Lecture 27 - Page Rank, Singular Value Decomposition, and Image Compression

December 7, 2022

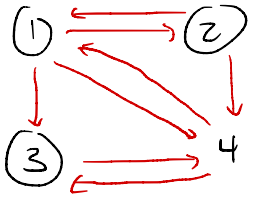
Goals: Look at some fun applications of eigenvalues!

PageRank (See Appendix D of the text for more information)

When one does a Google search for a phrase, the output of the search algorithm is a list of webpages containing that phrase. The output needs to be ordered in some way before being displayed to the user. The method of doing so is the key innovation that made Google so successful. It involves assigning a value to each webpage and then displaying the pages in decreasing order. The higher the value, the more "important" the webpage. We now explain a simplified version of the algorithm:

Let n be the number of webpages on the Internet (n is certainly in the billions). For the ith webpage, Google wants to assign a value $v_i > 0$ so that $\sum_{i=1}^{n} v_i = 1$. Their insight was that v_i should be determined only by which page links to which and not about the contents of a specific webpage. That is, the v_i's should be a weighted sum of the values of v_j for which the jth webpage has a link to the ith one. In other words, we want $v_i = \sum_{i=1}^{n} w_{ij} v_j$ for some weights w_{ij}. We will continue the explanation by working through an example:

Ex $n = 4$ web pages



Adjacency Matrix

$$A = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{bmatrix} 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 1 & 0 \end{bmatrix} \end{array}$$

ij^{th}-entry is # of edges from page j to page i

Weights matrix: $W = (w_{ij})$ = columns of A ÷ by their sum

$$W = \begin{bmatrix} 0 & 1/2 & 0 & 1/2 \\ 1/3 & 0 & 0 & 0 \\ 1/3 & 0 & 0 & 1/2 \\ 1/3 & 1/2 & 1 & 0 \end{bmatrix}$$

Markov matrix: cols add to one → $W^k \approx W^r$ $k > r$ stabilizes

$v_1 = w_{11}v_1 + w_{12}v_2 + w_{13}v_3 + w_{14}v_4$

$v_2 = w_{21}v_1 + w_{22}v_2 + w_{23}v_3 + w_{24}v_4$

$\vec{v} = W\vec{v} \rightarrow \vec{v}$ is eigenvector of W w/ $\lambda = 1$

1-eigenspace: find $N(W - 1 I_4)$

$$N(W - 1 I_4) = N\begin{bmatrix} -1 & 1/2 & 0 & 1/2 \\ 1/3 & -1 & 0 & 0 \\ 1/3 & 0 & -1 & 1/2 \\ 1/3 & 1/2 & 1 & -1 \end{bmatrix} = \text{span}\left\{\begin{bmatrix} 6 \\ 2 \\ 7 \\ 10 \end{bmatrix}\right\}$$

page 4 is most important, then page 3, page 1, then page 2

$$\vec{v} = \frac{1}{25}\begin{bmatrix} 6 \\ 2 \\ 7 \\ 10 \end{bmatrix}$$

Singular Value Decomposition

We have discussed that not every square matrix has eigenvalues/eigenvectors. Certainly, we do not have any notion of eigenvalues for non-square $m \times n$ matrices. There is a related idea, however, known as the singular value decomposition. The idea is that we want an orthonormal basis $\mathbf{w}_i$ of $\mathbb{R}^m$ and an orthonormal basis $\mathbf{v}_i$ of $\mathbb{R}^n$ so that

$$A\mathbf{v}_i = \sigma_i \mathbf{w}_i.$$

Recall that for any $m \times n$ matrix A, the $m \times m$ matrix $AA^\intercal$ and the $n \times n$ matrix $A^\intercal A$ are both symmetric. Hence, by the spectral theorem, each of these matrices will emit an orthogonal basis of eigenvectors of $\mathbb{R}^m$ and $\mathbb{R}^n$ respectively. What's even more interesting is that the **nonzero** eigenvalues of $A^\intercal A$ and $AA^\intercal$ are the same! Using these eigenvalues and eigenvectors, we can in fact form a very similar decomposition of A that mimics the one in the spectral theorem.

Theorem 27.3.3: (Singular Value Decomposition (SVD)) For **every** $m \times n$ matrix A we can find:

- an $m \times n$ diagonal matrix D which has diagonal entries $d_{ii} = \sigma_i$ where $\sigma_1 \geq \sigma_2 \geq \cdots \geq 0$;
- an $m \times m$ orthogonal matrix U and an $n \times n$ orthogonal matrix V

for which

$$A = UDV^{-1} = UDV^\intercal$$

The diagonal matrix D is uniquely determined. The numbers σ_i are called the **singular values** of A.

Since the SVD exists for **any** matrix, this technique is often regarded as the primary method for doing any numerical computations with matrices. Note that if the A is a symmetric matrix, then the SVD is almost the same as the decomposition from the Spectral theorem; the singular values of A are the absolute value of the eigenvalues of A.

