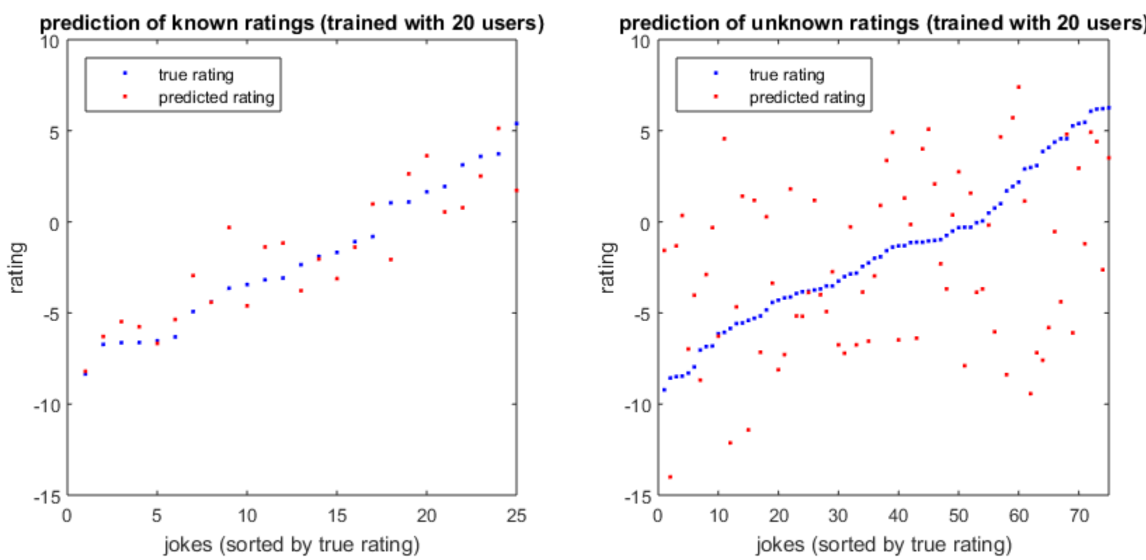
CS/ECE/ME 532
Homework 4: The SVD

In this homework set you will work with an analyze the dasaset `jesterdata.mat`, which is available on the moodle site. The dataset contains an $m = 100$ by $n = 7200$ dimensional matrix X. Each row of X corresponds to a joke, and each column corresponds to a user. Each of the users rated the quality of each joke on a scale of $[-10, 10]$.

1. Suppose that you work for a company that makes joke recommendations to customers. You are given a large dataset X of jokes and ratings. It contains n reviews for each of m jokes. The reviews were generated by n users who represent a diverse set of tastes. Each reviewer rated every movie on a scale of $[-10, 10]$. A new customer has rated $k = 25$ of the jokes, and the goal is to predict another joke that the customer will like based on her k ratings. Use the first $n = 20$ columns of X for this prediction problem (so that the problem is overdetermined). Her ratings are contained in the file `newuser.mat`, also on moodle, in a vector b. The jokes she didn't rate are indicated by a (false) score of -99. Compare your predictions to her complete set of ratings, contained in the vector `trueb`. Her actual favorite joke was number 29. Does it seem like your predictor is working well?

SOLUTION: If we take only the first 20 columns of X and call this new matrix $\bar{X}$, then our model looks like $\bar{X}w \approx b$. However, some entries of b are missing, so we only keep the rows of $\bar{X}$ and b for which we have data. Then, we use the least-squares solution to solve for $\hat{w}$ and our prediction is $\hat{b} = \bar{X}\hat{w}$. See the plot below for a comparison of how well the predictor predicts the new user's ratings for (i) the jokes the new user rated and (ii) the jokes the user hasn't rated.



