

Solving least squares using SVD

So, now I am going to tell you about one of the most popular ways of solving this system of equations. Again, this is probably something that you have done in linear algebra, but it gets used a lot in optimization as well, which is using the SVD. Ok. Many of you who have done linear algebra have not done SVD. So, this will be a bit of a revision. Anyone here who has not solved this using SVD? Yeah, you can raise your hand, it is ok. I am expecting most of you to raise your hand over here. Good.

Solving using the SVD.

$A = USV^T$

$m \times n$ $m \times m$ $m \times n$ $n \times n$

$\sigma_i \geq 0$

$m > n$ $m < n$

$A^T A = V S^T U^T U S V^T = V S S^T V^T$

So, this is going to be fun because this is a really useful tool. So, my matrix A over here... Every matrix has an SVD, that is the good news, right? So, I am going to write this as

$$A = USV^T$$

That is the singular value decomposition. This matrix A is $m \times n$. My input matrix is $m \times n$, right? For now, we will keep it general; it could be tall, fat, square, whatever.

What is U ? U is $m \times m$, something more about it: it is an orthogonal matrix. All the columns are orthogonal to each other, right? It forms, therefore, if they are orthogonal with respect to each other, what is the other nice property that you get out of it? It is a basis for \mathbb{R}^m . Any vector in m -dimensional space can be written in terms of the columns of U . What about Σ ? This guy is size

$m \times n$, and what are its properties? It is a diagonal matrix. It may not be square, but it is a diagonal matrix, and the diagonal entries are the singular values, right. Similarly, this is $n \times n$, V , and V is also what? Orthogonal matrix, ok. It is an $n \times n$ orthogonal matrix; therefore, we also have a nice basis for \mathbb{R}^n , right?

Handwritten notes on a whiteboard:

$m \times n$ $m \times m$ $n \times n$ $\sigma_i \geq 0$ $\begin{bmatrix} \diagup & \diagdown \\ 0 & \end{bmatrix}$ $\begin{bmatrix} \diagup & \diagdown \\ & \end{bmatrix}$

$$A^T A = V S^T U^T U S V^T = V \underbrace{S^T S}_{n \times n} V^T$$

$$A^T A x = A^T b \rightarrow \underbrace{V S^T S V^T}_{n \times n} x = \underbrace{V S^T U^T}_{n \times n} b \quad \text{L.M by } V^T$$

$$\underbrace{S^T S}_{n \times n} V^T x = S^T U^T b$$

Invertible? $\sigma_i^2, i=1 \dots n$ if $m \geq n$ AND full col rank then $\sigma_n \neq 0$.
in this case $S^T S$ invertible.

So, this is σ_i , and all the singular values, regardless of what the matrix is, are greater than or equal to 0. There is never a negative singular value, ok. So, if it is a tall matrix, this is how Σ is going to look like when $m > n$, and if m is less than n , then it is going to look like this:

$$\Sigma = \begin{bmatrix} \sigma_1 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 \\ 0 & 0 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \sigma_n \end{bmatrix}$$

This is how the Σ matrix looks. Ok. So, now let us look at this equation that we are trying to solve: $A^T A x = A^T b$. So, let us look at A^T . What is $A^T A$? A^T going to be, when we apply the transpose operation, everything slips around. Right? $(ABC)^T = C^T B^T A^T$. So, let us write down $A^T A$:

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

Can this get simplified any further? $U^T U$ is the identity matrix, ok. So, this becomes:

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

What will happen to $\Sigma^T \Sigma$? It is going to be a square, of course, it is going to be square. Anything else that we can say about it? It will be diagonal. The singular values will be sitting on the diagonal as what? σ_i^2 . So, $\Sigma^T \Sigma$ is the square of the singular values. So, $\Sigma^T \Sigma$ will be a diagonal

matrix. The size of Σ^T is $n \times m$, and so $\Sigma^T \Sigma$ is $n \times n$, right? So, this is $n \times n$, and that makes sense because V and V^T are anyway $n \times n$.

