CS 4220 / MATH 4260: Project 1

Instructor: Anil Damle
Due: March 6, 2019

Policies

For this project, each individual will be turning in their own write up and code. However, you may work in groups of up to three on the project, this includes helping each other write and debug code, and reading and editing each others reports. This is a bit beyond the level of collaboration allowed on the HW. Within a group you can look at "solutions" and discuss them in detail. However, you cannot simply copy each others. Furthermore, please list your collaborators (if any) in your project report. Part of the purpose of this project is to provide you with an opportunity to practice writing more free form reports and carefully choosing what plots, results, etc. are needed to convince us your solution is correct. Therefore, some of the goals are leading, but do not give a concrete list of exactly what has to be included. Please submit your code along with the report via CMS.

Preamble

For this project, we are going to work with images, and a simple form of so-called matrix completion. One note up front, we have picked an application (this type of algorithms has broad ranging applications beyond imaging) and algorithms that you have all of the tools to complete. The state of the art in this field involves more complicated algorithms and for images in particular there are modifications that can be made to improve the results. Furthermore, I have made some slight modifications to the underlying images to ensure things work out reasonablly here. If you are curious about this area I would be happy to chat more about it.

For the purposes of this project, all of the images we will work with are gray-scale and are represented as matrices with real valued entries between 0 and 1 (0 for black and 1 for white). So, an image that is $n_1 \times n_2$ pixels will be represented by an $n_1 \times n_2$ matrix. These techniques can be expanded to deal with color images, but here we will focus on the linear algebra. When producing plots/images (*e.g.* with the "imagesc" command in Matlab) you may want to set the color space to gray so that it looks as you would expect. In Matlab this can be accomplished via the command "colorspace gray".

Compressing images

Given a matrix $A \in \mathbb{R}^{n_1 \times n_2}$ we may want to store a compressed representation of it. One way to do this is to store a low-rank representation of A, and this can be accomplished using the SVD. Say we compute the SVD

$$A = USV^T,$$

where $U \in \mathbb{R}^{n_1 \times p}$, $V \in \mathbb{R}^{n_2 \times p}$, $S \in \mathbb{R}^{p \times p}$, and $p = \min\{n_1, n_2\}$. Let U_k denote the first k columns of U, V_k denote the first k columns of V, and S_k denote the upper left $k \times k$ block of S. Now, we can write $A \approx U_k S_k V_k^T$ and, in fact, this is the optimal approximation of A in either the 2 or Frobenius norm. This means that of all rank k matrices (those with k non-zero singular values), the one closest to A in 2 or Frobenius norm is $U_k S_k V_k^T$. The error of approximation is determined

by the singular values of A, specifically

$$\left\|A - U_k S_k V_k^T\right\|_2 = \sigma_{k+1} \qquad \text{and} \qquad \left\|A - U_k S_k V_k^T\right\|_F = \left(\sum_{i=k+1}^{p} \sigma_i^2\right)^{1/2}.$$

So, if the singular values of A decay rapidly there may be a good low rank approximation for A, and if k is small enough, storing U_k, V_k, and S_k may be cheaper than storing A.

Now, while we have not talked about them, there are good algorithms to compute the SVD. However, it may not always be possible to use these algorithms directly in certain settings. So, we now need to think of low rank approximations more broadly. We can write any rank k approximation of A as WZ^T where $W \in \mathbb{R}^{n_1 \times k}$ and $Z \in \mathbb{R}^{n_2 \times k}$. Our low-rank approximation problem can then be cast as the optimization problem

$$\min_{W,Z} \left\|A - WZ^T\right\|_F^2.$$

Unfortunately, solving this problem given A can be a bit tricky if we do not resort to the SVD. One possible approach is known as alternating least squares. It is based on the observation that if we fix W or Z then we can solve for the other factor via a (matrix) least-squares problem. This gives Algorithm 1, for our purposes we will always run it for a small, fixed number of iterations rather than checking for convergence.

