Singular Value Decomposition - Algorithms and Applications

Group 4: Yuqin Mu, Zhengjie Xiong, Ivan Khurudzhi, Zhenya Liu

Spring 2021

1 Introduction

In class, we have seen diagonalization for a square matrix under certain conditions. However, for an arbitrary $m \times n$ matrix A, we also want to find out a way to decompose it, thus we introduce singular value decomposition (SVD).

1.1 Definition of singular value decomposition

Let A be an $m \times n$ matrix with rank r, then there exists a matrix factorization called singular value decomposition with the following form

$$A_{m\times n} = U_{m\times r}\Sigma_{r\times r}V^T_{r\times n} \tag{1}$$

where U is an $m \times r$ matrix with orthonormal columns, Σ is a diagnal matrix, V^T is an $r \times n$ matrix with orthonormal rolls. Note that for complex cases we apply V^* instead of V^T.

Next Step is to show the definition makes sense, i.e. we are going to find out U, Σ and V^T respectively. In our case, we pay attention to A^TA and AA^T.

1.1.1 Proposition 1

A^TA and AA^T have non-negative eigenvalues and they have same positive eigenvalues.

Proof. Assume v is an eigenvector of A^TA with eigenvalue λ, then $A^TAv = \lambda v$, thus $v^TA^TAv = \lambda v^Tv$, i.e. $\|Av\|^2 = \lambda\|v\|^2$, and $\lambda \geq 0$. Similarly, we can show AA^T has non-negative eigenvalues. Meanwhile, since $A^TAv = \lambda v$, then $AA^TAv = \lambda Av$. Let $Av = v'$, then $AA^TV' = \lambda v'$, i.e. λ is an eigenvalue of AA^T. Similarly we can show an eigenvalue of AA^T is also an eigenvalue of A^TA.

□

1.1.2 Proposition 2

Rows of V^T are orthonormal eigenvectors of A^TA; columns of U are orthonormal eigenvectors of AA^T. And entries of Σ are square roots of eigenvalues of A^TA.

Proof. Since $A = U\Sigma V^T$, then $A^T = V\Sigma^T U^T = V\Sigma U^T$ and $A^TA = V\Sigma U^T U\Sigma V^T = V\Sigma^2 V^T$. Then $A^TAV = V\Sigma^2 V^T V = V\Sigma^2$. Similarly, we can show $AA^TV = U\Sigma^2$. According to Proposition 1, entries of Σ are square roots of eigenvalues of A^TA and we call the entries singular values of A. □

1.1.3 Remark

We look at A^TA, AA^T again and find A^TA is an $n \times n$ matrix, A^TA is an $m \times m$ matrix. In particular, $n \geq r$, $m \geq r$, thus A^TA has $n - r$ many zero eigenvalues and AA^T has $m - r$ many zero eigenvalues. We can extend singular value decomposition of A in the following way,

$$A_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^T_{n\times n} \tag{2}$$

or

$$A_{m\times n} = \begin{pmatrix} | & | & & | \\ u_1 & u_2 & \cdots & u_m \\ | & | & & | \end{pmatrix}_{m\times m} \begin{pmatrix} \sigma_1 & & & \\ & \sigma_2 & & \\ & & \ddots & \\ & & & \sigma_r \\ & & & \end{pmatrix}_{m\times n} \begin{pmatrix} | & | & & | \\ v_1 & v_2 & \cdots & v_n \\ | & | & & | \end{pmatrix}^T_{n\times n} \tag{3}$$

where $\{u_1, \cdots, u_m\}$ are eigenvectors of U (including zero eigenvectors), $\{v_1, \cdots, v_n\}$ are eigenvectors of V (including zero eigenvectors) and $\{\sigma_1, \cdots, \sigma_r\}$ are non-zero singular values.

1.2 Simple example of SVD

Find SVD of matrix A

$$A = \begin{pmatrix} 3 & 2 & 2 \\ 2 & 3 & -2 \end{pmatrix}$$

Solution.

Step 1 Find A^TA and AA^T

$$A^TA = \begin{pmatrix} 3 & 2 \\ 2 & 3 \\ 2 & -2 \end{pmatrix} \begin{pmatrix} 3 & 2 & 2 \\ 2 & 3 & -2 \end{pmatrix} = \begin{pmatrix} 13 & 12 & 2 \\ 12 & 13 & -2 \\ 2 & -2 & 8 \end{pmatrix} \tag{4}$$

$$AA^T = \begin{pmatrix} 3 & 2 & 2 \\ 2 & 3 & -2 \end{pmatrix} \begin{pmatrix} 3 & 2 \\ 2 & 3 \\ 2 & -2 \end{pmatrix} = \begin{pmatrix} 17 & 8 \\ 8 & 17 \end{pmatrix} \tag{5}$$

Step 2 Find eigenvalues of A^TA and AA^T

By calculating characteristic polynomials, A^TA has three eigenvalues: $\lambda_1 = 25$,$\lambda_2 = 9$, $\lambda_3 = 0$. From Proposition 1, AA^T has two positive eigenvalues same with A^TA, and AA^T has at most two eigenvalues because it is a 2×2 matrix. $\lambda_1 = 25$,$\lambda_2 = 9$ are eigenvalues of AA^T. We get $\sigma_1 = \sqrt{25} = 5, \sigma_2 = \sqrt{9} = 3$, σ_1 and σ_2 are singular values of A, and

$$\Sigma = \begin{pmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \end{pmatrix} \tag{6}$$

Step 3 Find an orthonormal basis made of eigenvectors of A^TA in R^2 and a correspondent orthonormal basis made of eigenvectors of AA^T in R^3.

