Singular Value Decomposition

An application to Big Data

Davide Sferrazza

January 11, 2023

Università degli Studi di Palermo

Outline

Singular Value Decomposition

Definition of SVD

Theorem

Given a matrix $A \in \mathbb{R}^{m \times n}$, it can always be found a decomposition such that

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

where $U \in \mathbb{R}^{m \times m}$, $V \in \mathbb{R}^{n \times n}$ and $\Sigma \in \mathbb{R}^{m \times n}$.

U and V are two orthogonal matrices and Σ is a diagonal matrix, namely:

$$(\Sigma)_{ij} = \begin{cases} 0, & i \neq j \\ \sigma_i, & i = j \end{cases}$$

where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_p \geq 0$, $p = \min\{m, n\}$.

Definition of SVD

The non-zero entries of Σ, denoted by σ_i, are called *singular values*.

They are arrenged in a nonincreasing order by convention.

The column vectors $\boldsymbol{u_i}$ of U are called *left singular vectors* and those $\boldsymbol{v_i}$ of V are called *right singular vectors*.

Since in general $m \neq n$, we have:

$$A = \sum_{i=1}^{p} \boldsymbol{u_i} \sigma_i \boldsymbol{v_i}^T$$

Definition of SVD

Theorem

If for some r such that $1 \leq r < p$ we have

$$\sigma_1 \geq \ldots \geq \sigma_r > \sigma_{r+1} = \ldots = \sigma_p = 0$$

then

- $rank(A) = r$
- $A = \sum_{i=1}^{r} \boldsymbol{u_i} \sigma_i \boldsymbol{v_i}^T$

This means that all other $p - r$ dimensions of matrix A are linear combinations of the first r.

Definition of SVD

Lower rank approximation

Let $A \in \mathbb{R}^{m \times n}$ be a matrix whose rank is $rank(A) = r$.

If for a fixed integer value $k < r$ we define

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

and

$$\mathcal{B} = \left\{ B \in \mathbb{R}^{m \times n} : rank(B) = k \right\}$$

then

$$\min_{B \in \mathcal{B}} \|A - B\|_2 = \|A - A_k\|_2 = \sigma_{k+1}$$

This result tell us that A_k represents the best approximation (considering the *spectral norm*) of rank k of matrix A.

Singular values computation

To compute the singular values, consider the transpose of A given its decomposition:

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

The symmetric matrix $A^T A$ is equal to:

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

Furthermore, this equation can be written as:

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

This means that the diagonal entries of the square matrix $\Sigma^T \Sigma$, which are the square of the singular values, are the eigenvalues of matrix $A^T A$ and V is the matrix of eigenvectors.

Singular values computation

Similarly, consider the product of AA^T. It is equal to:

$$AA^T = (U\Sigma V^T)(V\Sigma^T U^T) = U\Sigma\Sigma^T U^T$$

Which means that:

$$AA^T U = U\Sigma\Sigma^T$$

Hence U is the matrix of eigenvectors of AA^T.

Since $rank(A) = r$, only the first r eigenvalues of AA^T and $A^T A$ are non-zero.

Finding eigenvalues and eigenvectors

QR Method

A possible method to find eigenvalues and eigenvectors of a matrix is based on QR decompositions and this theorem:

Theorem

Suppose $A \in \mathbb{R}^{n\times n}$ is a matrix having eigenvalues $\lambda_1, \lambda_2, \ldots \lambda_n$ satisfying

$$|\lambda_1| > |\lambda_2| > \ldots > |\lambda_n| \tag{2}$$

then the following sequence for $A_1 = A$ and $k = 1, 2, \ldots$

$$\begin{cases} A_k = Q_k R_k \\ A_{k+1} = R_k Q_k \end{cases} \tag{3}$$

converges to an upper triangular matrix where $(A_k)_{ii} = \lambda_i,\ i = 1, 2, \ldots, n$.
In case (2) is not satisfied, this sequence converges to a triangular matrix with square blocks of order at most 2 along the diagonal.
If A is symmetric, then the sequence converges to a diagonal matrix.

QR Method

So a basic implementation would be like this:

```
while err < toll
    [Q, R] = qr(A);
    A = R * Q;

    err = max( max( tril(A, -1) ) );
end
```

This method could be speed up using a technique called *shifing*:

```
n = length( A );
while err < toll
    % A(n, n) is an usual choice, it could be any real number
    T = A(n, n) * eye(n);
    [Q, R] = qr( A - T );
    A = R * Q + T;

    err = max( max( tril(A, -1) ) );
end
```

QR Method

In our case, this method must be applied to AA^T and A^TA, so if (3) converges then we have a diagonal matrix.
Now, consider the diagonalization of B, where $B = AA^T$ or $B = A^TA$:

$$B = P\Lambda P^{-1} = P\Lambda P^T$$

As B can be factored using (3), the matrix containing the eigenvectors must be equal to:

$$P = \prod_i Q_i = Q_1Q_2Q_3\cdots$$

Hence, for every iteration $B_k = Q_kR_k$ and $B_{k+1} = R_kQ_k$, requiring each step to have a computational cost equal to $O(\frac{2n^3}{3})$.