Example 1: The matrix $A = \begin{bmatrix} 2 & 1 & 1 \\ -2 & 1 & 3 \end{bmatrix}$ has SVD

$$\begin{bmatrix} 2 & 1 & 1 \\ -2 & 1 & 3 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \begin{bmatrix} \sqrt{14} & 0 & 0 \\ 0 & \sqrt{6} & 0 \end{bmatrix} \begin{bmatrix} -\sqrt{2/7} & 1/\sqrt{14} & 3/\sqrt{14} \\ \sqrt{2/3} & 1/\sqrt{6} & 1/\sqrt{6} \\ 1/\sqrt{21} & -4/\sqrt{21} & 2/\sqrt{21} \end{bmatrix}.$$

Example 2: The matrix $A = \begin{bmatrix} 2 & -1 \\ -1 & 3 \\ -1 & 1 \end{bmatrix}$ has SVD

$$\begin{bmatrix} 2 & -1 \\ -1 & 3 \\ -1 & 1 \end{bmatrix} = \begin{bmatrix} -7/\sqrt{195} & 2\sqrt{2/13} & \sqrt{2/15} \\ 11/\sqrt{195} & 3/\sqrt{26} & -1/\sqrt{30} \\ \sqrt{5/39} & -1/\sqrt{26} & \sqrt{5/6} \end{bmatrix} \begin{bmatrix} \sqrt{15} & 0 \\ 0 & \sqrt{2} \\ 0 & 0 \end{bmatrix} \begin{bmatrix} -2/\sqrt{13} & 3\sqrt{13} \\ 3\sqrt{13} & 2/\sqrt{13} \end{bmatrix}$$

Image Compression

One application of SVD in practice is for compressing images. On a computer, a grayscale image can be viewed as $m \times n$ matrix A, where m is the height of the image and n the width (in pixels). The ij-entry of A is a value between 0 and 255, where 0 means that pixel is the color white and 255 means that pixel is the color black. For example, a grayscale wallpaper image on a 1080p monitor would be a 1080×1920 matrix.

Having a lot of high quality images would take up a lot of memory. Often, images will get compressed to lower the file size. We can do this using SVD!

The guiding principal is that the larger the singular values encode the most important information of the matrix A. So if we remove the p smallest singular values in the SVD $A = UDV^\top$, we will get $A' = U'D'V'^\top$, where U' is an $m \times (m-p)$ matrix, D' is a $(m-p) \times (n-p)$ diagonal matrix, and V' is a $n \times (n-p)$ matrix. The resulting A' will also be an $m \times n$ matrix, but its **rank** (dimension of its column space) will be less than the rank of A. So in some sense, it contains less information than A while retaining its size, which will result in a smaller file size.

Example 3: Here is a grayscale image of my dog Pikachu:

The size of the matrix is 3363×3024. We will use python to perform our calculations.

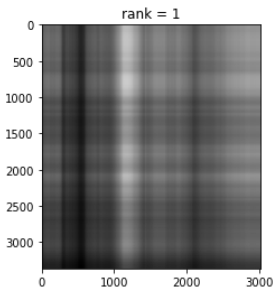
We will find the SVD of our image, and see what we get when we only consider the first (largest) singular value.

```
og = np.array(true_color.getdata()).reshape(true_color.size[1],true_color.size[0],3)
gray = ImageOps.grayscale(true_color)
gray = np.array(gray.getdata()).reshape(gray.size[1],gray.size[0])
gray
```

```
array([[198, 189, 187, ..., 206, 208, 205],
       [183, 175, 172, ..., 192, 194, 193],
       [182, 178, 176, ..., 187, 187, 189],
       ...,
       [ 72,  74,  75, ...,  39,  41,  50],
       [ 73,  78,  82, ...,  42,  43,  50],
       [ 76,  86,  90, ...,  46,  46,  51]])
```

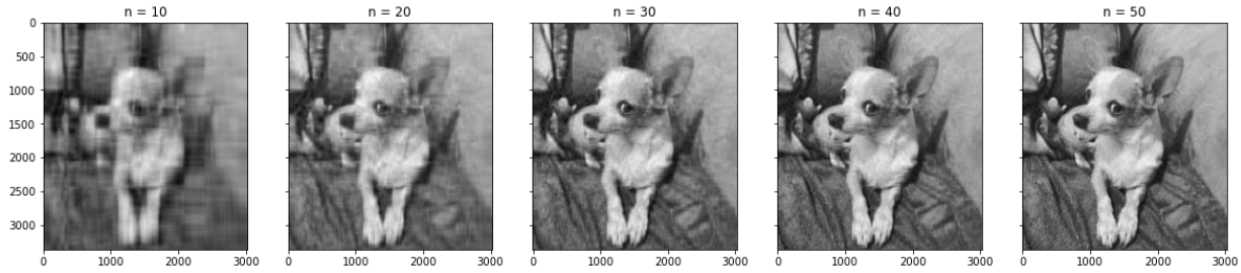
```
#Calculate singular value decomposition
U, sigma, V = np.linalg.svd(gray)

#Reconstruct with most prominent SV
rank_1 = np.matrix(U[:,:1]) * np.diag(sigma[:1]) * np.matrix(V[:1,:])
plt.imshow(rank_1, cmap='gray')
plt.title('rank = %s' %(np.linalg.matrix_rank(rank_1)))
```

```
Text(0.5, 1.0, 'rank = 1')
```



