Tutorial: Image Compression using the Singular Value Decomposition

Matteo Fasiolo

Image Compression

One usage of Singular Value Decomposition (SVD) is compressing large matrices.

We have learned that images are essentially matrices of numeric values when stored in computers.

Therefore, we can use SVD to compress images.

SVD

SVD of a matrix $M \in \mathbb{R}^{m \times n}$ finds the following three matrices:

- $U \in \mathbb{R}^{m \times r}$,
- $D \in \mathbb{R}^{r \times r}$ is a diagonal matrix with non-negative diagonal elements.
- $V \in \mathbb{R}^{n \times r}$,

where $r = \min(m, n)$ and such that

$$M = UDV^\top.$$

The diagonal elements of D are called singular values.

In numerical software (such as R or MATLAB), the singular values stored in D **are sorted in decreasing order**.

SVD Compression

Suppose singular values in D are sorted decreasingly, SVD can be used to construct an approximation of M:

$$M_1 = U_1 D_1 V_1^\top,$$

where

- $U_1 = U_{[1:m,1:r_1]}$,
- $D_1 = D_{[1:r_1,1:r_1]}$,
- $V_1 = V_{[1:n,1:r_1]}$.
- r_1 is a positive integer smaller than r.

Loading Images

Install the `imager` package if you have not.

```
install.packages("imager")
```

Load an image into the matrix `M`.

```
library(imager)
img <- load.image("UoB.jpg")
img <- grayscale(img)
M <- as.matrix(img)
```

Now `M` should contain a matrix whose entries are pixel values of the image.

Checking out the Image

To plot we need to convert `M` back to an image:

```
plot(as.cimg(M), axes=FALSE)
```

Note: just typing `plot(M)` does not quite produce what we want (try it!).

Check out how much memory does it take to store this image in `M`:

```
# fill out the blank.
size1 <- ______
print(paste("size:", size1, "bytes"))
```

You should be able to compute this just by considering the fact that elements of `M` are double precision floating-point number (equivalent to `double` in C).

The number you compute should be similar, but not identical, to the output of `object.size(M)`.

Compression

Now, use built-in `svd` function to obtain U, D, V for `M`.

- Hint: `?svd`

Double check you have used `svd` correctly by computing the Frobenious distance between M and its SVD-reconstructed version $UDV^\top$.

The Frobenious norm of an $(n \times m)$ matrix A is:

$$||A||_F = \left(\sum_{i=1}^{n} \sum_{j=1}^{m} A_{ij}^2 \right)^{\frac{1}{2}}.$$

The Frobenious distance between two matrices is $\text{dist}_F(A, B) = ||A - B||_F$.

Let us $r_1 = 100$.

Construct U_1, D_1, V_1 using U, D, V and r_1.

Construct the corresponding M_1 using U_1, D_1, V_1.

Plot the compressed image:

```
plot(as.cimg(M_1))
```

Examine the Compression

Calculate how much memory is needed to store the compressed image:

```
# Fill out the blank
size2 <- ____
print(paste("size:", size2, "bytes"))
```

What is the compression rate `size1/size2`?

Experiment with different values of r_1, how do the compression rate and the quality depend on this parameter?

Pick a value of r_1 that seems to provide a good compromise.

Evaluating Approximation Error and Compression Rate

Let M_{r_1} be an SVD-based approximation of M resulting from r_1 singular values.

Why is M_{r_1} called an approximation of M?

- Note that we can write $M = M_{r_1} + (M - M_{r_1})$.
- Consider the approximation error $M - M_{r_1}$ and the relative error

$$\text{Rel\_error}(M, M_{r_1}) = \frac{\text{dist}_F(M, M_{r_1})}{\text{dist}_F(M, 0)}$$

  where 0 is just a matrix of zeros of the same size as M.
- Define a grid of possible values for r_1 (recall that $r_1 \in \{1, 2, \ldots, r\}$).
- Plot the compression rate and the relative error as functions of r_1.

What value of $r1$ would you use if you can not use more than 100KBytes to store M_1?

Consider the quantity

$$z_{r1} = \frac{\sum_{i=r_1+1}^{r} d_i^2}{\sum_{j=1}^{r} d_i^2}$$

Compute z_{r_1} for $r_1 = 1, \ldots, r$ and compare it with the relative error you computed previously (you will need to use the same grid).

Interpret your results: how do the singular values relate to the approximation error?

Challenge: compute the vector z_{r_1} for $r_1 = 1, \ldots, r$ without using any loop (or `lapply`).

Hint: you might need the functions `rev()` and `cumsum()`, see the R documentation.

Finally, note that our construction M_{r_1} is the best "low rank approximation" of M in terms of Frobenius norm.

- Read: https://en.wikipedia.org/wiki/Low-rank_approximation
- Low rank approximation is a classic problem in machine learning.