Analyzing Movie Ratings via SVD

In this notebook we're going to be working with a subset the MovieLens25M dataset. The original dataset (https://grouplens.org/datasets/movielens/25m/) contains

- 25 million ratings
- from 162,000 users
- on 62,000 movies

The dataset was generated on November 21st, 2019, so it is pretty current. In this activity, we're going to be using a reduced subset of this data that only includes popular movies (that have been rated at least 1000 times) and users that have rated lots of movies (at least 500). This leaves us with

- 7.1 million ratings
- from 9,663 users
- on 3,790 movies

The goals of this activity are threefold.

1. To work with a different type of data than images or temperatures (here we will be working with ratings). Applying the tools you have learned in this module to different domains will help solidify your learning, help you see connections, and potentially get you excited for your module 1 project.
2. To see how SVD can be used to examine the important trends in your data (since we had lots of practice with using the EVD on the overnight).
3. To have some fun!

To get started, we're going to load the data and display a little bit of the data. Please see the comments in the code for some more information.

```
load('movielens25m.mat');
sizeOfMovies = size(movies)
% the cell array `movies` is 3706 by 3.  Each of the 3706 entries correspopnds to a
% particular movie, and along the second dimension the entries correspond to the movie
% the movie title, and the movie genre
%
% Here we extract the information about the first movie in the dataset
[movieId, movieTitle, movieGenre] = movies{1,:}

ratingsSize = size(ratings)
% the matrix `ratings` is 6040 by 3706 and encodes the rating that a
% particular user (row) gave to a particular movie (column).  The ratings
% are 1, 2, 3, 4, or 5 stars or the special value NaN (not a number) if the
% user didn't rate that particular movie.
%
% Let's look at the ratings that were given to the first movie in the
% dataset, which as we saw is Toy Story.  We can do this using the histc
% function (we'll ignore missing values in this analysis)
possibleRatings = [0.5:0.5:5];
nRatings = histc(ratings(1,:), possibleRatings);
figure;
```

```
bar(possibleRatings, nRatings);
xlabel('Rating');
ylabel('Number of Users');
title(['Ratings for ', movieTitle])
```

Okay, yeah that was a pretty great movie. Let's check out a less good movie, Anaconda. Highly recommended!! Look at this cast https://www.imdb.com/title/tt0118615/fullcredits !!!

```
anacondaIndex = 850;
[movieId, movieTitle, movieGenre] = movies{anacondaIndex,:}
nRatings = histc(ratings(anacondaIndex,:), possibleRatings);
figure;
bar(possibleRatings, nRatings);
xlabel('Rating');
ylabel('Number of Users');
title(['Ratings for ', movieTitle])
```

Cleaning up the Data

As you probably guessed, we're going to be applying SVD to this data. Before we start analyzing this data, we're going to do a few things to make the problem a bit easier to handle. First we're going to have to deal with the fact that we have a bunch of missing values in our ratings matrix (i.e., movies that particular users did not rate). The step of filling in missing values is called **data imputation**. There are many ways to do this, but we've chosen a particularly easy strategy of simply replacing any ratings with the average rating of that particular movie (e.g., if a user didn't rate Toy Story, we would fill it in with the average rating of Toy Story based on the other users in the dataset who actually rated that movie).

```
ratingsFilled = fillmissing(ratings, 'constant', nanmean(ratings));
```

As a final data cleaning step, we're going to subtract out the mean of each row. This will control for the fact that users vary considerably in how the numerical score they assign to movies (e.g., one user's 3 may be more comparable to amother user's 1).

```
ratingsMeanCentered = ratingsFilled - mean(ratingsFilled,2);
```

Framing the Problem Using SVD

Next, let's think about how SVD might help us to analyze this dataset. Suppose we compute the SVD of the matrix `ratingsMeanCentered`. Let's use $\mathbf{u_1}$ to refer to the first left singular vector, $\mathbf{v_1}$ to refer to the first right singular vector, and σ_1 to refer to the first singular value (let's assume that the first pair of singular vectors has the largest singular vector).

Exercise

Before running any other code in this notebook, answer the following questions regarding the first pair of singular vectors.

1. What are the sizes of $\mathbf{u_1}$ and $\mathbf{v_1}$? What do each of the dimensions of $\mathbf{u_1}$ correspond to? How about each dimension of $\mathbf{v_1}$?

2. In 15.3.8 we talked about compressing the original matrix down to m + n + 1 values. If we think of $\mathbf{u_1}, \mathbf{v_1}, \sigma_1$ as the compressed version of ratings data, how would we reconstruct the ratings data using $\mathbf{u_1}, \mathbf{v_1}, \sigma_1$ (you essentially did this already, we're hoping you can recall this fact from earlier and apply it here).
3. We can think of $\mathbf{v_1}$ as encoding the dominant trend that explains the ratings of each movie. For this dataset, what might this correspond to?
4. We can think of $\mathbf{u_1}$ as encoding the dominant trend that explains the ratings by each usere. For this dataset, what might this correspond to? Keep in mind we have already subtracted out the mean of each user. It might be helpful to expand your formula from problem 2 to see how $\mathbf{u_1}, \mathbf{v_1}, \sigma_1$ interact with each other.