NPTEL

$$V^T x = (S^T S)^{-1} S^T U^T b \quad \text{in this case } S^T S \text{ invertible.}$$

$$x = V (S^T S)^{-1} S^T U^T b.$$

So, everything, all the sizes are consistent, right? So, that is one check. Ok. So, now we are looking at the equation $A^T A x = A^T b$, that was what I was trying to solve. Now, let us plug in what we have done. So, $V \Sigma^T \Sigma V^T x = A^T b$.

So, that becomes:

$$V \Sigma^T \Sigma V^T x = V \Sigma^T U^T b$$

What can we do next to simplify this? There are a lot of extra terms over here. What would you do? Remember, these are not scalars that I can just knock off a common factor on both sides, right? One suggestion is pre-multiply, I mean, left-multiply by V^T , because for an orthogonal matrix, the inverse is the transpose, right? Ok. So, left-multiply by V^T , that is going to give me:

$$\Sigma^T \Sigma V^T x = \Sigma^T U^T b$$

Now, the question is, is this invertible? It depends, right? So, first of all, what are they? $\Sigma^T \Sigma$ is $n \times n$, right? It is square and of size $n \times n$, and the entries of this are what? σ_i^2 , for $i = 1$ to n . Now, if this is a tall matrix and full column rank, what does it mean for the singular values? All are non-zero, right, up to σ_n . Ok, so if this is a tall matrix and full column rank, then $\sigma_n \neq 0$.

In that case, this matrix is invertible, yeah. So, let us flip this guy around. I am going to get:

$$V^T x = \Sigma^T \Sigma^{-1} \Sigma^T U^T b$$

Can I still simplify it further? I am interested in x . Left-multiply by V , so that $V V^T$ goes away. So, then I am left with:

$$x = V \Sigma^T \Sigma^{-1} \Sigma^T U^T b$$

So, remember, you can see why full column rank is absolutely critical because without that, I cannot push this guy to the right-hand side; I am stuck. Right? So, full column rank is allowing me to write down x in a closed-form expression. Is this term underlined over here going to be something simple? Right? $\Sigma^T \Sigma^{-1}$ is a square matrix, and what are the entries? $1/\sigma_i^2$, that is getting multiplied by Σ^T . What will I be left with? $1/\sigma_i$, kind of thing. But we have to do it a little carefully because I am multiplying a square matrix by a fat matrix. Σ^T is now a fat matrix, right? Now, Σ^T is something that looks a little bit like this. So, I am going to write this as V remains as it is, this term underlined is what I am going to call diagonal inverse, roughly $1/\sigma_i$, ok.

The whiteboard shows the following derivation:

$$x = V (\Sigma^T \Sigma)^{-1} \Sigma^T U^T b$$

$$x = V D^{-1} U_1^T b$$

where U_1 is labeled as the first n columns of U .

$$= \begin{bmatrix} v_1 & & & & v_n \end{bmatrix} \begin{bmatrix} 1/\sigma_1 & & & & \\ & & & & \\ & & & & \\ & & & & \\ & & & & 1/\sigma_n \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} \begin{bmatrix} b \\ \vdots \\ b \end{bmatrix}$$

$$= \sum_{i=1}^n \left(\frac{1}{\sigma_i} U_i^T b \right) x v_i$$

The lecturer is visible in the bottom right corner of the whiteboard frame.

And this, I am going to write it in a little bit of a different notation: U_1 . Now, why am I calling this U_1 ? It is because this guy is like this, right? Σ^T is like this, and when it gets multiplied by U^T , what will happen? I am going to have a whole bunch of zeros get multiplied by the rightmost columns of this U^T . So, what I will be left with are the first n columns of U^T . So, this U_1 is what I am calling the matrix which is found by taking the first n columns of U .

