Chapter 15

Day 8: Eigenface Synthesis and Project Kick-Off

Schedule

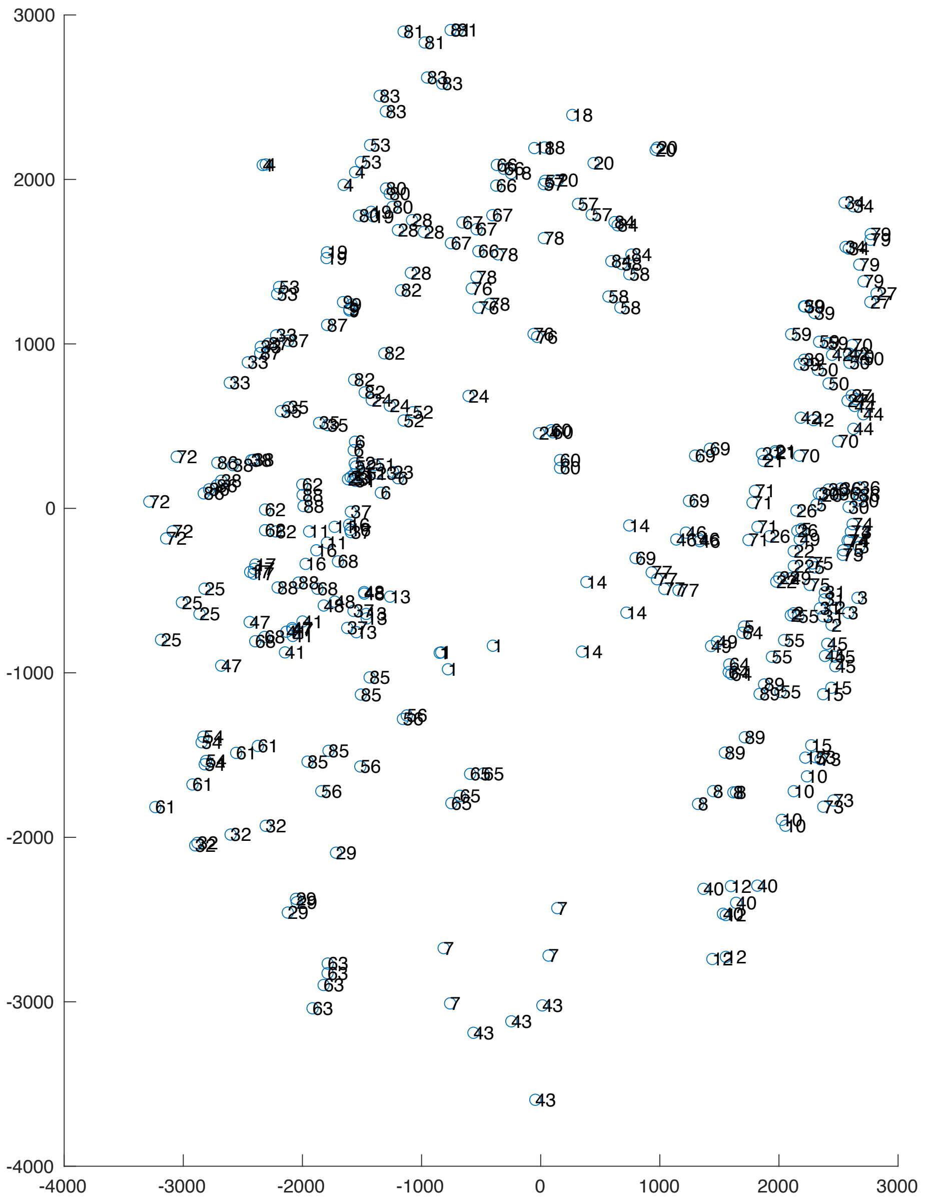
- 0900–1015 Debrief and Synthesis Activity on Eigenfaces
- 1015–1030 Coffee Break
- 1030–1115 Singular Value Decomposition — Theoretical
- 1115–1200 Singular Value Decomposition — In action
- 1200–1220 Review and Preview
- 1220–1230 Survey

15.1 Debrief and Synthesis Activity on Eigenfaces

During the overnight assignment you implemented a facial recognition routine. For the first part of the morning, we want you to work with your table mates to think again about your method and implementation.

