SVD Exercises
Comp Stat 2014

April 28, 2014

Thanks to Isaac Lee and Jesse Chan for providing most of the content and text of this exercise.

1 The Fundamental Equation

The genius and power of the singular value decomposition is most commonly used in one of two ways: low-rank approximation of data and principal component analysis. In this exercise we will explore those two ways to look at the SVD. Before starting, though, let's make sure we understand the key equation.

Let $A \in \mathbb{R}^{n \times m}$. Then the singular value decomposition theorem says that there exist $U \in \mathbb{R}^{n \times n}$, $\Sigma \in \mathbb{R}^{n \times m}$ and $V \in \mathbb{R}^{m \times m}$ such that Σ is composed of decreasing values along its diagonal, U and V are unitary, that is their columns are vectors in $\mathbb{R}^n$ and $\mathbb{R}^m$ respectively which are all of length 1 and orthogonal to each other, and

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

Show that

$$A = \sum_{i=1}^{min(n,m)} s_i u_i v_i^T, \tag{1}$$

where s_i is the i-th value along the diagonal of Σ and u_i and v_i are the column vectors of U and V.

2 Low-rank data approximation: Image Compression

The first way to look at the SVD, as given in Equation (1), is as a sum of ***data*** matrices of the form $u_i v_i^T$ (rank 1), each of which has importance s_i to the final outcome. A fun way to demonstrate this is by applying it to one of the most common data matrix forms around: pictures.

In this exercise, we seek to compress an image while keeping most of its features intact. We will use SVD to do so. Consider the following cacti image of pixel dimensions $h \times w = 400 \times 600$.

For a 24-bit, RGB image, there are three channelsred, green, and blueeach with 8 bits (0 - 255). Thus, we construct three matrices A_R, A_G, $A_B \in \mathbb{R}^{h \times w}$, whose (i, j)-th entry is an integer between 0 and 255 denoting the intensity of that color in the (i, j)-th pixel. (From now on, we hide the subscripts R, G, and B for simplicity.) For each channel, we then find the SVD of its matrix, i.e. $A = U\Sigma V^T$, where $U \in \mathbb{R}^{h \times h}$, $\Sigma \in \mathbb{R}_{h \times w}$, and $V \in \mathbb{R}^{w \times w}$.

1. Open the file *svd_exercise.m* and complete the sections indicated by PROBLEM. The problems include:

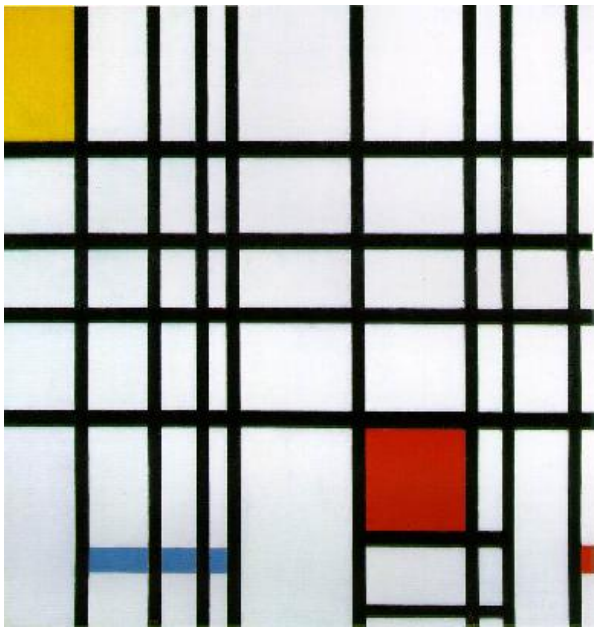
   (a) Perform SVD on the three color channels.

   (b) Plot the singular values to see where the information content begins to wane.

(c) Choose a set of k's which you think will demonstrate the tapering off of the quality of the image.

(d) Plot pictures and note the error.

2. When you feel satisfied with your analysis of *cacti.jpg*, look at the picture *ryb.jpg*, shown below. Do you think this will require bigger or smaller k? Go ahead and run the program on it. What explains the difference? Now try it with *ryb_twist.jpg*. Does that change your answer?



3. Try it with a picture of your own. How does it turn out?

4. From above, we see that we can approximate each color channel matrix A with A_k , a sum of k rank-one matrices, and indeed, A_k is the best approximation of A in the Frobenius-norm sense. Now, A requires storing hw entries. In comparison, how many entries does $A_k = \sum_{i=1}^{k} s_i u_i v_i^T$ require in terms of h, w, and k? For the cacti example, evaluate the ratio in the number of entries that need to be stored for A_k to A for $k = 5, 10, 20, 40, 80$. Since this ratio shouldnt

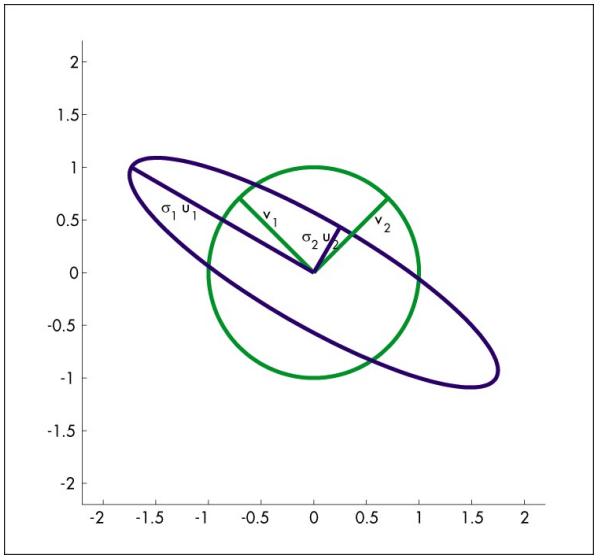
exceed 1, we have an upper bound on k. Express it in terms of h and w, and evaluate it for the cacti example.

3 Principal Component Analysis: Analyzing Congressional Votes

The second way to look at the SVD, again in the form we proved in Equation (1), is as the sum of ***transform*** matrices $u_i v_i^T$. Specifically, if you take any vector x, then

$$Ax = \left(\sum_{i=1}^{k} s_i u_i v_i^T \right) x = \sum_{i=1}^{k} s_i u_i v_i^T x.$$

That is, for each i, you measure how much x points in the v_i direction ($v_i^T x$), multiply that amount by s_i, and then output a vector with that length in the direction of u_i. Finally add them all up. A effectively inflates the importance of the parts of x which point in the direction of the first few v_i's (pointing in a new direction) and tends to ignore those parts of x which point in the direction of the last v_i's. That is, the first few v_i's tend to summarize the most important trends in your data. The picture below tries to show this dynamic for unit vectors in two dimensions.



The lecture for Wednesday will cover this in more detail, but since we will not have an exercise then, here are a couple scripts to run to see how this can work.

1. Run the script *getCongress.py*. This retrieves the 2012 congressional voting record from the web, parses it, and creates the files *names.txt*, *parties.txt*, and *votes.txt*.

2. Write a script which performs the SVD of the voting record and extracts v_1 and v_2. Each entry in the v_i's corresponds to a congress person, so we have a coordinate $(v_1(j), v_2(j))$ for each. Create a 2-d scatter plot of these coordinates, with republicans colored red and democrats blue. Try labeling congress people you know to come up with interpretations for the two axes.

3. If you run out of time, you can run the script *congressSVD.m*. This performs the analysis in part 2.