Machine Learning & Data Mining — Caltech CS/CNS/EE 155

Miniproject 3 — March 10, 2017

MovieLensVis

by Shari Kuroyama and Carly Robison

1 Data Formatting

We made three dictionaries using the `movies.txt` file for ease in generating graphs and in graphing movies by genre.

- movie_names: movie ID to movie name. Used for plotting the movie name.
- movie_genres: movie ID to the list of genres that it is categorized as.
- genres: genre name to the set of movie IDs that fit in that genre. Used for generating graphs by genre.

These structures worked well for our purposes.

When extracting ratings from `data.txt`, we put data into a dictionary of movie id to list of ratings, useful for the histograms, as well as the Y_{ij} format necessary for matrix factorization.

2 Basic Visualizations

Packages Used

To plot the histograms, we used `matplotlib`'s function `pyplot.hist`. This function has options for side-by-side data as well as stacked.

Methods

For visualizing all ratings in the data set and all ratings from certain genres, we counted the total number of 1s, 2s, etc., and plotted their frequencies. A histogram like this provides an accurate visualization of the distribution of ratings.

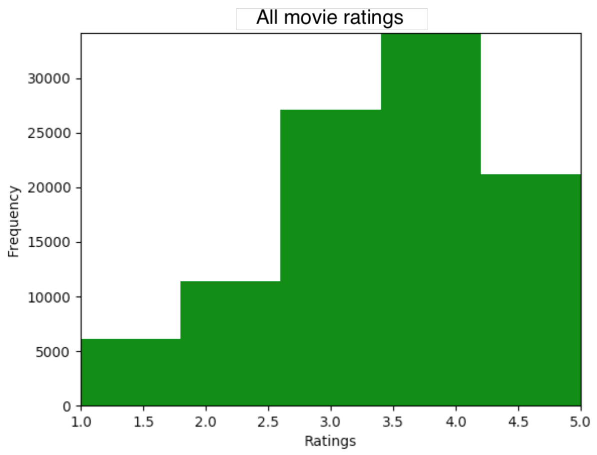
For visualizing the ten most popular or most highly rated, we calculated the top ten movies and plotted the number of 1s, 2s, etc. that each movie got side by side. For highest average ratings, a naïve try will pick up movies with only one rating of 5. To ensure that the movies we found were actually good movies, we required our choices to have more than 20 ratings.

We decided not to normalize the frequency data, since this way the movies which had more ratings are clearly distinguishable from those with smaller numbers. This is especially relevant for the visualization of the best movies, since it shows how many more ratings *Star Wars* got than, for example, *Wallace & Gromit*.

Observations

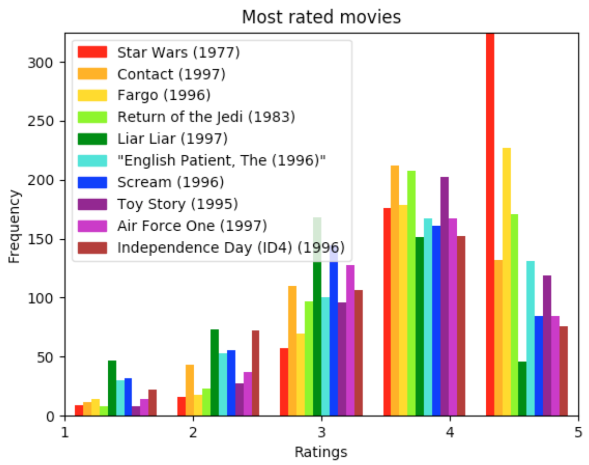
1. Most of the ratings in the dataset were 3s and 4s, with very few 1 ratings.

Figure 1: All ratings in the MovieLens Dataset.



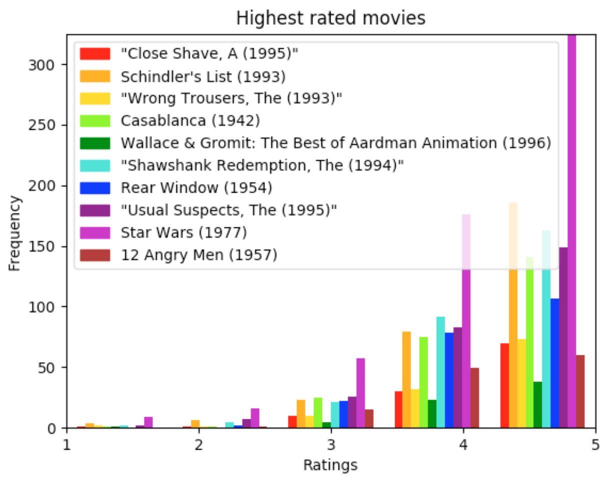
2. Most of these movies were not only rated a lot, but rated fairly highly as well. A notable exception is *Liar Liar*, which has very few 5 ratings (mostly 4s and 3s). This makes sense, because the movies which are watched the most are likely to have been promoted by people who watched them and liked them.