Algorithm 1 Alternating least squares, without regularization

1: **initialize** $W \in \mathbb{R}^{n_1 \times k}$, $Z \in \mathbb{R}^{n_2 \times k}$
2: **while** not converged **do**
3:
$$Z \leftarrow \text{argmin}_Z \left\|A - WZ^T\right\|_F^2$$
4:
$$W \leftarrow \text{argmin}_W \left\|A - WZ^T\right\|_F^2$$
5: **end while**
6: **return** W, Z

First set of goals

- The first goal of this project is to implement a way to compute QR factorizations and use that building block to implement this algorithm. Your QR factorization routine should take in a $n \times k$ matrix B with $k \leq n$ and returns a $n \times k$ matrix Q with orthonormal columns (or sufficient information to be able to apply it to a vector), and a $k \times k$ upper triangular matrix R. Please demonstrate that your algorithm behaves as expected (this means both that you are getting a valid QR factorization as output and that it achieves the desired computational scaling). **Please explain which variant of QR factorization you implemented, demonstrate the expected scaling, and explain how you tested your implementation.**
- Using your QR factorization implement Algorithm 1. This is written as a matrix least squares problem, but think about the Frobenius norm and how you might be able to split it up into problems we know how to solve. **Please explain how you do this in the project report.** Practically, it is possible to then block some of these operations together for efficiency; you may certainly take advantage of this if you wish.

- Using your implementation, load the file Cornell_seal_bw.mat which has an image stored in the matrix C and try to compute the best rank 75 approximation of the image. Compare this with the result of the best rank 75 image from the SVD (you can use a built in routine in either Matlab or Julia to compute this), what do you observe qualitatively and quantitatively?

What about if we do not know all the entries of our image?

In the preceding section, alternating least squares seems like a roundabout way to get at something we know how to compute. But now we will explore a setting where we cannot compute the SVD as we might like. Specifically, we will consider being given an image A where we only know the true values of some subset of the pixels, the rest are unobserved. In this case we can still use a variant of alternating least squares to try and recover an underlying low-rank approximation that should roughly resemble our image.

The two key differences are that we will add regularization and only measure error on the observed pixels. Let Ω be a set of (i, j) pairs that represents the set of pixels we observe in the image. So, for example, if $\Omega = \{(1, 2), (10, 20)\}$ then we are only observing the $(1, 2)$ and $(10, 20)$ pixels (matrix entries) of the image. We now define a restricted variant of the Frobenius norm as follows

$$\|A - B\|_{F,\Omega}^2 = \sum_{(i,j)\in\Omega} (A_{i,j} - B_{i,j})^2 .$$

This norm simply measures the difference between A and B on a subset of their entries defined by Ω. The second adjustment we will make is to not allow W or Z to get to large in a manner that depends on some parameter β. This yields the following optimization problem

$$\min_{W,Z} \left\|A - WZ^T\right\|_{F,\Omega}^2 + \beta^2 \|W\|_F^2 + \beta^2 \|Z\|_F^2 .$$

The hope is that WZ^T is then a good approximation of the underlying image.

We can now rephrase alternating least squares with regularization and incomplete information in Algorithm 2, once again we will simply run it for a small number of iterations rather than checking for convergence as one would in practice.

Algorithm 2 Alternating least squares, with regularization and unknown entries

1: **initialize** $W \in \mathbb{R}^{n_1 \times k}$, $Z \in \mathbb{R}^{n_2 \times k}$
2: **while** not converged **do**
3:
$$Z \leftarrow \text{argmin}_Z \left\|A - WZ^T\right\|_{F,\Omega}^2 + \beta^2 \|Z\|_F^2$$
4:
$$W \leftarrow \text{argmin}_W \left\|A - WZ^T\right\|_{F,\Omega}^2 + \beta^2 \|W\|_F^2$$
5: **end while**
6: **return** W, Z

Second set of goals

Turns out, you have all of the knowledge and machinery to implement this algorithm (save for one small item we develop in the first point below). Please consider and report on the following.

- Show that if we want to solve

$$\min_x \|Ax - b\|_2^2 + \beta^2 \|x\|_2^2$$

we can instead solve

$$\min_x \left\| \begin{bmatrix} A \\ \beta I \end{bmatrix} x - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|_2^2.$$

- Using your QR factorization implement Algorithm 2. From the first set of goals, you should have worked out how to split this up into solving many least squares problems. Now, you have to think about what size those problems are, and which entries of the matrices they involve. Given that, you can then leverage your QR factorization and the preceding item to implement the algorithm. **Please explain how you do this in the project report.**

- Using your implementation, load each of the image files in Project1_test*.mat where * is 1,4, and 6. Each file contains an image C and a variable called mask. mask is a matrix of the same size as C and its entires are 1 if the corresponding entry of C is observed and 0 if it is unobserved (so it defines the set Ω). The unknown entries of C have been set to 1 and you should not access or use them. Using Algorithm 2 try and recover the underlying images. You now have some parameters to consider and we encourage exploration of their values. We can say that you certainly should not have to go to k larger than 75 for any of the examples, and good β to try are between 10^{-2} and 1. Report the images you recover, and discuss what you observe.