Image Compression Using SVD

Himank Jain

March 12, 2019

Synopsis:

In the era of big data where data is being accumulated with high velocity and veracity with sources like internet, social networking sites, log files, banking sector, etc. Most storage in big data contents is occupied by images & video files. In this context the concept of image compression may be considered for data storage so that enormous amount of memory space can be saved. In this document, we have followed Singular Vector Decomposition from linear algebra to compress the image by considering both gray scale and color images. Further we have analyzed the effectiveness of compression by considering different sets of singular vectors in SVD. This work can be useful in data storage and transfer of data over the internet. This document contains a guide on image compression using Single Value Decomposition in R. It requires basic prior knowledge in Linear Algebra and R programming language.

SINGLE VALUE DECOMPOSITION

In linear algebra, the singular-value decomposition (SVD) is a factorization of a matrix.SVD decomposes a $m * n$ matrix into three components i.e UDV^* where U is of size $m * m$, D is of size $m * n$ and is diagonal, and V^* is of size $n * n$. The columns of U and the columns of V are called the left-singular vectors and right-singular vectors of the original matrix.,

$$M = UDV^*$$

Calculating SVD:

Singular value decomposition can be can be computed using the following observations: * The left singular vectors of M are a set of orthonormal eigenvectors of $MM*$ * The right-singular vectors of M are a set of orthonormal eigenvectors of $M * M$ * The non-zero singular values of M (found on the diagonal entries of D) are the square roots of the non-zero eigenvalues of both $M * M$ and $MM*$.

For a better understanding and detailed guide on calculating SVD of a matrix refer:
SVD examples

Applications of SVD

Applications that employ the SVD include computing the pseudoinverse, least squares fitting of data, multivariable control, matrix approximation, and determining the rank, range and null space of a matrix and **image compression**.

Now that we know a little about Single Value Decomposition, it's time to move onto image compression in R...

Image compression in R using SVD:

- Before We think of compressing image, It's important to understand basic concepts of colors in digital images. The colors white (or yellow or any other color) that you see in your monitor is not exactly how it appears. Infact there is actually no white or yellow pigment in your screen.What you are seeing is actually a mixture of RGB (red,green and blue) displayed by small pixels in your screen.These red,

green, and blue pixels range in saturation on a scale of 0 to 255; 0 being completely off, and 255 being completely on.

- A grayscale image is one in which the value of each pixel is a single sample representing only an amount of light, that is, it carries only intensity information.
- Here, we will look at two ways of compressing an image in R:
- compression in grayscale images
- compression in color images

1.Gray scale compression:

Original Image :*FLOWERS*

Getting grayscale of image i.e. discarding RGB colors:

```
megray<-grayscale(image)
plot(megray,axes=FALSE,main="Grayscale image")
```

Grayscale image

Now 'megray' stores the matrix that contains the image data in grayscale

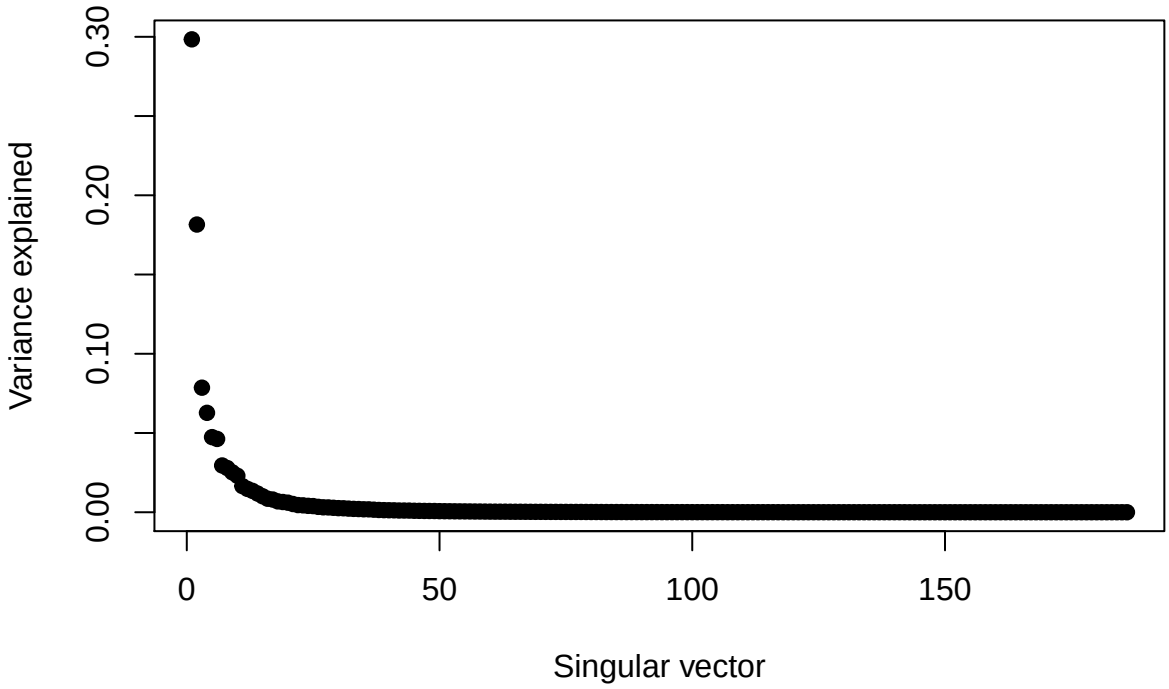
Getting Single Value decomposition of 'megray' matrix:

```
svd1<-svd(scale(megray))
str(svd1)
```

```
## List of 3
##  $ d: num [1:186] 122.4 95.5 62.8 56.1 48.8 ...
##  $ u: num [1:271, 1:186] 0.126 0.125 0.123 0.12 0.118 ...
##  $ v: num [1:186, 1:186] -2.85e-02 -2.25e-02 -1.45e-02 -7.04e-03 -1.12e-05 ...
```