Figure 2: All ratings of the ten most popular movies.



3. As noted above, we restricted the movies we chose to have more than 20 ratings in order to be considered one of the "best". We agree with most of the outputs; in particular, the fact that three of these movies are from the Wallace & Gromit series was amusing.

Figure 3: All ratings of the ten best movies.



4. All ratings of movies from three genres of your choice. We chose Comedy, War, and Western. We looked at the same genres in the Matrix Factorization analysis.

   The Comedy movies were the most popular, with over 8000 ratings of 3 and 4 each. However, the War movies had higher average ratings, with many more 5s and relatively few 1s. The Westerns were kind of in-between, but with the fewest ratings (by far) of the three genres we chose.

Figure 4: All ratings of Comedy movies.

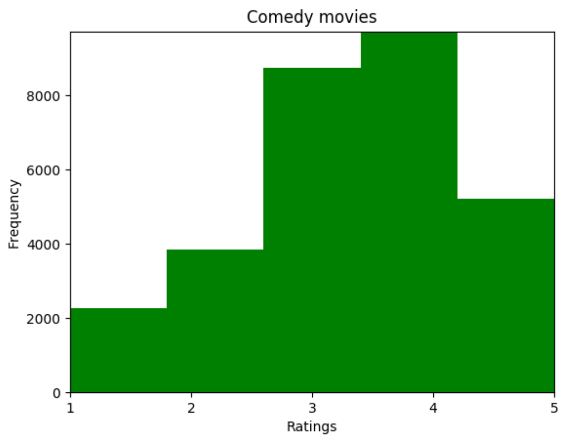


Figure 5: All ratings of War movies.

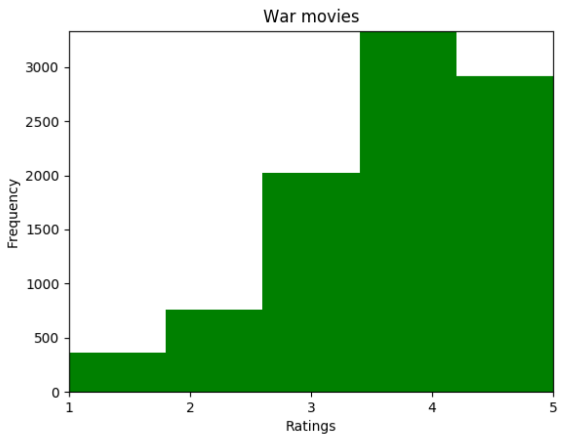
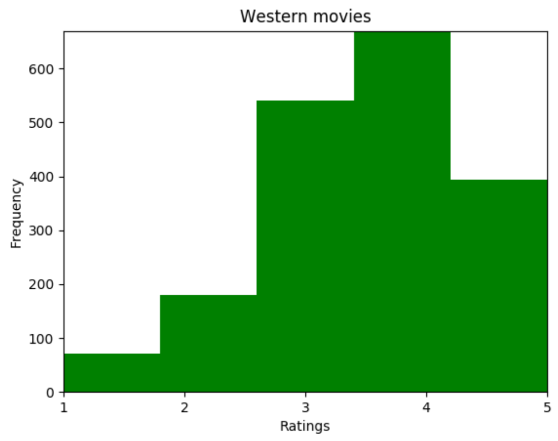


Figure 6: All ratings of Western movies.



3 Matrix Factorization Visualization

Packages Used

We used the code from the updated solutions to Homework 6 for matrix factorization and training on Y. We also used `numpy.linalg.svd` to perform SVD on the component matrices U and V.

Algorithm

We set M to the number of users + 1, because users are 1-indexed and the matrix is 0-indexed. Similarly we set N to the number of movies + 1.