An Aside on Nearest Neighbor Classification

One area where folks ran into trouble on the night assignment was the nearest neighbor approach to classification. In nearest neighbor classification you classify a test point, $\mathbf{x_t}$ (think of representing this as a column vector containing a new piece of data like a face image of someone whose identity we don't know), by comparing it to a set of labeled training points $(\mathbf{x_1}, y_1), (\mathbf{x_2}, y_2), \ldots, (\mathbf{x_n}, y_n)$. Each y_i is the thing we are trying to predict (e.g., the identify of the person in a face image). In nearest neighbor classification we check to see which of the n training points is closest to the test point. When we find the one that is closest, we predict the label for $\mathbf{x_t}$ to be the same as the label of the closest point in the training set. To help get the idea across, here is a figure that shows all of your faces projected onto the first two principal components. Also shown are the subject ids for each point. You should be able to see from the figure that points with the same subject id tend to cluster together (i.e., are close together in facespace). This is where the power of nearest neighbor classification comes from.



Eigenfaces Review [45 mins]

We'd like you to take this opportunity to discuss with your table-mates the approach you took to facial recognition and the way you implemented it in MATLAB. The goal here is to think through the method at different levels from conceptual to code, resolving any confusion, and identifying whether any issues are primarily at the conceptual-level, the implementation-level, or the translation space in between. **We highly recommend that you set aside your existing code and focus on developing the approach with your table-mates.** Here are some questions to guide you:

- Will you pre-process the data? If so, how?
- How will you compute the Eigenfaces?
- How will you decide how many Eigenfaces to include? Which Eigenfaces might you leave out?
- How will you identify a face?
- How will you measure the accuracy of your implementation?
- How will you handle false positives?

Exercise 15.1

1. Sketch out on the board your conceptual approach to facial recognition—use words and diagrams here, and think of a set of key frames.
2. Sketch out on the board your mathematical approach to facial recognition—translate the key frames from your conceptual approach to mathematical expressions or equations.
3. Sketch out on the board your implementation approach to facial recognition—translate the key conceptual and mathematical ideas to MATLAB pseudo-code.

Eigenfaces Paper [45 mins]

This is on the next overnight assignment, but it would make sense to start this now if you have the time.

Check out *Eigenfaces for Recognition*, an early paper on eigenfaces, by M. Turk and A. Pentland. You have most of the tools to understand this paper, but the writing style might be unfamiliar (intense!). Spend 1 hour on this paper and then feel free to move on. The first 6 pages of this paper describe the use of eigenfaces in face recognition.

Check out other sources as well. Wikipedia is pretty useful for eigenfaces, and this later paper talks about eigenfaces and an extension called Fisherfaces (not fish faces).

Exercise 15.2

We are asking you read this paper for several reasons. We hope that it highlights and synthesizes all the material you've learned in this module. It will also give you practice reading a technical paper, which is a skill you'll continue to develop over your career.

1. In what ways was your approach to implementing the eigenfaces algorithm similar or different from the authors' approach?
2. In what ways did your understanding of the eigenfaces algorithm change after reading the paper?
3. Were there places in the reading that you "got stuck?" If so, how did you address that?
4. What questions do you have after reading the paper?

15.2 Singular Value Decomposition (SVD) — Theoretical

We previously met the Eigenvalue Decomposition (EVD), which we used on square matrices. There is no EVD for rectangular matrices, but there does exist a generalization known as the Singular Value Decomposition, which is one of the most useful matrix decompositions in applied linear algebra. We will begin with some background concepts on singular values and singular vectors, and then explore them in the context of a user-movie rating data matrix. See the following webpage at the American Mathematical Society for a good geometric discussion of the SVD.

The Big Idea

Rectangular matrices don't have eigenvalues and eigenvectors. However, they have a generalisation of these known as singular values and singular vectors.