Note that both A^TA and AA^T are symmetric matrices, which imply eigenvectors from distinct eigenvalues are orthogonal, and this leads to the fact that every real symmetric matrix can have an orthonormal basis made of eigenvectors. A^TA has three distinct eigenvalues, so we only need to get one eigenvector from each eigenvalue, and we can form one orthogonal eigenvector basis. Solve linear equations, found $v_1 = \frac{1}{\sqrt{2}}\{1, 1, 0\}$ with λ_1, $v_2 = \frac{1}{\sqrt{18}}\{1, -1, 4\}$ with λ_2, $v_3 = \{\frac{2}{3}, -\frac{2}{3}, -\frac{1}{3}\}$ with λ_3. $\{v_1, v_2, v_3\}$ is the expected orthonormal and spectral basis of R^3. found $u_1 = \frac{1}{\sigma_1}Av_1 = \frac{1}{\sqrt{2}}\{1, 1\}$ is a normalized eigenvector with λ_1, and $u_2 = \frac{1}{\sigma_2}Av_2 = \frac{1}{\sqrt{2}}\{1, -1\}$ is a normalized eigenvector with λ_2. $\{u_1, u_2\}$ is the expected basis of R^2.

Step 4 Use vectors from two basis to construct matrices U and V.

From **Remark**, let u_1 and u_2 be columns of U, v_1 and v_2 and v_3 be columns of V, we get

$$U = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \tag{7}$$

$$V = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{18}} & \frac{2}{3} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{18}} & -\frac{2}{3} \\ 0 & \frac{4}{\sqrt{18}} & -\frac{1}{3} \end{pmatrix} \tag{8}$$

Here we finish SVD of A

$$A = U\Sigma V^T = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{18}} & \frac{2}{3} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{18}} & -\frac{2}{3} \\ 0 & \frac{4}{\sqrt{18}} & -\frac{1}{3} \end{pmatrix}^T$$

(9)

□

1.3 An Introduction to Golub-Reinsch Algorithm

The most well-known and widely used algorithm for computing the Singular Value Decomposition is the Golub-Reinsch algorithm [2]. In this paper, we will neglect some definitions and only introduce an abstract procedure to work with this algorithm.

For any $m \times n$ matrix A, assume $m \geqslant n$ (if $m < n$, calculate A^t). The whole algorithm consists of two phases

1.3.1 Reduction to bidiagnoal form

In the first phase constructs two finite sequences of Householder transformations,

$$P^{(k)}, \quad k = 1, 2, ..., n \tag{10}$$

and

$$Q^{(k)}, \quad k = 1, 2, ..., n-2 \tag{11}$$

such that

$$P^{(n)} \cdots P^{(1)} A Q^{(1)} \cdots Q^{(n-2)} = \overbrace{\begin{bmatrix} x & x & & 0 \\ & \ddots & \ddots & \\ 0 & & \ddots & \\ & & x & x \\ \hline & & 0 & \end{bmatrix}}^{n} \begin{matrix} \left.\vphantom{\begin{bmatrix} a \\ b \end{bmatrix}}\right\} n \\ \left.\vphantom{\begin{bmatrix} a \\ b \end{bmatrix}}\right\} (m-n) \end{matrix} = J^{(0)},$$

J_0 is an upper bidiagonal matrix.

Specifically, $P^{(t)}$ zeros out the subdiagnoal elements in column i and $Q^{(t)}$ zeros out the subdiagnoal elements in row j.

Note that all Householder transformations introdudced are orthogonal, the singular matrix of $J^{(0)}$ is the same as that of A. Thus, if

$$J^{(0)} = G\Sigma H^T \tag{12}$$

is the SVD of $J^{(0)}$, then

$$A = PG\Sigma H^T Q^T \tag{13}$$

so that

$$U = PG, \qquad V = QH \tag{14}$$

with

$$P = P^{(1)} \cdots P^{(n)}, \qquad Q = Q^{(1)} \cdots Q^{(n)} \tag{15}$$

1.3.2 SVD of the Bidiagonal Matrix by a "chasing" process

Define one iterative algorithm on $J^{(0)}$ so that

$$J^{(0)} \to J^{(1)} \to \cdots \to \Sigma \tag{16}$$

where

$$J^{(i+1)} = S^{(i)T} J^{(i)} T^{(i)} \tag{17}$$

and $S^{(i)}$ and $T^{(i)}$ are orthogonal. The martrices $T^{(i)}$ are chosen so that the sequence $M^{(i)} = J^{(i)T} J^{(i)}$ converges to a diagonal matrix while the matrices $S^{(i)}$ are chosen so that all $J^{(i)}$ are of the bidiagonal form.