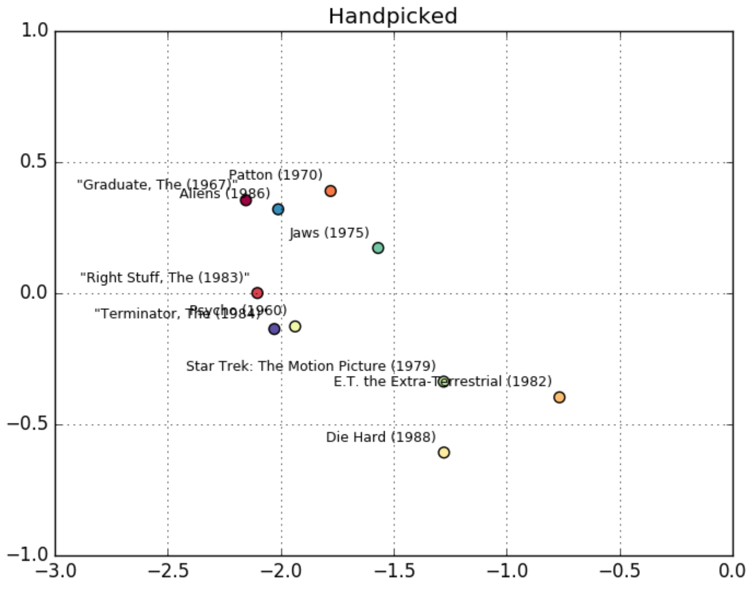
In getting the matrix factorization, we adjusted the regularization parameter and the learning rate. We saw last week that a regularization parameter of about .1 worked well for all numbers of data points, so we used that and it worked well. We also started with a learning rate of 0.03 as in last week's set but the learning didn't take that long so we decreased it to 0.01 to make sure we weren't overshooting an optimum. Our final mean error was about .26. We use the same stopping condition as in the HW6 solutions: when the change in error is small enough or when the error starts to rise.

Observations

The axes didn't make much sense to us, but in matrix factorization and SVD they aren't supposed to make sense, so this is unsurprising. In general we saw similar movies graphed near each other, but we are not that knowledgeable about movies, so we can't say this for certain. We were surprised at the varied locations of movies within each series (e.g. Star Wars, Star Trek, Wallace & Gromit) – we would have expected these to be closer together.

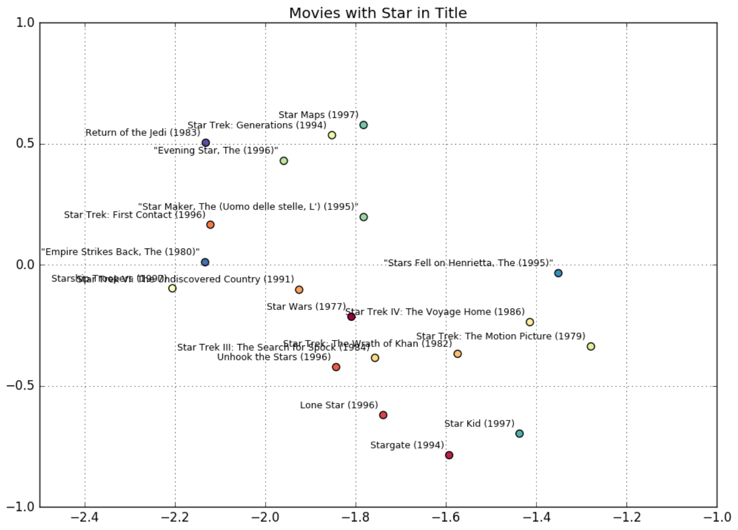
1. We asked a friend not in this class for ten movies of their choice, and plotted them. In this graph we can see two space movies, Star Trek and E.T., are closer to each other than they are, say, to The Graduate.

Figure 7: Handpicked movie visualization.



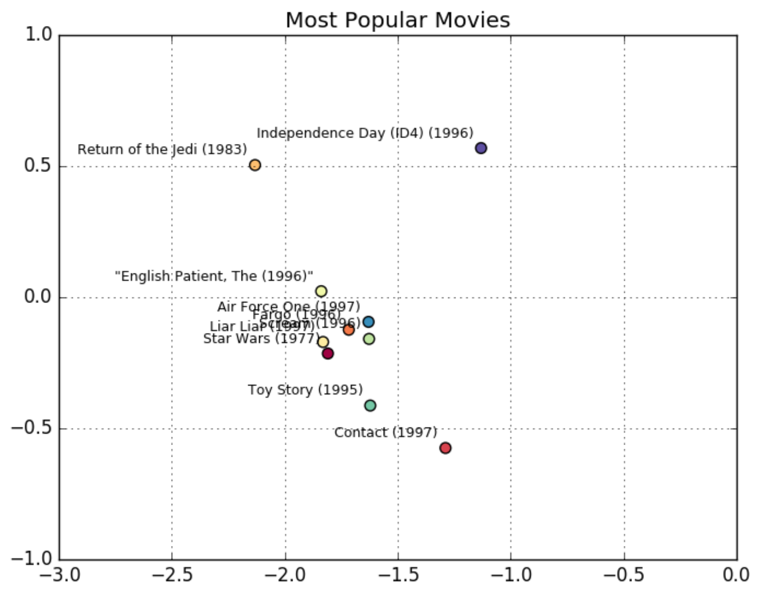
2. We also plotted all movies with "star" in their title, as well as some extra Star Wars films. We were surprised to see that the Star Wars and Star Trek movies were not highly clustered – instead, they were spread over the entire output space.