We can observe that svd contains three components i.e. U,D and V^*. If we plot The D component of our svd as $D^2/sum(D^2)$ we get the variance explained by each singular vector in the image. Plotting to see variance explained:

```
plot(svd1$d^2/sum(svd1$d^2),pch=19,xlab="Singular vector",ylab="Variance explained")
```



As we can see that the most of the variance is explained by first 25 singular vectors in this image.After 25 vectors the variance is very low and almost steady. After about 50 vectors it's miniscule. Hence if we only keep them and discard the others. we can compress the image without losing much of the quality. Also we already know that: $M = UDV^*$ So let's reduce the number of vectors to 5,10,25,35 and 50:

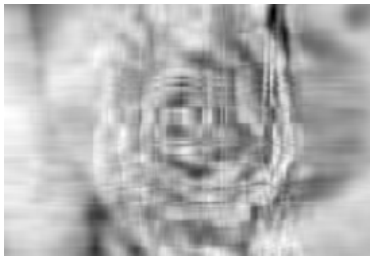
```
#matrix multiplication of U,D and V*
#including only first 5 singluar vectors
approx5<-svd1$u[,1:5] %*% diag(svd1$d[1:5]) %*% t(svd1$v[,1:5])
#including only first 10 singular vectors
approx10<-svd1$u[,1:10] %*% diag(svd1$d[1:10]) %*% t(svd1$v[,1:10])
#including only first 20 singular vectors
approx25<-svd1$u[,1:25] %*% diag(svd1$d[1:25]) %*% t(svd1$v[,1:25])
#including only first 20 singular vectors
approx35<-svd1$u[,1:35] %*% diag(svd1$d[1:35]) %*% t(svd1$v[,1:35])
#including only first 20 singular vectors
approx50<-svd1$u[,1:50] %*% diag(svd1$d[1:50]) %*% t(svd1$v[,1:50])
```

If we now plot the above reduced matrices as images We can see 4 images for above above approximations:

```
par(mfrow=c(2,3),mar=c(1,1,1,1))
#plotting for reduced images
plot(as.cimg(approx5),main="(a) 5 singular Vectors",axes=FALSE)
plot(as.cimg(approx10),main="(b) 10 singular Vectors ",axes=FALSE)
plot(as.cimg(approx25),main="(c) 25 singular Vectors",axes=FALSE)
plot(as.cimg(approx35),main="(c) 35 singular Vectors",axes=FALSE)
plot(as.cimg(approx50),main="(c) 50 singular Vectors",axes=FALSE)
plot(as.cimg(megray),main="(d) Full image",axes=FALSE)
```

(a) 5 singular Vectors

(b) 10 singular Vectors



(c) 25 singular Vectors

(c) 35 singular Vectors

(c) 50 singular Vectors

(d) Full image

The original image matrix has rank 187 and compressed image matrices have ranks 5,10,25,35 and 50. We can observe that even after reducing the rank by more than 150,we still have almost same quality image.

2.Color image Compression:

Loading the image:

```
library(imager)
par(c(1,1))
```

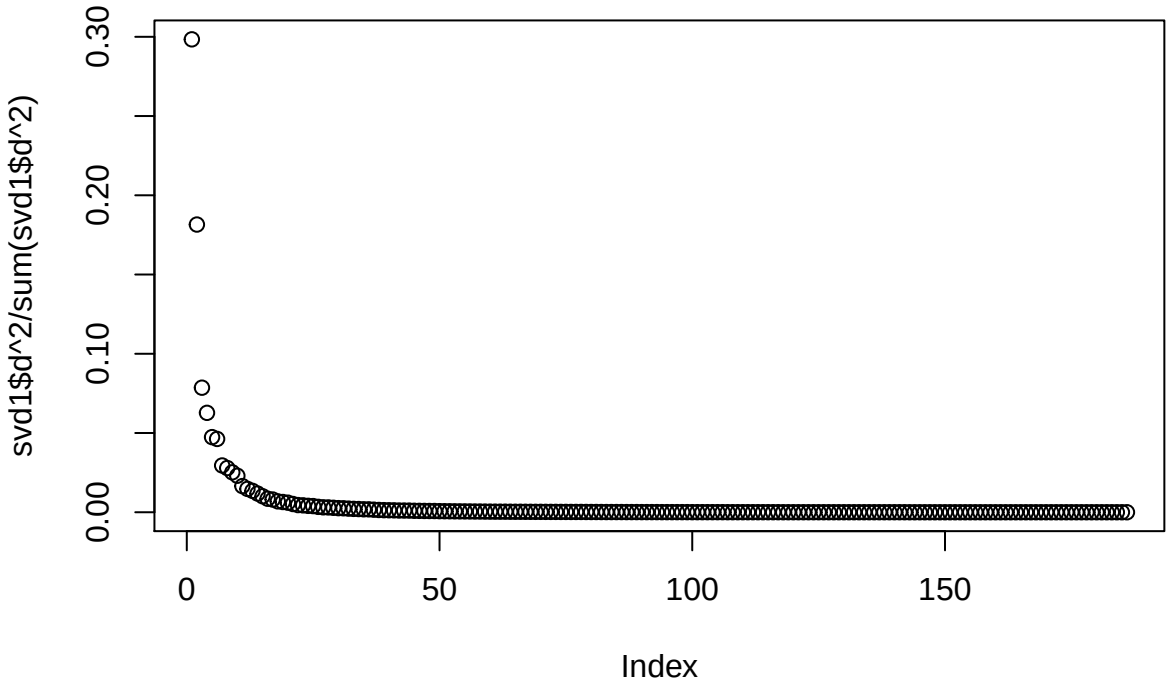
```
## NULL
```

```
image<-load.image("flower.jpg")
plot(image,main="FLOWER",axes=FALSE)
```

FLOWER

Below we see that most of the variance is explained by first 25 vectors of the image:

```
svd1<-svd(scale(grayscale(image)))
plot(svd1$d^2/sum(svd1$d^2))
```



The original image is a mixture of three images with *red*,*green* and *blue* components respectively. Separating the RGB components:

```
R = image[,,1]
G = image[,,2]
B = image[,,3]
```

As explained above the original image is nothing but a mixture of below three images:

```
par(mfrow=c(1,3))
```

```
par(mfrow=c(1,3),mar=c(1,1,1,1))
cscale <- function(v) rgb(v,0,0)
grayscale(image) %>% plot(colourscale=cscale,rescale=FALSE,axes=FALSE)

cscale <- function(v) rgb(0,v,0)
grayscale(image) %>% plot(colourscale=cscale,rescale=FALSE,axes=FALSE)

cscale <- function(v) rgb(0,0,v)
grayscale(image) %>% plot(colourscale=cscale,rescale=FALSE,axes=FALSE)
```

Single Value Decomposition of the **RGB** images and grouping them into one list.

```
svdR<-svd(R)
svdG<-svd(G)
svdB<-svd(B)
svdRGB<-list(svdR,svdG,svdB)
```

Computing UDV^* for every sub color matrix and combining them into one 3-D array. Repeating the process for first 5,10,25 and 50 vectors.

```
par(mfrow=c(2,3),mar=c(1,1,1,1))
for(j in c(5,10,25,35,50)){
comp <- sapply(svdRGB, function(i){
        compressed = i$u[,1:j] %*% diag(i$d[1:j]) %*% t(i$v[,1:j])
}, simplify = 'array')
comp<-as.cimg(comp)
plot(comp,axes=FALSE,main=paste("Rank=",j))
}
```

```
## Warning in as.cimg.array(comp): Assuming third dimension corresponds to
## colour

## Warning in as.cimg.array(comp): Assuming third dimension corresponds to
## colour

## Warning in as.cimg.array(comp): Assuming third dimension corresponds to
## colour
```

```
## Warning in as.cimg.array(comp): Assuming third dimension corresponds to
## colour

## Warning in as.cimg.array(comp): Assuming third dimension corresponds to
## colour
```

```
plot(image,axes=FALSE,main="Rank=187")
```

Rank= 5

Rank= 10

Rank= 25

Rank= 35

Rank= 50

Rank=187

Results:

Image compression is minimizing the size in bytes of a graphics file without degrading the quality of the image to an unacceptable level.From above observations we conclude that Single Value Decomposition can be used to compress images by reducing the image matrix ranks. We can preserve most of the quality in the image even after compressing it to a miniscule amount. We observe this both in grayscale images and color images.The reduction in file size allows more images to be stored in a given amount of disk or memory space. It also reduces the time required for images to be sent over the Internet or downloaded from Web page. Ever since the dawn of internet graphical data generation has increased exponentially. The emergence of social media has also played a huge role in this. Hence it has become important that we optimize our data in terms of size and compress files as much as possible without losing much value out of it.