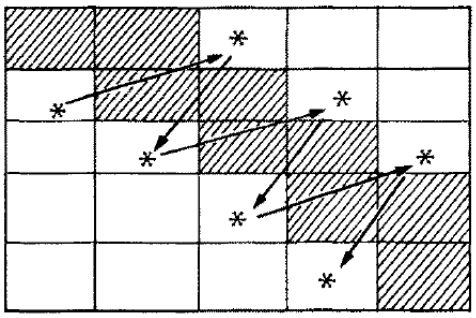
For notational convenience, we drop the suffix and use the notation

$$J \equiv J^{(i)}, \quad \bar{J} \equiv J^{(i+1)}, \quad S \equiv S^{(i)}, \quad T \equiv T^{(i)}, \quad M \equiv J^T J, \quad \bar{M} \equiv \bar{J}^T \bar{J} \tag{18}$$

The transition $J \to \bar{J}$ is achieved by application of Givens rotations to J alternately from the right and the left. Thus,

$$\bar{J} = \underbrace{S_n^T S_{n-1}^T \cdots S_2^T}_{S^T} J \underbrace{T_2 T_3 \cdots T_n}_{T} \tag{19}$$

T_2 is defined with a determined angle ϕ and can generate one non-zero entry J_{21}. S_2^T is defined to annihilate J_{21}, and generates an entry J_{13}. T_3 is defined to annihilate J_{13}, and generates an entry J_{32}. Inductively, S_2^T is defined to annihilate $J_{n,n-1}$, and generates nothing. The whole process can be shown in graph below



This process is frequently described as "chasing". Since $\bar{J}$ is still a bidiagonal matrix, $\bar{M} = \bar{J}^T\bar{J}$ is a tri-diagonal matrix just as M is. The angle ϕ of T_2 is chosen so that $M \to \bar{M}$ is a QR transformation. That means, for the output $\bar{J}$ of each loop of "chasing" process starting on $J^{(0)}$, calculated $\bar{M}$ will be the output of each loop of QR algorithm starting on $J^{(0)T}J^{(0)}$. QR algorithm on one symmetry matrix $J^{(0)T}J^{(0)}$ can finally converge to one diagonal matrix. Thus, after many loops we can get $\bar{M}$ as diagonal matrix, and $\bar{J}$ becomes Σ we expect to get. In the whole algorithm, we do not actually calculate any step of QR algorithm on J, but we use the connection with it to show our designed "chasing" algorithm can finally output an expected result. This is called "implicit QR algorithm".

2 Applications

2.1 Pseudoinverse

One important application of SVD is to solve systems of linear equations, *i.e* $Ax = b$. Generally we can use elementary method like Gaussian Elimination to completely check all possible solutions of this system. Specifically, if A is an invertible square, there is only one solution which can be quickly calculated by $x = A^{-1}b$. In machine learning, for some systems of linear equations, sometimes we only need to get one approximated solution which has a good fitting to our data or models. For a general $m \times n$ matrix A, we want to find a $n \times m$ matrix A^+ which acts like a invertible matrix, *i.e.* $A^+A \approx I_n$, then we can get one approximated solution $z = A^+Ab$. By using SVD, we can construct a "pseudoinverse" A^+ of A, which can help get one best-fit solution of the system.

2.1.1 Definition

Let A be a real $m \times n$ matrix of rank r, by *Singular Value Decomposition*, $A = U\Sigma V^T$ with nonzero singular values $\sigma_1 \geqslant \sigma_2 \geqslant \cdots \geqslant \sigma_r$, let Σ^+ be the $n \times m$ matrix defined by

$$\Sigma^+_{ij} = \begin{cases} \frac{1}{\sigma_i} & if\ i = j \leqslant r \\ 0 & otherwise \end{cases} \tag{20}$$

Then pseudoinverse of A is $A^+ = V\Sigma^+U^T$.

2.1.2 Proposition 3

Consider the system of real linear equations $Ax = b$, where A is an $m \times n$ matrix. If $z = A^{+}b$, then z has the following properties [8].

(a) If $Ax = b$ is consistent, then z is the unique solution to the system having minimal norm, and if y is any solution to the system, then $\|z\| \geqslant \|y\|$ with equality if and only if $z = y$.
(b) If $Ax = b$ is inconsistent, then z is the unique best approximation to a slution having minimum norm, *i.e* $\|Az - b\| \geqslant \|Ay - b\|$ for any $y \in R^n$, with equality if and only $Az = Ay$.