Figure 8: "Star" movie visualization.



3. The most popular movies are highly clustered at the mean of the output space. This makes sense, because if people from all walks of life like these movies then they will be popular, and the matrix factorization won't get any utility out of pigeonholing these movies. The exceptions are Return of the Jedi, Independence Day, and Contact.

Figure 9: Visualization of the ten most popular movies.



4. We used the same list of most popular movies here as in the histogram; specifically, those with more than 20 ratings overall. The highest rated movies are significantly more spread out than the most popular movies, and they are spread out along the vertical axis, which indicates that the horizontal axis could be some measure of the average rating of a movie. Here we see the Wallace & Gromit movies have a large spread.

5. Movies from three genres of your choice. We chose Comedy, War, and Western. We looked at these same genres in the histogram analysis. In the visualization, we chose 20 random movies from each genre. We chose these genres because they were the most highly clustered. Comedy movies seem to occupy the upper left section of the graph. Surprisingly, the War films seem to cluster in the same quadrant as the Comedy films! We also notice that most of the older films (pre-1960) are high on the vertical axis. The Western genre, for the most part, clusters in the bottom center part of the graph. Because the Western is a very defined genre, viewers might like Western movies for their Westernness (plot, design, etc.) rather than for their filmmakers or actors. Thus this high clustering is expected.

Figure 10: Visualization of the ten best movies.

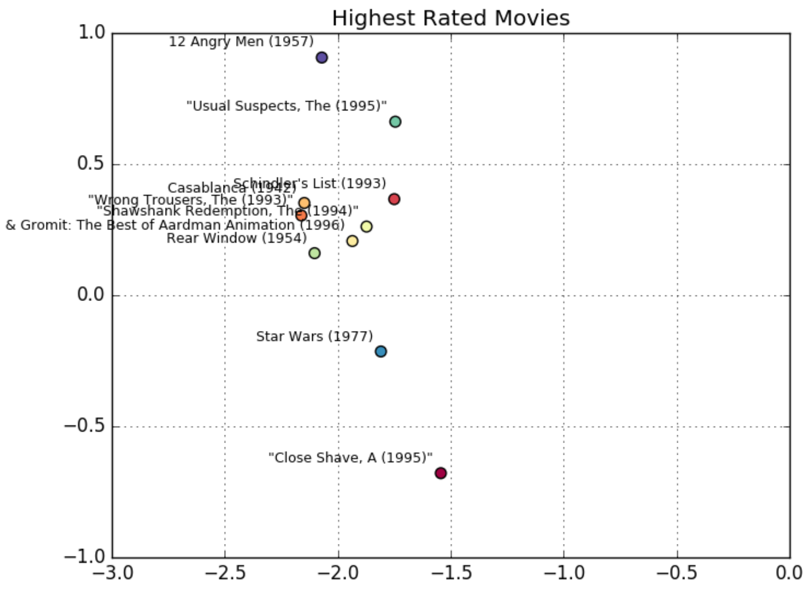


Figure 11: Visualization of Comedy movies.