As we can see, the prediction is pretty bad for the unrated jokes. To quantify the error, since the ratings are continuous, we computed the *average* error: $e_{\text{avg}} = \frac{1}{n}\|\hat{b} - b\|_2$ where n is the length of the b vector. We found: **average error on the 25 rated jokes is 0.3437 and average error on the 75 unrated jokes is 0.6191.**

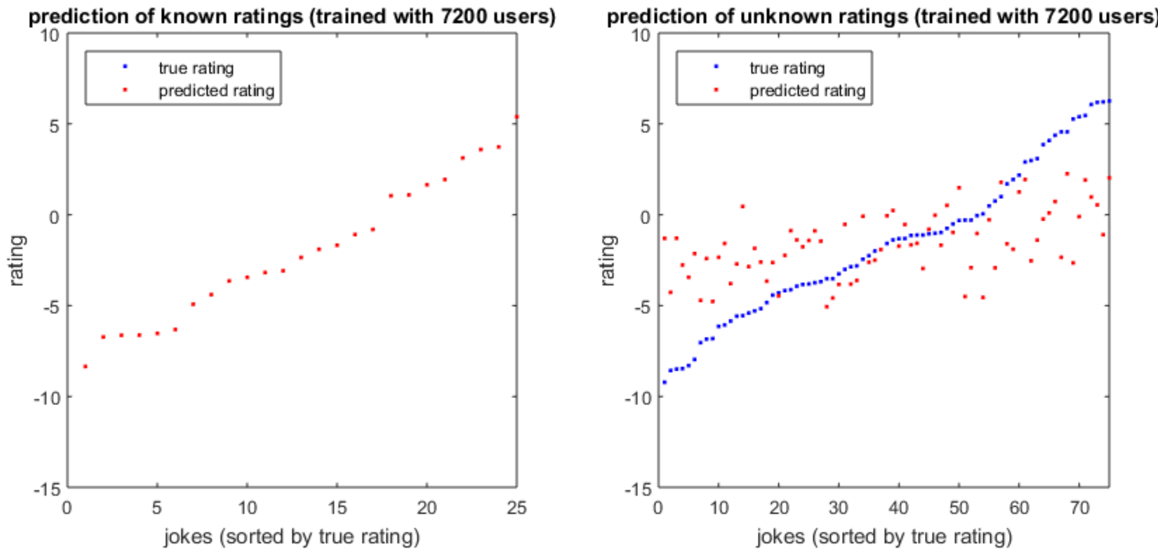
The predicted rating for the best joke (joke #29, rating 6.26) was 3.52, and many other jokes were rated higher than this. So this predictor doesn't do a good job of predicting the favorite joke either. Here is the code that produces the figures above:

```
%% Problem 1
load jesterdata  % loads data matrix X
load newuser     % loads known ratings b and true ratings trueb

indr = find(b ~= -99);  % movies that were rated
ntotal = numel(b);      % total number of movies (100)
ntrain = numel(indr);   % number of training movies (25)
indv = setdiff(1:ntotal,indr);  % movies used for validation
nvalid = numel(indv);   % number of validataion movies (75)

Xdata = X(indr,1:20);
bdata = b(indr);
bvalid = trueb(indv);

% solve for weights
w = Xdata\bdata;

% compute predictions
bhat =  X(:,1:20)*w;

% performance on rated jokes
bhat_train = bhat(indr);
avgerr_train = 1/ntrain*norm(bhat_train - bdata)
[srt,ind] = sort(bdata);
figure(1); clf; subplot(121)
plot( 1:ntrain, srt, 'b.', 1:ntrain, bhat_train(ind), 'r.' )
title('prediction of known ratings (trained with 20 users)')
xlabel('jokes (sorted by true rating)')
ylabel('rating')
legend('true rating','predicted rating','Location','northwest')
axis([0 ntrain -15 10])

% performance on unrated jokes
bhat_valid = bhat(indv);
avgerr_valid = 1/nvalid*norm(bhat_valid - bvalid)
[srt,ind] = sort(bvalid);
subplot(122); plot( 1:nvalid, srt, 'b.', 1:nvalid, bhat_valid(ind), 'r.' )
title('prediction of unknown ratings (trained with 20 users)')
xlabel('jokes (sorted by true rating)')
ylabel('rating')
legend('true rating','predicted rating','Location','northwest')
axis([0 nvalid -15 10])
```