Now we're going to compute the SVD. We'll just compute the 10 pairs of left and right singular vectors with the largest singular values.

```
[U, Sigma, V] = svds(ratingsMeanCentered, 10);
```

Examining the Right Singular Vectors

Now that we've computed our singular vectors, let's see if we can make sense of them. It turns out that the right singular vectors (the ones that have to do with movies) are generally more interpretable than the left singular vectors (the ones that have to do with users). We'll start out by looking at each right singular vector.

Exercise

Before running the code, think through the following question with your table-mates.

What might you do in order to make sense of what a particular right singular vector represents? Consider things like examining small or large values, looking for correlations, etc. There's not only one right answer, so throw out some ideas and try to think through what examining a particular aspect of the vector might tell you.

(we'll leave a little space to make it easier not to look at what we did)

Looking at Large and Small Values

One simple way to understand the right singular vectors is to look at the largest and smallest components of each vector. This will tell us which movies are either most strongly (positively) and most strongly (negatively) associated with this component. In the code below, we'll print out the title, genre, and component of the 10 movies that are most positively and most negatively associated with each right singular vector. **Exercise: Based on these outputs, can you tell a story about what the singular vector represents?**

```
for i = 1 : 10
    disp(['Component ', num2str(i)]);
```

```
    getHighAndLowMovies(V(:,i), movies)
end
```

Examining the Left Singular Vectors

Now we're going to check out the left singular vectors.

Exercise

Before running the code, think through the following question with your table-mates.

What might you do in order to make sense of what a particular left singular vector represents? Consider things like examining small or large values, looking for correlations, etc. There's not only one right answer, so throw out some ideas and try to think through what examining a particular aspect of the vector might tell you.

(we'll leave a little space to make it easier not to look at what we did)

Looking at Large and Small Values

Similarly to what we did for the right singular vectors, let's take a look at large (positive) and small (negative) components of each singular vector. Instead of looking at the top 10 and bottom 10, we're instead going to look at the single highest and single lowest component (each of which correspond to a user). For that user, we're going to show a sampling of movies that the user rated (focusing on the top 10 and bottom 10 ratings for that particular user). **Exercise: Given what you know about the corresponding right singular vector, try to make sense of the users that are at either extreme of the left singular vectors.**

```
for i = 1 : 10
    [~, highestUserIndex] = max(U(:,i));
    [~, lowestUserIndex] = min(U(:,i));
    disp('');
    disp(['Component ', num2str(i)]);
    disp('The user with the largest component rated the following movies as high and lo
    getHighAndLowUserRatings(highestUserIndex, movies, ratings)
    disp('The user with the smallest (probably negative) component rated the following
    getHighAndLowUserRatings(lowestUserIndex, movies, ratings)
end
```

Next Steps

To give you a sense of where you might take this in a project, here are some things you might investigate next with this dataset.

1. We didn't really look at how you would use the SVD to make recommendations. It turns out the SVD can be used to come up with good guesses for the missing values in the original ratings matrix (the NaNs) and you can then provide recommendations based tailored for a praticular user.
2. We didn't quantify how well the svd predicted the ratings. In order to do that, you could divide the ratings into a training and test set and see how well your SVD model can predict the test ratings (i.e., a rating set that wasn't used to compute the SVD).
3. We filled in the missing values with the means of each movie, but there are variants of SVD that can handle the missing values directly (they do entail tradeoffs). You could inverstigate how one of those methods would work on this data.

```
function movieExtremes = getHighAndLowMovies(v, movies)
    % return a cell array with the most positive and most negative
    % components of the right singular vector v.
    nHighLow = 10;
    movieExtremes = cell(nHighLow*2, 3);
    [c, indices] = sort(v);
    movieExtremes(1:nHighLow,1) = movies(indices(end-(nHighLow-1):end),2);
    movieExtremes(1:nHighLow,2) = movies(indices(end-(nHighLow-1):end),3);
    movieExtremes(1:nHighLow,3) = num2cell(c(end-(nHighLow-1):end));
    movieExtremes(1+nHighLow:end,1) = movies(indices(1:nHighLow),2);
    movieExtremes(1+nHighLow:end,2) = movies(indices(1:nHighLow),3);
    movieExtremes(1+nHighLow:end,3) = num2cell(c(1:nHighLow));
end

function userRatings = getHighAndLowUserRatings(userIndex, movies, ratings)
    % return a cell array with the most positive and most negative reviews
    % given by the specified user
    nHighLow = 10;
    userRatings = cell(nHighLow*2,2);
    [r, indices] = sort(ratings(userIndex,:));
    % filter out NaNs
    indices = indices(~isnan(r));
    r = r(~isnan(r));
    userRatings(1:nHighLow,1) = movies(indices(end-(nHighLow-1):end),2);
    userRatings(1:nHighLow,2) = num2cell(r(end-(nHighLow-1):end));
    userRatings(1+nHighLow:end,1) = movies(indices(1:nHighLow),2);
    userRatings(1+nHighLow:end,2) = num2cell(r(1:nHighLow));
end
```