So, here is my solution which comes from SVD with a simple manipulation of the first n columns of U^T (no, it should be of U). So, let us have a look at this. These are the columns of V , v_1 up to v_n . Diagonal, so this is going to be $\frac{1}{\sigma_1}$ up to $\frac{1}{\sigma_n}$. And then I have something that is what is going on over here. You can also write this; you can open this up in terms of what is very popularly called as the outer product notation, which will make it very, very simple.

So, let us write that down over here. $n = 1$ to n . This $\frac{1}{\sigma_i}$ is a scalar, it is going to pop out $\frac{1}{\sigma_i}$, and u_i^T is getting multiplied by b . So, before I write this, let us just get some intuition. Let us start

from the rightmost side. I have a whole bunch of row vectors that are getting multiplied by b , and those are going to form scalars.

Those scalars are then getting multiplied by v . Right. So, imagine if this whole thing over here should boil down to what? A scalar, a vector, or a matrix? It should boil down to a column vector. Once I get a column vector, that is getting multiplied by a matrix V . So, in terms of linear combination, how am I interpreting this? x is a linear combination of the V 's with some coefficients.

Now, all we are trying to do is interpret what those coefficients are. So, I am going to get $\frac{1}{\sigma_i}$, and you notice this is going to be $u_i^T \cdot b$. This is my coefficient, and that is simply getting multiplied by v_i . So, this is my solution to the least squares. This is like the ideal textbook solution that starts with the SVD, you work it through, and you end up like this.

This turns out to be hiding a lot of important points over here, which, from a numerical point of view, will make or break your solution. So, now let us have a look at that. I have written n over here because we said that the matrix had full column rank.

So now let us make things a little bit more interesting. This summation over here, I can write it in a way that makes things a little bit more realistic. In this entire setup $Ax = b$, where is noise appearing? b , right? Because my $A \cdot x$ was my linear model, $x_1 + x_1 \cdot x_2 \cdot t$, whatever, right? That is the model part. b is where I make a measurement, and that measurement can have some noise. So, if I had noise in my measurements, who is the most dangerous guy, σ_1 or σ_n ? I hear σ_n , anybody else? So, where is the noise sitting now? The noise is hiding in this guy, right. So, who is the most dangerous guy? Who?

NPTEL

$$x^* = \sum_{i=1}^n \left(\frac{1}{\sigma_i} u_i^T b \right) v_i$$

the least squares problem.

pseudo inverse

noise

$k = 10^4, \sigma_1 = 1, \Rightarrow \sigma_n = 10^{-4}, b = b_0 + \Delta$

Reduce error by truncating SVD

$$x^* \approx \sum_{i=1}^r \left(\frac{1}{\sigma_i} u_i^T b \right) v_i$$

OPTIMIZATION THEORY AND ALGORITHMS

Sigmas are always in decreasing order; σ_1 is highest, σ_n is smallest. So, σ_n , right? Because σ_n is the smallest guy, and it is in the denominator. So, the way to see this is, I can just write this as

$$\sum_{i=1}^{n-1} \frac{1}{\sigma_i} u_i^\top \cdot b \cdot v_i + \frac{1}{\sigma_n} u_n^\top \cdot b \cdot v_n.$$


Let us say that my condition number κ was 10^4 and σ_1 was 1. Therefore, σ_n is how much? 10^{-4} , this is my condition number. This is a reasonable condition number; it is not something very bad, right. So, what is going to happen now in this term $\frac{1}{\sigma_n} \cdot 10^4 \cdot b$? I can write it as $b_0 + \delta$, right? This noise is inside b . You never have access to it separately. The b is a noisy measurement; it has some true value and some measurement, and you can never split the two. You get it as a wholesale deal.