2.1.3 Example

Consider a linear system used with the same matrix A in the prior example

$$\begin{pmatrix} 3 & 2 & 2 \\ 2 & 3 & -2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tag{21}$$

This system is consistent with infinite solutions, and we can use the pseudoinverse A^{+} to find one solution z which has minimal norm.
Recall that

$$A = U\Sigma V^T = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \begin{pmatrix} 5 & 0 & 0 \\ 0 & 3 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{18}} & \frac{2}{3} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{18}} & -\frac{2}{3} \\ 0 & \frac{4}{\sqrt{18}} & -\frac{1}{3} \end{pmatrix}^T$$

(22)

Find Σ^{+}

$$\Sigma^{+} = \begin{pmatrix} \frac{1}{5} & 0 \\ 0 & \frac{1}{3} \\ 0 & 0 \end{pmatrix} \tag{23}$$

and get A^{+} and z

$$A^{+} = V\Sigma^{+}U^T = \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{18}} & \frac{2}{3} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{18}} & -\frac{2}{3} \\ 0 & \frac{4}{\sqrt{18}} & -\frac{1}{3} \end{pmatrix} \begin{pmatrix} \frac{1}{5} & 0 \\ 0 & \frac{1}{3} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} & -\frac{1}{\sqrt{2}} \end{pmatrix} \tag{24}$$

$$= \begin{pmatrix} 7/45 & 2/45 \\ 2/45 & 7/45 \\ 2/9 & -2/9 \end{pmatrix} \tag{25}$$

$$z = \Sigma^{+}b = \begin{pmatrix} 7/45 & 2/45 \\ 2/45 & 7/45 \\ 2/9 & -2/9 \end{pmatrix} \begin{pmatrix} 1 \\ 2 \end{pmatrix} = \begin{pmatrix} 11/45 \\ 16/45 \\ -2/9 \end{pmatrix} \tag{26}$$

now check the answer

$$\begin{pmatrix} 3 & 2 & 2 \\ 2 & 3 & -2 \end{pmatrix} \begin{pmatrix} 11/45 \\ 16/45 \\ -2/9 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \tag{27}$$

2.2 Machine Learning. Principal Component Analysis (PCA)

2.2.1 Definition of PCA[3]

Principal component analysis (PCA) is a standard tool in modern data analysis - in diverse fields from neuroscience to computer graphics - because it is a simple, non-parametric method for extracting relevant information from confusing data sets. With minimal effort PCA provides a roadmap for how to reduce a complex data set to a lower dimension to reveal the sometimes hidden, simplified structures that often underlie it.

2.2.2 Purpose of PCA

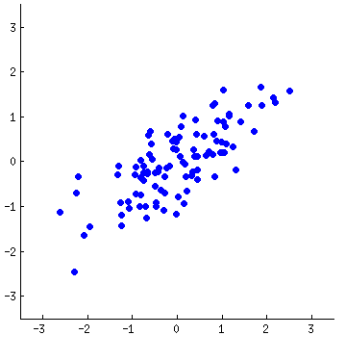
The main purpose of PCA is to reduce the complexity of data analysis as well as to identify the most meaningful basis to re-express the data to filter out the noise and reveal underlying patterns. SVD helps with extracting important data features and the correlations between them, which is very useful in high-dimensional data, in which finding patterns manually is nearly impossible. The main component of PCA is the reduction of the number of dimensions of the data without losing the accuracy of those data features. So, the goal of PCA is finding such an orthonormal basis for the data matrix, that that variance of the data projected onto the direction of the vectors is maximized.

2.2.3 Intuitive Explanation of PCA[1]

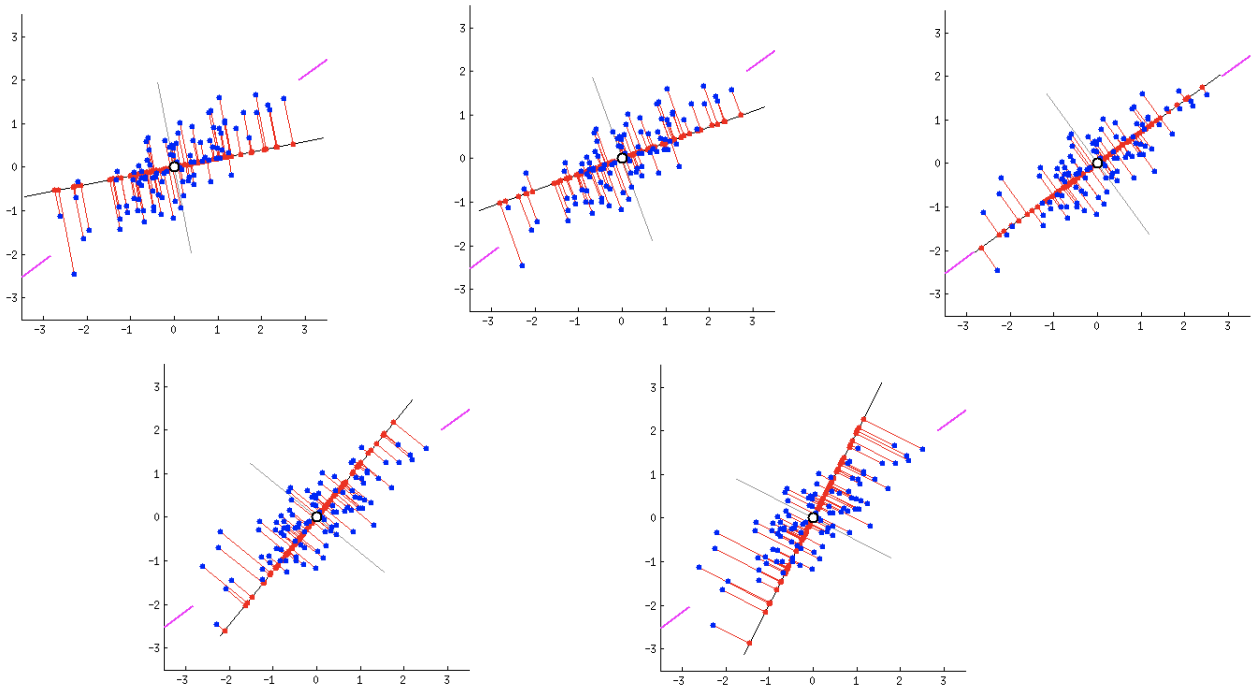
Let's say we have some number of milk bottles in the fridge. Each one of them has a number of properties they could be described by: color, fat-percentage, date, cow species, cow age, cow nutrition. What if we could reduce the number of these properties by constructing some new properties that would combine the characteristics of some original properties?