2. Repeat the prediction problem above, but this time use the entire X matrix. Note that now the problem is underdetermined. Explain how you will solve this prediction problem and apply it to the data. Does it seem like your predictor is working? How does it compare to the first method based on only 20 users?

SOLUTION: The problem is now underdetermined because our X matrix used for training is 25×7200 instead of being 25×20. So the least-squares problem does not have a unique solution. One sensible thing to try is to pick the solution with minimum norm. This can be done in two ways: (1) compute the regularized least-squares solution $\hat{w} = (X^T X + \lambda I)^{-1} X^T b$ and choose a λ that is sufficiently small and (2) compute the minimum-norm solution directly using the method seen in class $\hat{w} = X^T (X X^T)^{-1} b$. Both produce the same answer, but the second approach is much faster because it only requires inverting a 25×25 matrix rather than a 7200×7200 matrix.

Note: if you use the first approach, be careful — typing `inv(A)*b` into Matlab may lead to numerical inaccuracies due to the large matrices involved. Instead, use `A\b`. The code is essentially the same, so we won't repeat it here. Here are the new plots:



This time: **average error on the 25 rated jokes is 1.8208e-15 (zero) and average error on the 75 unrated jokes is 0.4035.** So the error improved a bit, but qualitatively it still looks like our predictor is struggling.

This new predictor has no error on the first 25 jokes, which is to be expected since we are underdetermined. The error on the other 75 jokes has lower variance than before, so it does a better job of predicting which joke is the funniest... but still does a poor job of predicting the actual rating of that joke.

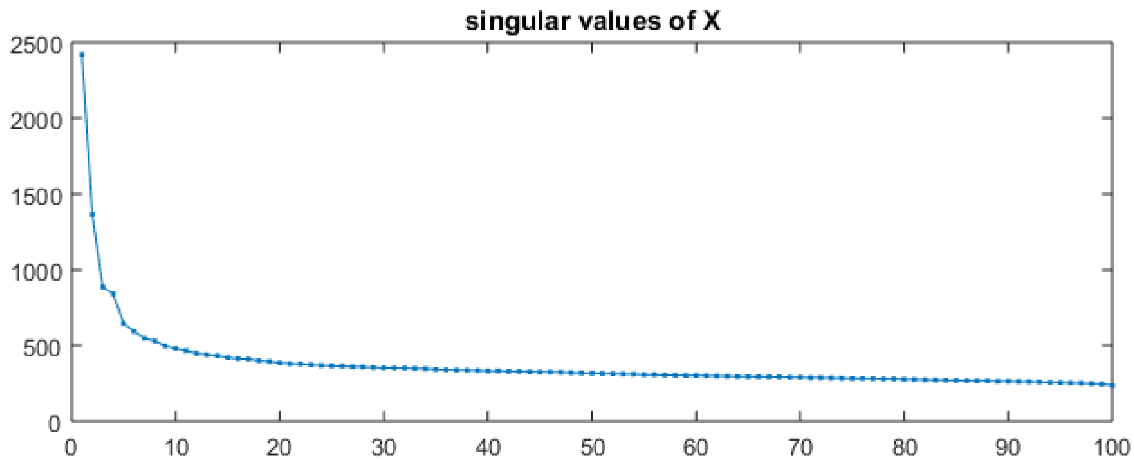
3. Propose a method for finding one other user that seems to give the best predictions for the new user. How well does this approach perform? Now try to find the best two users to predict the new user.