So, you notice what will happen now. This b , which I have written as $b_0 + \delta$, that δ is getting multiplied by $\frac{1}{\sigma_n}$, which is 10^4 . So, okay. That is one noise term. And on the other hand, look at the higher singular values, which you were so happy about. σ_1 was 1; that is forming the first half of the solution, the first part of the summation $\frac{1}{\sigma_1}$, no problem. But it is competing against $10^4 \cdot \delta$ and the noise. So, this noise over here in the solution is going to swamp your true solution.

Okay, so when you write this, it is one of the major advantages of writing the solution using the SVD: you can immediately see which is the guy who is going to corrupt your solution the most. The smallest singular values are the ones that are going to corrupt and give you the maximum error in your solution. If you had done, for example, Gaussian elimination or Cholesky, whatever, you would not be able to see, "Oh, this is what is happening!"

I mean, there are of course many strategies over here. Supposing you are in charge of controlling the error, what would you do? The simplest strategy is, let us knock off the lowest singular values, meaning I exclude them from the solution.

It is a very popular technique in engineering, and the name for it is, therefore, truncated SVD. That is where the idea comes from: that if I have some singular values which are very low, they are going to amplify the error. Rather than face that error, I get rid of it. There is going to be some inaccuracy now that is coming, right? So, reduce error by... Here I gave you the example of just the highest, I mean σ_n being very small, but you could have the last several sets of singular values could be very bad. So, that is why truncated does not mean just one; you could truncate it by a few terms, right?

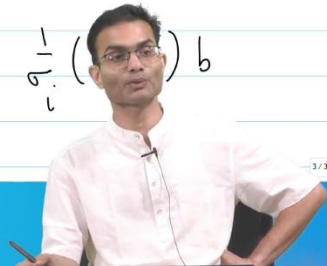


Reduce error by truncating SVD

$$x^* \approx \sum_{i=1}^r \left(\frac{1}{\sigma_i} u_i^T b \right) v_i$$

v_n

$$= \sum_{i=1}^r \frac{1}{\sigma_i} v_i (u_i^T b) \rightarrow \sum_{i=1}^r \frac{1}{\sigma_i} v_i u_i^T b$$

$$x = \sum_{i=1}^r \frac{1}{\sigma_i} (u_i^T b) v_i$$


So, this would then be written as

$$\sum_{i=1}^n \frac{1}{\sigma_i} u_i^T \cdot b \cdot v_i.$$

So, this is my x^* . By the way, this solution which I wrote over here goes by another very common name in the literature. Does anyone want to guess what the solution is called? It is called the pseudo-inverse, called the pseudo-inverse because it is not a square matrix that I inverted; therefore, it is not the inverse. But it has properties that are almost as good as the inverse because I basically minimized $A \cdot x - b$. So, this is also called the pseudo-inverse and this is our truncated SVD.

So, from a linear algebra point of view, can you tell what just happened over here? The hint is that we said that x is an n -dimensional vector; therefore, I should be able to express it in the basis of which? U or V ? V , right? Any vector should be expressed as a linear combination of my n basis vectors. My original solution had... Did it have all n basis vectors? The pseudo-inverse did.

The truncated solution, however, does it live in the same space? No, right. So, as error is coming, the singular values which are lowest in magnitude are corrupting that part of the subspace, which is corresponding to the highest singular index, right? σ_n . So, you notice that this guy, this truncated solution, has no components of x along v_n , and if I truncate more, it will have nothing along v_{n-1} , right. So, that is how it is.

There are tricks and techniques in signal processing where what they will do is, you say that I know that there are certain dimensions, for example, v_n, v_{n-1}, v_{n-2} , where I dare not use my data. If I use my data, my error is going to blow up. So, I do not estimate that part of the solution

from the data; I estimate that part of the solution using some a priori information about the problem. This is like things like compressive sensing and all of these ideas. What can be determined from data, you nail it down using the truncated SVD. What is left, you use your other information about the problem. For example, I may – and this is a very real-life example – I may say that the solution is sparse in the DFT domain.