For example, we could build a production quality property of milk, that would combine the cow age and cow nutrition properties together into a new one with the use of some linear combination of these properties. That would reduce the dimensionality of our data, helping us find patterns, as well as reduce the complexity of our data analysis. So with the use of PCA we can find the best way to combine certain properties into new properties, thus reducing the dimensionality of our data. PCA should look for properties that would have high variation across our milk bottles. If we have some new property of our milk bottles that is the same for all of them, this whole operation would be useless.

Let's take two properties of our milk bottles, let's say color and date, and let's say two of these properties could be numerically expressed, and we could have a scatter plot for these properties that would look like this.



We could construct a new property by drawing a line through the center of this cluster and projecting all the points onto this line. Now if we spin this line around, how do we find the best position for it?



As we said before, we need high variation across our milk bottles, so we are looking for such a position that the spread of the red dots on the line is maximized. This happens when the line coincides with the two little lines on the sides. In this position of our line, we have a new property of our milk bottles. This property is called our first principal component.

2.2.4 SVD in PCA[9]

These vectors that determine the direction of the variance are the eigenvectors of the covariance matrix. The drawbacks of using the eigendecomposition for the covariance matrix are the computational complexity for larger datasets as well as the round-off errors (difference

between computation of the result using exact arithmetic and rounded arithmetic). So let out data matrix A be of $n \times m$ size, where n is the number of samples and m is the number of features (e.g. a matrix A representing the weight and height measurements collected from a sample of 100 people would be a 100×2 matrix). If we perform SVD on A, we obtain

$$A_{n\times m} = U_{n\times r}\Sigma_{r\times r}V^T_{r\times m} \tag{28}$$

[Note: our matrix A is centered, i.e. we have extracted the mean of the feature from each entry corresponding to this feature.]

The formula for finding the sample covariance matrix C is

$$C = \frac{A^T A}{n-1} \tag{29}$$

Then the covariance matrix for our matrix A is

$$C = \frac{A^T A}{n-1} = \frac{V\Sigma U^T U\Sigma V^T}{n-1} \tag{30}$$

Since U is orthogonal, i.e.

$$I = U^T U = UU^T \tag{31}$$

then

$$C = \frac{V\Sigma U^T U\Sigma V^T}{n-1} = \frac{V\Sigma^2 V^T}{n-1} \tag{32}$$

From here we can see that the right singular vectors are principal directions, and that the eigenvalues of the covariance matrix λ_i can be expressed as

$$\lambda_i = \frac{s_i^2}{n-1} \tag{33}$$

To get the set of our principle components (let's call this set z),

$$z = AV \tag{34}$$

Which with our derivation of A using SVD can be expressed as

$$z = AV = U\Sigma V^T V \tag{35}$$

Since V is also orthogonal, i.e.

$$I = V^T V = VV^T \tag{36}$$

then

$$z = U\Sigma V^T V = U\Sigma \tag{37}$$

Thus, we can see that PCA uses SVD to get principal components by composing matrices U and Σ.

Let's say we have a new user David with ratings $R = [4, 3, 5, 0, 4]$ for this set of movies. To place David in the space of genres, we would:

$$VR = \begin{bmatrix} 0.58 & 0.58 & 0.58 & 0 & 0 \\ 0 & 0 & 0 & 0.71 & 0.71 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \\ 5 \\ 0 \\ 4 \end{bmatrix} = \begin{bmatrix} 6.96 \\ 2.84 \end{bmatrix} \tag{38}$$

That would make David more interested in sci-fi movies than in romance movies, but not by a large margin.