As you can see, the corresponding matrix has rank 1; indeed, all columns of pixels in this image are multiples of the same vector. We also can't tell at all what this is a picture of. Let us see what happens when we consider the first 10, 20, 30, 40, and 50 singular values.

```
fig,ax = plt.subplots(1,5,figsize = (20,10),sharey=True)
for i in range(1,6):
    reconstruct = np.matrix(U[:,:10*i])*np.diag(sigma[:10*i])*np.matrix(V[:10*i,:])
    #ax = fig.add_subplot(1,4,i,True)
    ax[i-1].imshow(reconstruct,cmap='gray')
    title = 'n = %s' %(10*i)
    ax[i-1].set_title(title)
```



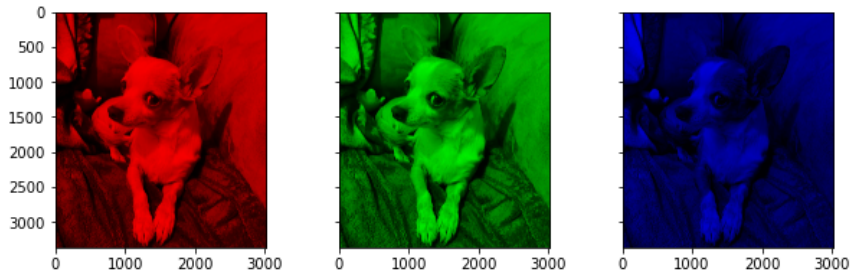
Even just using 10 singular values out of the 3363 is enough to tell us what the image is; pretty cool! Finally, let us use the first 200 singular values and look at the resulting image:

```
reconstruct = np.matrix(U[:,:200])*np.diag(sigma[:200])*np.matrix(V[:200,:])

#Compare images
fig, ax = plt.subplots(1,2,figsize=(20,10),sharey=True)
ax[0].imshow(gray,cmap='gray')
ax[0].set_title('Original')
ax[1].imshow(reconstruct,cmap='gray')
ax[1].set_title('Rank 200');
```

The pictures look almost identical! The original images is storing $3363 \times 3024 = 10,169,712$ numbers, where as our rank 200 approximation is storing $3363 \times 200 + 200 + 200 \times 3024 = 1,277,600$ numbers, or about 12.6% of the original image.

Example 4: Color images are stored slightly differently. A color image is viewed as an $m \times n$ matrix like the grayscale image, but each entry of this matrix is actually a 3-vector, whose elements give the RGB (red, green, and blue) intensities of that pixel which then correspond to the color you see. In this form, we can't quite do what we did in Example 3. However, we just need a simple adjustment: first extrapolate the red, green, and blue matrices R, G, and B respectively, and then perform the operations as in Example 3 on each of them. We then combine the resulting matrices to form our reconstructed image. Here's an example with a color photo of my dog.

```
red_img = og[:,:,0]/255
green_img = og[:,:,1]/255
blue_img = og[:,:,2]/255

fig,ax = plt.subplots(1,3,figsize=(10,3),sharey=True)
count = 0
for color in [r,g,b]:
    ax[count].imshow(color)
    count+=1
```




```
#SVD for each color matrix
Ur, Sr, Vr = np.linalg.svd(red_img)
Ug, Sg, Vg = np.linalg.svd(green_img)
Ub, Sb, Vb = np.linalg.svd(blue_img)

#Reconstruct each color image using first 200 singular values
small_red = np.matrix(Ur[:,:200])*np.diag(Sr[:200])*np.matrix(Vr[:200,:])
small_green = np.matrix(Ug[:,:200])*np.diag(Sg[:200])*np.matrix(Vg[:200,:])
small_blue = np.matrix(Ub[:,:200])*np.diag(Sb[:200])*np.matrix(Vb[:200,:])

#Combine them back to an image with smaller rank
small_image = np.zeros(og.shape)
small_image[:,:,0] = small_red
small_image[:,:,1] = small_green
small_image[:,:,2] = small_blue
small_image[small_image < 0] = 0 #Account for rounding errors
small_image[small_image > 1] = 1
```

```
#Compare images
fig, ax = plt.subplots(1,2,figsize=(20,10),sharey=True)
ax[0].imshow(og)
ax[0].set_title('Original')
ax[1].imshow(small_image)
ax[1].set_title('Rank 200')
```

```
Text(0.5, 1.0, 'Rank 200')
```