- The singular values σ_i and singular vectors $\mathbf{u}_i, \mathbf{v}_i$ of an $n \times m$ rectangular matrix $\mathbf{A}$ satisfy the definition

$$\begin{aligned} \mathbf{A}\mathbf{v}_i &= \sigma_i \mathbf{u}_i && (15.1) \\ \mathbf{A}^T\mathbf{u}_i &= \sigma_i \mathbf{v}_i && (15.2) \end{aligned}$$

  The singular vectors $\mathbf{v}_i$ are known as the **right singular vectors** and the singular vectors $\mathbf{u}_i$ are known as the **left singular vectors**.

- There are precisely $r = min(n, m)$ non-zero singular values. The singular vectors $\mathbf{v}_i$ are the eigenvectors of $\mathbf{A}^T\mathbf{A}$, and the singular vectors $\mathbf{u}_i$ are the eigenvectors of $\mathbf{A}\mathbf{A}^T$. The r non-zero eigenvalues of $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}\mathbf{A}^T$ are σ_i^2.

- The $n \times m$ matrix $\mathbf{A}$ has a *singular value decomposition* (SVD) of the form

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \tag{15.3}$$

  where $\mathbf{U}$ is an $n \times r$ orthogonal matrix whose columns are $\mathbf{u}_i$, $\mathbf{\Sigma}$ is an $r \times r$ diagonal matrix with r non-zero entries σ_i, and $\mathbf{V}$ is an $m \times r$ orthogonal matrix whose columns are $\mathbf{v}_i$. Please note that this version of the SVD is called the *reduced* or *economy* SVD - there is a more general form but this is the most useful in a practical setting.

Exercise 15.3

1. Read "The Big Idea" again!
2. Let's assume that $\mathbf{A}$ is a 3×2 matrix. What is the size of $\mathbf{A}^T$? What is the size of $\mathbf{v}_i$ and $\mathbf{u}_i$? What is the size of $\mathbf{A}^T\mathbf{A}$ and $\mathbf{A}\mathbf{A}^T$? How many eigenvalues will $\mathbf{A}^T\mathbf{A}$ have? How many eigenvalues will $\mathbf{A}\mathbf{A}^T$ have? What must be true about these eigenvalues according to "The Big Idea"?
3. Show that σ_i^2 and $\mathbf{v}_i$ are the eigenvalues and eigenvectors of $\mathbf{A}^T\mathbf{A}$ by multiplying Equation (15.1) by $\mathbf{A}^T$ and then using Equation (15.2) to simplify.
4. Show that σ_i^2 and $\mathbf{u}_i$ are the eigenvalues and eigenvectors of $\mathbf{A}\mathbf{A}^T$ by multiplying Equation (15.2) by $\mathbf{A}$ and then using Equation (15.1) to simplify.
5. Take the transpose of Equation (15.2) and justify the use of the term **left singular vector** for $\mathbf{u}_i$.
6. Why is it valid to write
$$\mathbf{A}[\mathbf{v}_1 \ldots \mathbf{v}_r] = [\mathbf{u}_1 \ldots \mathbf{u}_r] \begin{bmatrix} \sigma_1 & & \\ & \cdot & \\ & & \sigma_3 \end{bmatrix}$$
and why does this imply Equation (15.3)?
7. Why does Equation (15.3) imply that
$$\mathbf{A} = \sigma_1 \mathbf{u}_1 \mathbf{v}_1^T + \sigma_2 \mathbf{u}_2 \mathbf{v}_2^T + \ldots + \sigma_r \mathbf{u}_r \mathbf{v}_r^T \tag{15.4}$$
8. An $n \times m$ matrix has nm data values. How many data values do you need to store σ_1, $\mathbf{u}_1$, and $\mathbf{v}_1$? What kind of compression ratio would you have if you only stored the first singular value and the first singular vectors?

15.3 *Singular Value Decomposition (SVD) — In Action*

We're going to be using a LiveScript notebook to go through an example of using SVD to analyze movie ratings.

You'll need to download both the dataset (movielens25m.mat) and the LiveScript (movieLens25m.mlx) before getting started. We have included a PDF of the LiveScript in this document for your referene.

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.

```matlab
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
```