We could also project the resulting VR onto the movie space by computing:

$$(VR)^T V^T = \begin{bmatrix} 6.96 & 2.84 \end{bmatrix} \begin{bmatrix} 0.58 & 0.58 & 0.58 & 0 & 0 \\ 0 & 0 & 0 & 0.71 & 0.71 \end{bmatrix} \tag{39}$$

$$(VR)^T V^T = \begin{bmatrix} 4.0368 & 4.0368 & 4.0368 & 2.0164 & 2.0164 \end{bmatrix} \tag{40}$$

2.2.5 Recommendation Systems[4]

Recommendation Systems is another field where SVD is applied. Recommendation Systems usually use two types of algorithms: collaborative filtering and content-based filtering. Content-based filtering only uses the data from the observed user to recommend items for them. SVD is used in Collaborative Filtering, in which the recommendation of items for the user is based on the analyzed data from different users.

SVD decomposes the matrix that contains the data about the users and items into three matrix that could be very intuitively interpreted. The matrix U might be interpreted as a matrix representing relationships between users and factors, where factors could be genres, types of items, etc. The Σ matrix can be interpreted as a matrix representing the strength, or importance, popularity of each factor. And the matrix V can be interpreted as a matrix showing the relationships between the items and the factors.

Most interestingly, SVD was a very big and important part of the Netflix competition [7]. As a matter of fact, one of the winners of the competition, a coder named Simon Funk improved the Netflix algorithm by using SVD in combination with Stochastic Gradient Descent. To be able to explain in more detail how exactly Funk achieved this we have to start by understanding what Stochastic Gradient Descent.

Gradient Descent is a very important algorithm used in many aspects of machine learning, when we want to minimize the loss function (or the error function). In gradient descent we differentiate the loss function with respect to certain parameters to decide what parameters to use in our next step to move towards the minimum values of the loss function. Gradient Descent is not always optimal as it does have a tendency to get stuck in the local optima.

This is where Stochastic Gradient Descent comes into play. It follows pretty much the same steps as Gradient Descent, but it decides the starting points randomly and restarts the algorithm whenever the function gets stuck in local optima.

What Simon Funk did is reverse SVD by choosing certain matrices such that they approximate the original user data matrices most closely. By doing that again and again and using Stochastic Gradient Descent to minimize the loss function so that the approximation is the best, Funk succeeded in improving the Netflix algorithm.

2.3 Low-rank approximation

2.3.1 Definition[11]

Given:
-Structure specification $S : \mathbb{R}^{n_p} \to \mathbb{R}^{m\times n}$;
-vector of structure parameters $p \in \mathbb{R}^{n_p}$;
-norm $\|\cdot\|$;
-rank r;
we want to minimize over $\hat{p}$ subject to $rank(S(\hat{p})) \leq r$.

2.3.2 Basic low-rank approximation problem

Given arbitrary $m \times n$ matrix A with rank r, and an integer $k > 0$, we want to find a matrix A_k whose rank is at most k, such that $\|A - A_k\|_F$ is minimized. Note that we are using the Forbenius norm, where

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} a_{ij}^2} \tag{41}$$

Notice when $k = r$, then $A_k = A$, and $\|A - A_k\|_F = 0$; when $k < r$, the result is called low-rank approximation which is given by the following theorem.

2.3.3 Eckart–Young–Mirsky Theorem

Let

$$A = U\Sigma V^T \in \mathbb{R}^{m\times n}, m \geq n \tag{42}$$

be the singular value decomposition of A with partition of U, $\Sigma = diag\{\sigma_1, \cdots, \sigma_n\}$, V follows:

$$U = \begin{pmatrix} U_k & U_{m-k} \end{pmatrix}, \Sigma = \begin{pmatrix} \Sigma_k & \\ & \Sigma_{rest} \end{pmatrix}, V = \begin{pmatrix} V_k & V_{n-k} \end{pmatrix} \tag{43}$$

where U_1 is $m \times k$, Σ_1 is $k \times k$, V_1 is $n \times k$, then define

$$A_k = U_k\Sigma_k V_k^T \tag{44}$$

such that

$$\|A - A_k\|_F = \min_{rank(\bar{A})\leq k} \left\|A - \bar{A}\right\|_F = \sqrt{\sigma_{k+1}^2 + \cdots + \sigma_n^2} \tag{45}$$

and $\bar{A_k}$ is unique if and only if $\sigma_k \neq \sigma_{k+1}$. A proof can be found in the appendix of I. Markovsky's book *Low-Rank Approximation: Algorithms, Implementation, Applications*[5].

2.3.4 Remark

From the previous theorem, we construct $A_k = U_k\Sigma_k V_k^T$ and notice it has $k(m + n)$ entries which is far less than A's mn entries when k is small. On the other hand, we can write A_k in another form which is

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T \tag{46}$$

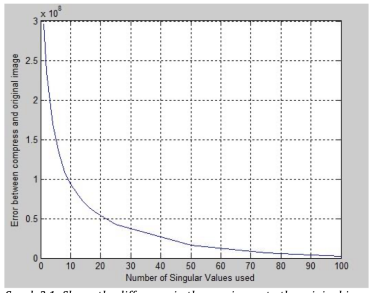
where σ_i are A's singular values with $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k \geq 0$ and u_i, v_i are the i-th left and right singular vectors of U_k, V_k. Therefore A_k is written as the sum of k's matrices with rank one and each weighted by a singular value. Conceptually, the big singular values accounts for more than the small numbers in terms of an approximation. When the first several singular values are far greater than the rest, the we let k equal to the number of large numbers and our approximation works well.

