCA-Based Low-Rank Approximations (Optional)

The techniques developed for PCA can also be used to produce low-rank matrix approximations, as follows.

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

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

3. In the notation of Figure 1, take the k rows of $\mathbf{Z}^\top$ to be the top k principal components of $\mathbf{A}$ — the k eigenvectors $\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_k$ of $\mathbf{A}^\top\mathbf{A}$ that have the largest eigenvalues.

4. For $i = 1, 2, \ldots, m$, the ith row of the matrix $\mathbf{Y}$ is defined as the projections $(\langle \mathbf{x}_i, \mathbf{v}_1\rangle, \ldots, \langle \mathbf{x}_i, \mathbf{v}_k\rangle)$ of $\mathbf{x}_i$ onto the vectors $\mathbf{v}_1, \ldots, \mathbf{v}_k$. This is the best approximation, in terms of Euclidean distance from $\mathbf{x}_i$, of $\mathbf{x}_i$ as a linear combination of $\mathbf{v}_1, \ldots, \mathbf{v}_k$.[6]

The above four steps certainly produce a matrix

$$\mathbf{Y} \cdot \mathbf{Z}^\top \tag{8}$$

that has rank at most k. How does it compare to the SVD-based low-rank approximation? In fact, it is exactly the same![7]

Fact 8.1 *The matrix $\mathbf{A}_k$ defined in* (8) *and the matrix $\mathbf{A}_k$ defined in* (6) *are identical.*

We won't prove Fact 8.1, but pause to note its plausibility. We defined $\mathbf{Z}^\top$ to be the top k principal components of $\mathbf{A}$ — the first k eigenvectors of the covariance matrix $\mathbf{A}^\top\mathbf{A}$. As noted in Section 6, the right singular vector of $\mathbf{A}$ (i.e., the rows of $\mathbf{V}^\top$) are also the eigenvectors of $\mathbf{A}^\top\mathbf{A}$. Thus, the matrices $\mathbf{Z}^\top$ and $\mathbf{V}_k^\top$ are identical, both equal to the top k eigenvectors of $\mathbf{A}^\top\mathbf{A}$/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 $\mathbf{Y}$ in (8) and the matrix $\mathbf{U}_k\mathbf{S}_k$ in (6) are intuitively defining the linear combinations of the rows of $\mathbf{Z}^\top$ and $\mathbf{V}_k^\top$ that give the best approximation to $\mathbf{A}$. In the PCA-based solution in Section 8, this is explicitly how $\mathbf{Y}$ is defined; the SVD encodes the same linear combinations in the form $\mathbf{U}_k\mathbf{S}_k$.

[6] 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 we plotted in the "map of Europe" example in Lecture #7.

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