Hessemberg Reduction

To achieve a lower computational cost, one solution consists in transforming B in a similar tridiagonal matrix using Householder matrices.
A triangular matrix can be obtained because B is symmetric.

In general, the process of trasforming a matrix in a similar matrix using Householder matrices allows us to obtain a matrix in which $B_{ij} = 0$, for all $i > j$. Such a matrix is called a *Hessemberg matrix*.

Consequently, after B has been transformed, each step of the previous code can be realized using *Givens rotation matrices*.

Givens rotation matrices

A *Givens rotation* can be represented with the following matrix:

$$G_{ij} = \begin{array}{cc}
\begin{pmatrix}
1 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\
\vdots & \ddots & \vdots & & \vdots & & \vdots \\
0 & \cdots & c & \cdots & -s & \cdots & 0 \\
\vdots & & \vdots & \ddots & \vdots & & \vdots \\
0 & \cdots & s & \cdots & c & \cdots & 0 \\
\vdots & & \vdots & & \vdots & \ddots & \vdots \\
0 & \cdots & 0 & \cdots & 0 & \cdots & 1
\end{pmatrix}
&
\begin{array}{l}
\\ \\ \longleftarrow \ \text{i} \\ \\ \longleftarrow \ \text{j} \\ \\ \\
\end{array}
\\
\begin{array}{ccccccc}
 & & \uparrow & & \uparrow & & \\
 & & \text{i} & & \text{j} & &
\end{array}
&
\end{array}$$

where $c = \cos\theta$ and $s = \sin\theta$, for a particular value of $\theta \in [0,\ 2\pi]$.
A rotation occurs in the plane spanned by the two coordinates axes i and j.

Givens rotation matrices

Using the following values:

$$\cos\theta = \frac{|b_{ii}|}{\sqrt{b_{ii}^2 + b_{ji}^2}}$$

$$\sin\theta = \operatorname{sign}\left(\frac{b_{ji}}{b_{ii}}\right) \frac{|b_{ji}|}{\sqrt{b_{ii}^2 + b_{ji}^2}}$$

when B gets left multiplied by G_{ij}, the final matrix will have the element in position $(j,\, i)$[1] equal to 0.

Therefore, instead of factoring B at each step by the QR method, if B is tridiagonal, we can simply iterate n times by constructing $G_{k\,k+1}$ and eliminating the elements in position $(k+1,\, k)$ and $(k,\, k+1)$.

[1]Note that the indices are reversed

Pseudocode for finding eigenvalues and eigenvectors

A pseudocode of what to do should clarify what we have done so far (excluding shifting for brevity):

$B, P \leftarrow$ hessemberg(B)
while error > toll **do**
  for $k = 1, 2, \dots, n$ **do**
    construct $G_{k\,k+1}$
    $B \leftarrow G_{k\,k+1} B G^T_{k\,k+1}$
    $P \leftarrow P G^T_{k\,k+1}$
  end for
  update error
end while

The function implementing the Hessemberg reduction must also return the matrix used for the transformation, meaning that $B = PHP^T$.
Note that P is updated in this way because $Q_i = G^i_{12} G^i_{23} \cdots G^i_{n-1n}$, where i denotes the generic iteration for error minimization.

The Algorithm

Calculation of left singular vectors

Suppose we start computing V by the method explained in the previous slides.
We apply the method to $A^T A$ and get $P = V$ and $B = \Sigma^2$.
Once we have take the square root of the singular values, by explicitly writing Σ, the matrix containing the left singular vectors can be calculated as:

$$A = U \Sigma V^T \Rightarrow U = A V \Sigma^{-1}$$

The algorithm is now compplete. The next slides will illustrate the code.

Function to calculate Givens rotation matrices

```
function [c, s] = givens(A, i, j)
%GIVENS Function to compute cos and sin of Gij
%   Given a matrix A, this function returns a vector containing the values
%   of givens rotation matrix Gij such that Gij * A is equal to A except
%   for the element (i, j), which will be zero.
     c = abs( A(i, i) ) / sqrt( A(i, i)^2 + A(j, i)^2 );
     s = - sign( A(j, i) / A(i, i) ) * abs( A(j, i) ) / sqrt( A(i, i)^2 + A(j, i)^2 );
end
```

Function to calculate the Hessemberg reduction i

```matlab
function [A, P] = hessemberg(A)
%HESSEMBERG Fuction to compute the Hessember reduction of a matrix
%   Given a matrix A, this function trasform A in a similar hessemberg
%   matrix. This produces as output the hessember matrix and the matrix P
%   used for the tranformation.
    n = size(A, 2);
    P = eye(n);
    for k=1:n-2
        sigma = sign(A(k+1, k)) * norm(A(k+1:n, k));
        v = [sigma + A(k+1, k); A(k+2:n, k)];
        beta = 1 / (sigma * (sigma + A(k+1, k)));
        for j=k:n
            tau = beta * (v' * A(k+1:n, j));
            A(k+1:n, j) = A(k+1:n, j) - tau * v;
        end
        for j=1:n
            tau = beta * (A(j, k+1:n) * v);
            A(j, k+1:n) = A(j, k+1:n) - tau * v';
```