2.3.5 Practical Application[4]

SVD has a very important application in image compression. If we think of an image as a matrix, in which every entry is a pixel value, the SVD algorithm can be used to decompose the matrix into matrices U, Σ, and V. Then the image can be compressed and approximated using the first k columns of U, first k singular values of Σ, and first k rows of V. These k values can also be called components. The more components are used to approximate the original image, the better is the approximation, the better is the resolution of the approximated image. However, logically, the more components we use, the more calculations there are, the more data there is to process.

The use of SVD in image compression does reduce the complexity of the algorithm. For greyscale images, in which there is only one channel of color, SVD allows to reduce the data from $m * n$ to $k * (m + n + 1)$. For color images, where we use three channels of color, SVD reduces the data from $m * n$ to $3k * (m + n + 1)$, where k is the number of modes, or as mentioned above, the number of singular values we use for our approximation.

From this graph we can see that the more singular values are used in the algorithm, the less is the error between the original and the approximated image. [6]



It is important to mention, that despite being used in image compression, some other techniques are more efficient. One of the more efficient techniques used for image compression is JPEG (main algorithm of compressing images to the image file format of .jpg). The main part of JPEG is DCT (Discrete Cosine Transform), which involves more complex computations, which also take into account human perception of color, therefore making the compressed images more high-quality for the visual perception.

2.4 Modifying the Classical Singular Value Decomposition (SVD) [10]

2.4.1 Introduction

An improved algorithm PSVD for computing the singular subspace of a matrix corresponding to its smallest singular values is presented. As only a basis of the desired singular subspace is needed, the classical Singular Value Decomposition (SVD) algorithm can be modified in three ways. First, the Householder transformations of the bidiagonalization need only to be applied on the base vectors of the desired singular subspace. Second, the bidiagonal must only be partially diagonalized and third, the convergence rate of the iterative diagonalization can be improved by an appropriate choice between QR and QL iteration steps. An analysis of the operation counts, as well as computational results, show the relative efficiency of PSVD with respect to the classical SVD algorithm. Depending on the gap, the desired numerical accuracy and the dimension of the desired subspace, PSVD can be three times faster than the classical SVD algorithm while the same accuracy can be maintained. The new algorithm can be successfully used in total least squares applications, in the computation of the null space of a matrix and in solving (non) homogeneous linear equations. Based on PSVD a very efficient and reliable algorithm is also derived for solving nonhomogeneous equations.

2.4.2 The partial SVD algorithm (PSVD)

The original motivation was to improve the speed of the classical SVD algorithm when one requires only a basis of the singular subspace of a matrix corresponding to its smallest singular values.

In that case, three improvements are possible. I will focus on two of them as follows.

Delay initialization phase

During the second phase of the classical SVD algorithm, the matrices U and V are initialized by forming the products of applied Householder transformations $P = P^{(1)}...P^{(p)}$ and $Q = Q^{(1)}...Q^{(q)}$ explicitly. Since only a few vectors in U and/or V are requested, we can save a great amount of work by delaying the initialization of U and V until the end of the diagonalization phase. By only applying the Householder transformations $P^{(i)}$(resp.$Q^{(i)}$)on those vectors in G (resp. H) in $U = PG$, $V = QH$ which are associated with the desired smallest singular values of the bidiagonal $J^{(0)}$,we obtain the desired base vectors in U (resp. V) with a reduced computational effort.

Partial diagonalization

If convergence has occurred to all desired singular values, then the associated singular vectors remain unchanged during the following QR iterations. Hence, we can stop the QR iterations immediately after all desired singular values have been reached. In case of one singular vector, this means that the QR method only needs to converge to the associated singular value. No further QR iterations are needed for the computation of the required singular vector. Thus, the iteration process may be stopped as soon as convergence has occurred to all desired singular values. Moreover, if only a basis for a singular subspace

is required, we need not zero out all associated superdiagonal elements of the bidiagonal $J^{(0)}$. We have only to partition the bidiagonal into unreduced subbidiagonals, such that all singular values of each subbidiagonal are either all greater than a given bound θ or all smaller than or equal to θ. The required basis then consists of all the vectors associated with those subbidiagonals, whose singular values are all smaller than or equal to the bound θ.

By choosing an appropriate shift for the QR iterations, we proceed as follows:

- One constructs a sequence of bidiagonals $J^{(i)}$, similar to $J^{(0)} \longrightarrow J^{(1)} \longrightarrow \cdots \longrightarrow \sum$ where $J^{(i+1)} = (S^{(i)})^{(T)} J^{(i)} T^{(i)}$ in the classical SVD algorithm.