SOLUTION: There is no *right answer* to this question — there are many ways to go about it. Here are some possibilities:

- pick the coefficient(s) of $\hat{w}$ with the largest value(s) and choose the corresponding user(s).
- iterate through the users and compare the vector of their ratings to those of the new user
- apply a regularization to the least-squares problem that promotes sparsity (such as the 1-norm) and increase λ until only a small number of nonzero entries in $\hat{w}$ remain. This method actually works a little *too* well, which might give you a hint as to how the data set was generated...

4. Use the Matlab function svd with the 'economy size' option to compute the SVD of $X = U\Sigma V^T$. Plot the sprectrum of X. What is the rank of X? How many dimensions seem important? What does this tell us about the jokes and users?

SOLUTION: See the plot of the spectrum below.



The rank of X is 100, as expected, since 100 is the smaller of the two dimensions of X and this matrix represents noisy data. The first four singular values are substantially larger than the others, so these seem to be the most important. Of course, the more you take, the better your approximation.

What it tells us about users: if each user is characterized by a vector in $\mathbb{R}^{100}$ representing their taste profile for the 100 jokes, then the 7200 users are well-approximated by a subspace of dimension 4. So we can imagine 4 canonical users and their different preference profiles, and all 7200 users can be well approximated by appropriate linear combinations of the preferences of the 4 canonical users.

What it tells us about jokes: if each joke is characterized by a vector in $\mathbb{R}^{7200}$ representing how it appeals to each of the 7200 users, then the 100 jokes are well-approximated by a subspace of dimension 4. So we can imagine 4 canonical jokes and their different user appeals, and all 100 jokes can be well approximated by appropriate linear combinations of the appeal of the 4 canonical jokes.

5. Visualize the dataset by projecting the columns and rows on to the first three principle component directions. Use the `rotate` tool in the Matlab plot to get different views of the three dimensional projections. Discuss the structure of the projections and what it might tell us about the jokes and users.

SOLUTION: The first three columns of U (the left singular vectors) are vectors of length 100 that represent canonical users that can be used to approximate all other users. The projection onto that subspace yields three components (the weights associated with each of the canonical users). So we get a triplet in $\mathbb{R}^3$ for each of the 7200 users.

The first four columns of V (the right singular vectors) are vectors of length 7200 that represent canonical jokes that can be used to approximate all other jokes. The projection onto that subspace yields three components (the weights associated with each of the canonical jokes). So we get a triplet in $\mathbb{R}^3$ for each of the 100 jokes.

Plotting these points can reveal some structure which can perhaps be useful if we want to further classify the jokes(users) into categories depending on their weighting profile with respect to the canonical jokes(users). We leave out the plots from these solutions.

6. One easy way to compute the first principle component for large datasets like this is the so-called power method (see `http://en.wikipedia.org/wiki/Power_iteration`). Explain the power method and why it works. Write your own code to implement the power method in Matlab and use it to compute the first column of U and V in the SVD of X. Does it produce the same result as Matlab's built-in `svd` function?

SOLUTION: The power method finds the largest eigenvalue (and corresponding eigenvector) of a matrix iteratively by starting with a guess value, and repeatedly multiplying by the matrix and normalizing. Eventually, the vector will point in the direction of the eigenvector associated with the largest eigenvalue. A minor modification is required to use the power method for finding *singular values* and *singular vectors*.

The singular values of X are the eigenvalues of $X^T X$ (or XX^T). This is easily checked by substituting in $X = U\Sigma V^T$. We then find that:

$$X^T X v_1 = \sigma_1 v_1 \qquad \text{and} \qquad XX^T u_1 = \sigma_1 u_1$$

So for example, we can apply the power method to XX^T to find σ_1 and u_1. Then, we can recover v_1 using the fact that $v_1 = \frac{1}{\sigma_1} X^T u_1$. It's preferable to apply the power method to XX^T rather than $X^T X$ since it's a much smaller matrix! Why does the power method work? substituting the SVD factorization, we have: $XX^T = U\Sigma^2 U^T$. Consider the power method without normalization: $w_{k+1} = U\Sigma^2 U^T w_k$. Rearranging and defining $q_k = U^T w_k$, we have $q_{k+1} = \Sigma^2 q_k$. So if we start at q_0, then $q_k = \Sigma^{2k} q_0$. If the components of q_0 are $(z_1, \dots, z_r)$, the normalized version is:

$$\frac{q_k}{\|q_k\|} = \frac{1}{\sqrt{\sigma_1^{4k} z_1^2 + \cdots + \sigma_r^{4k} z_r^2}} \begin{bmatrix} \sigma_1^{2k} z_1 \\ \vdots \\ \sigma_r^{2k} z_r \end{bmatrix}$$

Assuming $\sigma_1 > \sigma_2$ and $z_1 \neq 0$, the first component of this limit is:

$$\frac{\sigma_1^{2k} z_1}{\sqrt{\sigma_1^{4k} z_1^2 + \cdots + \sigma_r^{4k} z_r^2}} = \frac{1}{\sqrt{1 + (\frac{\sigma_2}{\sigma_1})^{4k} (\frac{z_2}{z_1})^2 + \cdots + (\frac{\sigma_r}{\sigma_1})^{4k} (\frac{z_r}{z_1})^2}} \to 1$$

because each ratio $(\sigma_i/\sigma_1)^{4k} \to 0$ due to the fact that $\sigma_1 > \sigma_i > 0$. Similarly, we can check that every other component has a limit of zero. So only the first component survives. Eventually, we obtain:

$$\lim_{k\to\infty} \frac{q_k}{\|q_k\|} = \begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

So our original iterate can be computed: $w_k = U q_k \to u_1$. So the power method applied to XX^T converges to u_1, as required. Then, it is straightforward to obtain σ_1 and v_1 using the identity $X^T u_1 = \sigma_1 v_1$. Here is Matlab code that applies the power method.

```
% compute the values using Matlab's svd function:
[U,S,V] = svd(X,'econ');

% we compute u first because it's smaller!
A = X*X';
u = randn(size(A,2),1); % random starting point:

tol = 1e-12;
max_iter = 100;

for i = 1:max_iter
    uu = A*u;
    uu = uu/norm(uu);
    if norm(u-uu) < tol
        disp('tolerance reached')
        break
    else
        u = uu;
        continue
    end
end
if i == max_iter
    disp('max iterations reached')
end
sv = X'*u;          % identity: X'*u1 = sigma1*v1
s = norm(sv);       % singular value
v = sv/s;           % right singular vector

% compute errors
abs(s - S(1,1))
abs(1 - abs(u'*U(:,1)))
abs(1 - abs(v'*V(:,1)))
```

Note that the code above will not always find the same singular vector as Matlab's SVD command. This is because if (u_1, σ_1, v_1) is a valid triplet, so is $(-u_1, \sigma_1, -v_1)$. The unit vectors will always point in the right direction, but might have a different sign. To test this, the code checks alignment of the unit vectors by taking a dot product.

Note we assumed above that $\sigma_1 > \sigma_2$. If instead we had $\sigma_1 = \sigma_2$, then the power method might not converge to the same singular vectors as the SVD function... but then again, the SVD will not be unique in such a case!

7. The power method is based on an initial starting vector. Give one example of a starting vector for which the power method will fail to find the first left and right singular vectors in this problem.

SOLUTION: In the derivation of Problem 6, we assumed that $z_1 \neq 0$. This turns out to be important! If the initial vector is such that $z_1 = 0$ (contains no component in the direction of u_1), then the power method will not converge to u_1. For example, if we initialize the power method with $w_0 = u_i$, then the power method will converge to (u_i, σ_i, v_i).

This is why we use a random initialization vector — because it guarantees that every possible component will be represented.