Function to calculate the Hessemberg reduction ii

```
            tau = beta * (P(j, k+1:n) * v);
            P(j, k+1:n) = P(j, k+1:n) - tau * v';
        end
    end
end
```

Function to calculate singular values and singular vectors i

```
function [M, H] = singular_vectors(A, toll, left)
%SINGULAR_VECTORS Function to compute the left or right singular vectors
%   Given a matrix A and a tollerance for the stopping criterion, this
%   function computes the left or right singular vectors of A and its
%   corrisponding singular values. If left is true then U is calculated, V
%   otherwise.
    if left
        [H, P] = hessemberg(A * A.');
    else
        [H, P] = hessemberg(A.' * A);
    end

    n = length(H);

    G = zeros(n - 1, 2);
    G_aux = zeros(2);

    M = P;
```

Function to calculate singular values and singular vectors ii

```
    err = toll + 1;

    while err > toll
        H1 = H;
        T = H(n, n) * eye(n);

        H = H - T;

        for k = 1:n-1
            [G(k, 1), G(k, 2)] = givens(H, k, k+1);
            G_aux(1, 1) = G(k, 1);
            G_aux(1, 2) = - G(k, 2);
            G_aux(2, 1) = G(k, 2);
            G_aux(2, 2) = G(k, 1);

            H(k:k+1, k:n) = G_aux * H(k:k+1, k:n);
        end

        for k = 1:n-1
```

Function to calculate singular values and singular vectors iii

```
            G_aux(1, 1) = G(k, 1);
            G_aux(1, 2) = G(k, 2);
            G_aux(2, 1) = - G(k, 2);
            G_aux(2, 2) = G(k, 1);

            H(1:k+1, k:k+1) = H(1:k+1, k:k+1) * G_aux;
            M(1:n, k:k+1) = M(1:n, k:k+1) * G_aux;
        end

        H = H + T;

        err = norm(diag(H - H1), 1);
    end

    H = sqrt( diag( diag( H ) ) );

    discard_imag = 10^-5;

    if all( imag( diag(H) ) < discard_imag )
```

Function to calculate singular values and singular vectors iv

```
        H = real(H);
    end
    if all( imag( diag(M) ) < discard_imag )
        M = real(M);
    end
end
```

Function to calculate SVD

```
function [U, S, V] = custom_svd(A, toll)
%CUSTOM_SVD Function to compute the SVD factorization of a matrix
%   Given a matrix A as input and a tollerance for the stopping criterion,
%   this function computes the SVD factorization of A.
    [V, S1] = singular_vectors(A, toll, false);

    [New_Diag, s_order] = sort(diag(S1), 'descend');
    S1 = diag( New_Diag );
    S1(length(S1), 1) = 0;

    V = V(s_order,:);

    S = S1;
    S1 = diag( 1./ diag( S1 ) );
    U = A*V*S1;
end
```

Application to Big Data

Dimensionality reduction

The SVD decomposition can be applied to Big Data in order to reduce the dimensionality of datasets.

As an example, consider the dataset [5] containing 33 different attributes for 145 students, which includes student ID, personal information and data, and higher education performance ratings[2].
Dimensionality can be reduced as follows:

1. truncate the SVD retaining k singular values, $A \approx U_k \Sigma_k V_k^T$;
2. compute the modified dataset as $D' = DV_k = U_k \Sigma_k$.

[2] https://archive.ics.uci.edu/ml/datasets/Higher+Education+Students+Performance+Evaluation+Dataset

References

References

[1] G. Monegato, *Metodi e algoritmi per il calcolo numerico*. Clut, 2008.

[2] C. C. Aggarwal *et al.*, *Data mining: the textbook*. Springer, 2015, vol. 1.

[3] J. Leskovec, A. Rajaraman, and J. D. Ullman, *Mining of massive data sets*. Cambridge university press, 2020.

[4] C. F. Van Loan and G. Golub, "Matrix computations (johns hopkins studies in mathematical sciences)," *Matrix Computations*, 1996.

[5] N. Yılmaz and B. Sekeroglu, "Student performance classification using artificial intelligence techniques," in *10th International Conference on Theory and Application of Soft Computing, Computing with Words and Perceptions - ICSCCW-2019*, R. A. Aliev, J. Kacprzyk, W. Pedrycz, M. Jamshidi, M. B. Babanli, and F. M. Sadikoglu, Eds., Cham: Springer International Publishing, 2020, pp. 596–603, ISBN: 978-3-030-35249-3.

[6] D. Dua and C. Graff, *UCI machine learning repository*, 2017. [Online]. Available: http://archive.ics.uci.edu/ml.