Natural images, for example, if you take an image of a dog, cat, or whatever, and take its DFT and look at the coefficients, you will find out most of the coefficients are actually... It is just a property of natural images. But if you look at it in the pixel domain, it is a full image. There is nothing that you can discard. If I start dropping out some pixels, you can see this is a pixelated image. On the other hand, take its DFT and look at its coefficient. You will find out that a large number of coefficients are actually very low in magnitude.

If you set them to 0 and then take the inverse DFT, the image looks almost the same. This is at the heart of, I mean in a very crude way, at the heart of JPEG and all that. You can get rid of information like this without losing the picture quality. So, I get some solution. Now, supposing I transform it into the DFT and I try to reduce the coefficients, that is how I can set these extra v_n , v_{n-1} , you know, in some way. These are the kind of tricks and techniques people will use in signal processing, image processing, and so on. But having this idea of the singular vectors and what noise is doing to what, then you are in command of the situation, okay?

These dimensions are fine. For example, v_1 , v_2 are fine. Noise is not going to bother me. v_n , v_{n-1} is going to be dangerous. Let me not let noise predict that part of the solution. This is how you can... Keep this in mind when you look at your course project papers. You will find tricks like this being played all the time.

Any other questions on this truncated SVD or SVD in general for those of you who saw the SVD solution for the first time? Another way in which you find this expression which I have written over here, $u_i^T b$, is it a scalar or a vector? It is a scalar. v_i is obviously a vector. Can I rearrange the order of scalar and vector? Who cares, right? So, I could also write this as $\frac{1}{\sigma_i} v_i \cdot u_i^T b$. Does this bracket have any real meaning or can I skip writing that bracket altogether? I can skip it.

So, I can also write it like this. Now, can I reinterpret this? Can I put a bracket like this now? Is it legal? It is perfectly legal. Now, what is the meaning of this now? So, I have written my solution as $i = 1$ to r , $x = \sum_{i=1}^r \frac{1}{\sigma_i} v_i \cdot u_i^T b$. What is the meaning of this thing that I have written now? It is a sum of rank-1 matrices because v_i is a column vector, u_i^T is a row vector, and their product is an $n \times n$ matrix, but it is a matrix of rank 1. How many such rank-1 matrices are there? r . So, I have written my solution as the sum of r rank-1 matrices. I have just changed the brackets around, and I have got this rank-1 approximation going on over here. This is another very nice interpretation.

For example, there are all of these rating problems, right? Netflix rating, that if a user has ranked these movies, can you guess what are the ratings that he or she would give to movies which they have not seen? In these kinds of problems, it ends up being a problem of completing an unknown matrix. I have some entries known and some entries unknown. Many times, that is written in terms of a minimum rank expansion of a matrix.

So, tricks like this help you over here. If you wanted to write the x in terms of the least number of rank-1 solutions, here it is sitting over here. Here, there are r rank-1 matrices. If you wanted to reduce it to make it just 1, you just put $r = 1$. So, this is sounding vague, but go look up ratings problems and matrix completion problems. So, this idea of using rank-1 matrices comes from here.

Full rank? Yeah, which was the pseudo-inverse. Yeah, this was the A has full column rank; this gave us the solution $x^* = \sum_{i=1}^n \frac{1}{\sigma_i} u_i^\top b v_i$. For the normal problem, not $Ax = b$. Well, yeah, you could call it for $Ax = b$ also, yeah. Okay, that is... Yeah, that is not invertible. I mean, then what happens? That $S^\top S$ is not invertible, right? I agree with you. So, what you could do instead is this. So, it is a defective kind of a solution, but you know in engineering problems, you are desperate, and you want a solution which is somewhat close to what you want.

So, this is what you can do. Well, if it is rank deficient, it will satisfy. If you keep aside the error part of it of the highest singular value. Actually, let us try that out. You can try that out. Take this, supposing A is not full rank, take this solution and plug it into the normal equation, and we will just see: does it satisfy? σ will get multiplied by, yes.