FINAL EXAM: SVD and the Netflix Problem

ACM 106a: Introductory Methods of Computational Mathematics (Fall 2013)
Due date: 4pm, Thursday, December 12, 2013

1 Recommender Systems and SVD

Recommender systems or *recommendation systems* are broadly defined as algorithms used to predict the rating an individual would assign to an item. For example, an online vendor may want to determine the preferences of its users in order to recommend new products (and increase its sales). These preferences can be thought of as being stored in a matrix with rows corresponding to items in the vendor's catalogue and columns corresponding to each user; the (i, j)th entry of this matrix would be the rating given to item i by user j. For example, the Netflix data set consists of a matrix with rows corresponding to movies in Netflix's catalogue, columns indexed by users and entries corresponding to a user rating of a particular movie on an integer scale from one to five.

Unfortunately, most entries of this preference matrix are unknown in general; many, if not all, users will not have rated every item in the catalogue. The goal of a recommender system is to obtain a good estimate of the unknown entries of this preference matrix. This is a special case of the *matrix completion problem*: given partially known matrix M, compute the unknown entries of M. Clearly, this problem is absurdly underdetermined; for example, without any additional assumptions, any value between one and five can be assigned to each unknown entry of the Netflix matrix. One classical approach to the matrix completion problem is to place some additional assumption on the matrix M to be completed, e.g., that M is low-rank, and perform regularization to choose a particular solution. This is not an unreasonable assumption regarding the structure of preference matrices. For example, it is widely theorized and/or assumed that individual users in the Netflix matrix can be represented as a linear combination of a relatively small number of *eigencustomers*, whose behaviour represent canonical preferences. For example, one eigencustomer may be a user who only watches documentaries, another is someone who only enjoys comedies, etc. Similarly, any movie can be represented as a combination of *eigenmovies*. Although there may be many canonical users or catalogue items, it is highly likely that the number of such users or items should be much smaller than the dimensions of the preference matrix (on the order of dozens versus tens of thousand of movies and millions of users in the case of the Netflix matrix). This corresponds to the preference matrix having very small rank (relative to its dimension).

This assumption leads to a natural formulation of the problem as an underdetermined linear system with matrix rank used for regularization. Unfortunately, this requires the solution of an intractable optimization problem and the methods for approximately solving it are well beyond the scope of this course; however, if you are interested this may be covered in ACM 113: Introduction to Optimization. Instead, we will use a classical technique based on singular value decomposition of the preference matrix to obtain our predicted user ratings. This heuristic follows two steps. In the first, values are assigned to the unknown entries of the preference matrix. For example, if the (i, j) entry is unknown, then we may assign a value by taking a combination of the average rating for movie i or the average rating by user j. Once the unknown entries have been assigned a value, we obtain a rank-k prediction matrix using the singular value decomposition of the preference matrix (with filled in entries). The use of low-rank matrices to identify a low-dimensional set of latent variables explaining variance also plays a significant role in the analysis of (incomplete) scientific data, especially that arising from applications in geophysics, and is closely related to principal component analysis; this will likely be discussed at length in ACM/ESE 118: Methods in Applied Statistics and Data Analysis.

2 Software Engineering

Recall that the singular value decomposition $A = U\Sigma V^T$ of the matrix $A \in \mathbf{R}^{m\times n}$ can be obtained by the following procedure:

1. Form the matrix A^TA.
2. Compute the eigendecomposition $A^TA = VDV^T$.
3. Let $\Sigma = D^{1/2}$.
4. Solve the system $U\Sigma = AV$ for the matrix of left singular vectors U.

Modify this procedure to find the rank-k approximation of A given by

$$A_k = \sum_{i=1}^{k} \sigma_i u_i v_i^T$$

for arbitrary k, i.e., describe how the leading k singular values and vectors can be obtained from a partial eigendecomposition of A^TA. Note that if $\sigma_k \gg \sigma_n$ then the loss of accuracy normally associated with this procedure is avoided.

Write a Matlab program to implement your algorithm for obtaining the leading k singular values and vectors of A:

- Your program should be written from scratch and *should not* use any of Matlab's built-in procedures for linear system solving or eigendecomposition. (You may use the built-in procedures for matrix-vector multiplication, evaluating norms, taking transposes of matrices, etc.; when in doubt, ask if you can use a procedure). If you are unable to implement your algorithm without using built-in procedures for the standard algorithms discussed in class (e.g., QR, Hessenberg reduction, Gaussian elimination, etc.) you may resort to using Matlab's built-in procedures (although with a corresponding loss of marks).
- Any subroutine should be encoded optimally in the sense that it requires the minimum number of floating point operations possible (up to constants). Here, optimality means that you choose the algorithm discussed in class that requires the minimum number of floating point operations (up to constants) when performing any intermediate steps. For example, if you need to compute the eigendecomposition of an $n \times n$ matrix by a direct method then the number of operations should be $O(n^3)$ and not $O(n^4)$.
- Your program should not compute the full singular value decomposition of A (unless $k = n$).