- After each QR iteration step with appropriate shift (ith step), one obtains:

$$J^{(i)} = \begin{pmatrix} J_1^{(i)} & 0 & & \\ & J_2^{(i)} & & \\ & & \ddots & \\ & & & J_k^{(i)} \end{pmatrix} \tag{47}$$

containing k unreduced subbidiagonals:

$$J_j^{(i)} = \begin{pmatrix} x & x & & & \\ & x & x & & \\ & & \ddots & \ddots & \\ & & & & x \\ & & & & x \end{pmatrix}, j = 1, \cdots, k \tag{48}$$

- The subbidiagonals $J_j^{(i)}$ (o are partitioned into three classes:

$C_1 = \{J_j \mid$ all singular values of J_j are$> \theta\}$,

$C_2 = \{J_j \mid$ all singular values of J_j are $\leq \theta\}$,

$C_3 = \{J_j \mid$ has at least one singular value $\leq \theta$ and at least one singular value $> \theta\}$,

- If C_3 is empty then stop, else apply one QR iteration step onto each element of C_3 and classify again.

In order to obtain a basis for the singular subspace of a matrix corresponding to its smallest singular values, the shift is set equal to the smallest diagonal element of the considered subbidiagonal. This shift must be smaller than the given bound θ, otherwise it is set equal to zero. With this choice the iteration process is forced to converge first to the smallest singular values of the matrix and its associated vectors.

If no bound on the singular values can be given but instead, the dimension d of the desired singular subspace is known, then one can compute after the bidiagonalization phase and using the bisection method, an appropriate bound θ such that the matrix has exactly d singular values smaller than or equal to θ.

The classification of the subbidiagonals is based on the Sturm sequence property of the unreduced tridiagonal matrices $T_j^{(i)} = J^{(i)T} J_j^{(i)}$. One can easily compute the number of singular values of $T_j^{(i)}$ smaller than or equal to the bound θ, by counting the number of sign changes in the sequence

$p_0(\theta)$,$p_1(\theta)$,...,$p_s(\theta)$

where $p_0(\theta) \equiv 1$ and $p_r(\theta) = \det(T_j^{(i)} - \theta I)$ for $r = 1, ...s$ with s the dimension of $T_j^{(i)}$.(Convention: $p_r(\theta)$ has the opposite sign from $p_{r-1}(\theta)$ if $p_r(\theta) = 0$). A simple determinantal expansion can be used to show that

$$p_r(\theta) = (\alpha_r - \theta)p_{r-1}(\theta) - \beta_r^2 p_{r-2}(\theta), r = 1, ..., s \tag{49}$$

with α_r the diagonal and β_r the superdiagonal elements of the tridiagonal $T_j^{(i)}(\beta_1 = 0)$. These coefficients are immediately obtained from their relation with the elements of the bidiagonal $J_j^{(i)}$. Let

$$T_j^{(i)} = J_j^{(i)T} J_j^{(i)} = \begin{pmatrix} \alpha_1 & \beta_2 & & 0 \\ \beta_2 & \alpha_2 & \ddots & \\ & \ddots & \ddots & \beta_s \\ 0 & & \beta s & \alpha_s \end{pmatrix}, J_j^{(i)} = \begin{pmatrix} q_1 & e_2 & & 0 \\ & q_2 & \ddots & \\ & 0 & \ddots & e_s \\ & & & q_s \end{pmatrix} \tag{50}$$

then

$\alpha_i = q_i^2 + e_i^2, i = 1, ..., s$ and $e_1 = 0$,

$\beta_{i+1} = q_i e_{i+1}, i = 1, ..., s - 1$.

Conclusion

In essence, a delay of the initialization phase improves the efficiency of PSVD optimally if the number of desired vectors $<< n$. The partial diagonalization works best if the gap between the singular values associated with the desired and undesired vectors is large. This is usually the case when solving TLS problems and (non)homogeneous linear equations. In case of a smaller gap, the convergence rate can be improved by using QL iterations.

Teamwork contribution

Ivan Khurudzhi: 2.2, 2.3.5
Zhengjie Xiong: 1.1, 2.3.1-2.3.4
Yuqin Mu: 2.4
Zhenya Liu: 1.2, 1.3, 2.1

References

[1] Making sense of principal component analysis, eigenvectors, and eigenvalues. 2011.

[2] G.H. GOLUB and C REINSCH. Singular value decomposition and least squares solutions. *Handbook for Automatic Computation, II, Linear Algebra*, 1971.

[3] Jonathan Hui. Machine learning — singular value decomposition (svd) principal component analysis (pca). 2019.

[4] Zecheng Kuan. Singular-value decomposition and its applications. 2018.

[5] Ivan Markovsky and Konstantin Usevich. *Low rank approximation*, volume 139. Springer, 2012.

[6] Brady Mathews. Image compression using singular value decomposition (svd). 2014.

[7] Gregory Piatetsky. Interview with simon funk. 2007.

[8] Friedberg S.H., Insel A.J., and Spence L.E. *Linear Algera, 4th Edition*. Pearson, 2002.

[9] Jonathon Shlens. A tutorial on principal component analysis. 2014.

[10] Sabine Van Huffel, Joos Vandewalle, and Ann Haegemans. An efficient and reliable algorithm for computing the singular subspace of a matrix, associated with its smallest singular values. *Journal of computational and applied mathematics*, 19(3):313–330, 1987.

[11] Wikipedia. Low-rank approximation.