SVD Dimension Reduction

Richard Ketchersid

November 30, 2020

MATLAB code for this is available in the file "SVDImRed.mlx."

SVD can be used to reduce the dimension of data. The idea is that the amount of information contained in the data is related to the variance, the variance is is precisely $\sigma_1^2 + \cdots + \sigma_k^2$ where $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_k > 0$ are the non-zero singular values. You should just able to keep some of the data that accounts for most of the variance without loosing too much information.

Suppose we have collected N data measurements on M variables $V_1, \ldots, V_N$. Let $\boldsymbol{a}_i$ be the data values for V_i, then we can put these into a matrix $A = \begin{bmatrix} \boldsymbol{a}_1 & \cdots & \boldsymbol{a}_n \end{bmatrix}$.

So if A is a rank k matrix and

$$\begin{aligned} &U = \begin{bmatrix} \boldsymbol{u}_1 & \boldsymbol{u}_2 & \cdots & \boldsymbol{u}_k \end{bmatrix} \text{ is the matrix of left singular vectors,} \\ &V = \begin{bmatrix} \boldsymbol{v}_1 & \boldsymbol{v}_2 & \cdots \boldsymbol{v}_k \end{bmatrix} \text{ is the matrix of right singular vectors, and} \\ &\Sigma = \operatorname{diag}(\sigma_1, \sigma_2, \cdots, \sigma_k) \text{ is the diagonal matrix of singular values, then} \\ &A = U\Sigma V^T \end{aligned}$$

and

$$A = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T + \cdots + \sigma_k \boldsymbol{u}_k \boldsymbol{v}_k^T$$

Recall: $\boldsymbol{u}\boldsymbol{v}^T = \boldsymbol{u} \otimes \boldsymbol{v}$ is the *outer product* of $\boldsymbol{u}$ and $\boldsymbol{v}$ and this is a rank 1 matrix.

Taking the first $j \leq k$ terms gives us a rank j matrix that is as close as possible to A in both the Frobenius and L_2-norm. Set

$$A_j = \sigma_1 \boldsymbol{u}_1 \boldsymbol{v}_1^T + \cdots + \sigma_j \boldsymbol{u}_j \boldsymbol{v}_j^T$$

So that $A_k = A$ and $||A_j - A||_F$ and $||A_j - A||_2$ are both as small as possible among all rank j matrices.

We can use A_j as a pretty good approximation to the original data set, this data will account for a fraction

$$R = \frac{\sigma_1^2 + \cdots + \sigma_j^2}{\sigma_1^2 + \cdots + \sigma_k^2}$$

of the variance. This ratio can be very close to 1 for j much smaller than k. Often an appropriate choice of j is made by requiring that say 99% of the variance be accounted for, so we find $j < k$ so that $R \geq .99$.

Let's apply this to an image, viewed as a a data set. An image is just an array of pixels and a pixel (in a gray image) is just a number between 0 and 255. So a gray image is really just a matrix of $M \times N$ integers between 0 and 255.

We can take an image A and find a rank j image A_j just as above which should be a pretty good approximation to the original. This allows for compression.

Suppose you start with a 1000×1000 image. That is 1,000,000 pixels, i.e., each pixel is 1 byte of data (byte = 8 bits, a bit is just 0 or 1, $2^8 = 256$), so the image is 1,000,000 bytes.

Now A_50 is coded by 50 left singular vectors and 50 right singular vectors ($100 \times 1000 = 100,000$ bytes and 50 singular values (8 bytes each for a float), this is $104,000$ bytes, about $1/10^{\text{th}}$ the size of the original image.

This is all easily achieved in MATLAB:

```
% Read in the image
img = imread("siamese-cat.png");
```

An image is really a $M \times N \times 3$ array where every pixel is assigned a vector (R, G, B) and these are values between 0 and 255. We will flatten this to one dimension by just averaging the RGB values.

```
% Make the image grayscale by just averaging the RGB values
img_gray = sum(img,3)/3;
```

Next find the SVD decomposition of the data:

```
[U,S,V] = svd(img_gray);
```

Now choose which of the terms $\sigma_i \boldsymbol{u}_i \boldsymbol{v}_i^T$ to include. This is easy in MATLAB

$$\sigma_M \boldsymbol{u}_M \boldsymbol{v}_m^T + \cdots + \sigma_N \boldsymbol{u}_N \boldsymbol{v}_N^T = U(:, M:N) * S(M:N, M:N) * V(:, M:N)'$$

So the matrix A_j would be formed by taking $M = 1$ and $N = i$. Here is the function that builds and displays the reduced image

```
show = @(M,N) {
          img_gray_reduced = U(:,M:N)*S(M:N,M:N)*V(:,M:N)';
          % For imshow you need the correct format for the integers
          imshow(uint8(img_gray));
          figure();
          imshow(uint8(round(img_gray_reduced)));
};
```

Here are the results:

Original

First 20 singular values

Download image.

The full code is

```
% Read in the image
img = imread("siamese-cat.png");

% Display the original image
imshow(img);

% Make the image grayscale by just averaging the RGB values
img_gray = round(sum(img,3)/3);

% Get SVD for gray image
[U,S,V] = svd(img_gray);

show = @(M,N) {
          img_gray_reduced = U(:,M:N)*S(M:N,M:N)*V(:,M:N)';
          % For imshow you need the correct format for the integers
          imshow(uint8(img_gray));
          figure();
          imshow(uint8(round(img_gray_reduced)));
};
```

Download here.