Debugging Hints: Although your final submission should not use any of Matlab's native procedures, it may be a useful exercise to build a schematic of your algorithm which uses these procedures as subroutines. The benefit of this is two-fold: first, it should give a good idea of what subroutines you will have to code to get a working singular value solver and, second, it provides a comparison to check if your algorithm is performing correctly. Your algorithm should perform similarly to Matlab's "svds" command (which may also be used to check your answers). You should not submit these comparison tests, but they will likely be necessary to obtain a working implementation.

Stopping Criteria: It may be helpful (especially when debugging) to use stopping tolerances significantly larger than machine epsilon. I would recommend passing `eps` as a parameter to your iterative algorithms and stop when relative error is below `eps`. Solving to within $\epsilon = 10^{-16}$ is probably more precision than you need to obtain a good recommender system in Sect. 3 and may potentially introduce numerical issues which may cause your algorithm to fail to converge (even if it converges for larger ϵ, say $\epsilon = 10^{-6}$).

Procedure Clarification: We could also find the rank-k approximation of A using a combination of Arnoldi iteration with a Rayleigh-Ritz type inner-step to approximate the leading singular values and vectors. In class we omitted most of the details of the bidiagonalization process and the diagonalization process for a bidiagonal matrix, which would form the backbone of such an algorithm. We instead obtain the leading k singular values and vectors using the eigendecomposition of $A^T A$ as above.

3 Data Analysis

We next apply our singular value solver to an instance of the Netflix problem drawn from a subset of the MovieLens data set (see here and here for more information regarding MovieLens). Our data is contained in the file `movies.mat`. The data set is stored as an 800×500 real integer matrix containing the ratings (on a scale from 1 to 5) of a catalogue of 800 movies by 500 users. The data is highly incomplete (roughly 12% of all possible ratings are known).

We will use the SVD-based procedure described above to learn the unknown customer ratings (which in turn would be used to make recommendations to our customers). The data has been split into the training and testing sets

- `movies_train`: training data used to learn unknown customer rankings, and
- `movies_test`: testing data used to compare our learned preferences with actual customer preferences.

We proceed as follows:

- **Training step:** We will apply the following two-stage procedure to learn the unknown customer rankings from the training matrix $M =$ `movies_train`.
  1. **Fill-in:** Any unknown entries of M (stored as 0) are replaced with either
     (a) a random integer between 1 and 5,
     (b) average movie rating (average of known entries in the row),
     (c) average customer rating (average of known entries in column), or
     (d) average of the two ($0.5\times$(column average + row average)).
  2. **Singular Value Decomposition:** We use the SVD of M to obtain the desired unknown ratings. Specifically, we form the rank-k approximation M_k given by the singular value decomposition and use the entries of M_k as the corresponding unknown entries of M.
- **Testing step:** To evaluate performance, we compare the learned entries of M with those of the test set $T =$ `movies_test`. We use *root mean square error* to measure accuracy of our predictions:

$$RMSE = \left(\frac{1}{N_{test}} \sum_{ij:T_{ij}\neq 0} (T_{ij} - [M_k]_{ij})^2 \right)^{1/2},$$

  where $N_{test} = |\{ij : T_{ij} \neq 0\}|$. That is, we measure error as the square-root of the average squared error in our prediction over all known ratings in the test set.

Use your partial singular value decomposition algorithm from the previous section to perform the following tasks:

(a) Compute the predicted ratings for each value of k in $\{1, 2, 3, 4, 5, \dots, 25\}$ and each fill-in method described above.

(b) Plot the RMSE of your prediction error as a function of k for each fill-in method and compare with a plot of relative error between each rank-k matrix and the true value of M with unknown entries filled in.

(c) Plot the convergence of each singular value for each fill-in method. That is, plot the value of σ_i for each step of your iterative singular value decomposition algorithm. See the plots in Sections 7.3 and 7.4 in Demmel for an idea of what these plots should look like.

(d) Which of the fill-in methods performs best? Here you should comment on the time required to obtain each prediction matrix, the trade-off between number of singular values and vectors used and predictive performance, etc.

4 Submission Instructions

Your completed exam should consist of three main components:

1. Matlab code implementing your partial SVD algorithm, including any necessary subroutines and procedures used within your algorithm called as separate Matlab functions.
2. Matlab code used to perform the analysis of the MovieLens data set, including:
   - Script file calling your partial SVD algorithm.
   - Diary file of your Matlab session.
   - Saved workspace containing the partial SVDs obtained using your algorithm for each fill-in method.
3. A written report including:
   - A description of your partial SVD algorithm as pseudocode. This should include an outline of the steps performed by your algorithm and an explicit proof/argument establishing that your algorithm correctly identifies the first k left singular vectors of A given the first k nonzero singular values and right singular vectors.
   - Exposition describing the use of your algorithm to analyze the MovieLens data set. This should also include some description/discussion of any input parameters (stopping criteria, maximum number of iterations, etc.) used inside the subroutines.
   - Discussion of the results of your data analysis (including any plots and comparisons asked of you in the previous section).

The final exam is due **Thursday December 12, 2013** at **4pm**. No late submissions will be accepted. Please submit your written report directly to Professor Ames (in Annenberg 211) and submit your Matlab files to the course submission email account: homework.acm106a@gmail.com. Your submission should follow the file naming conventions of the homework assignments.