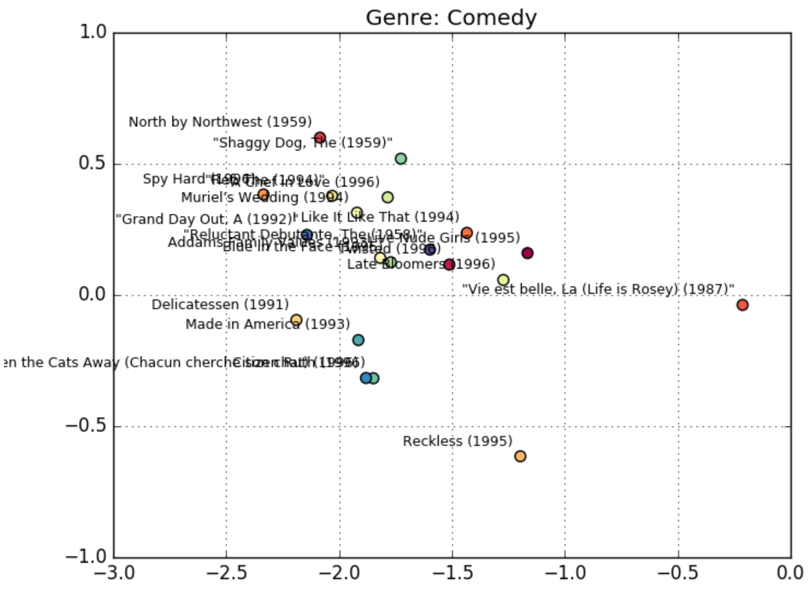


Figure 12: Visualization of War movies.

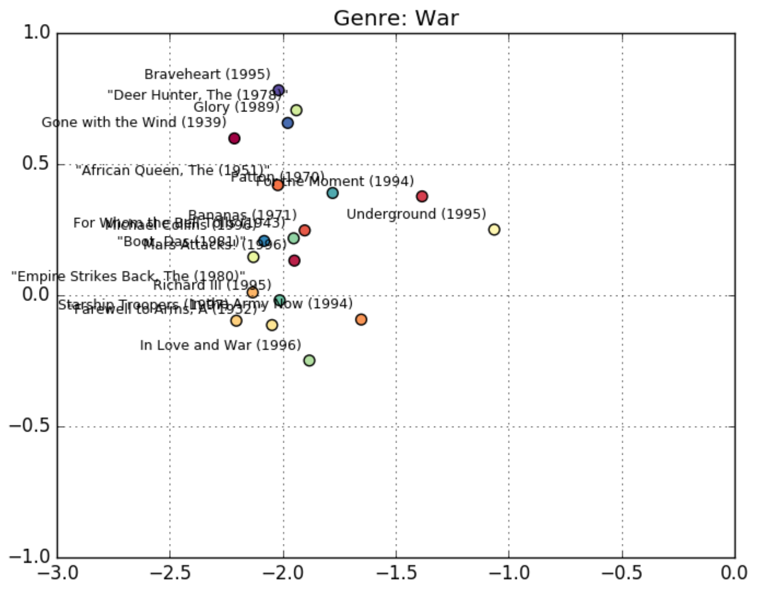
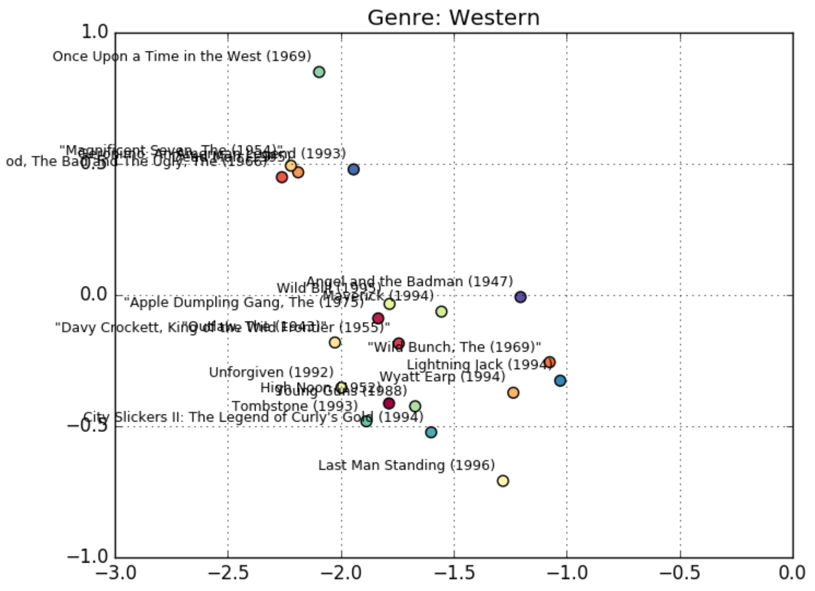


Figure 13: Visualization of Western movies.



4 Conclusion

Division of work

Shari did basic visualization and made the pretty rainbow histograms!

Carly worked on matrix factorization and plotting the projections in 2D.

Discoveries

We concluded that the axes generated make no sense to humans, but that the model does preserve closeness in similar movies. The histogram visualizations helped us to understand that ratings are skewed positively for all movies, and the visualizations of the matrix factorization helped us see that genres are a good indicator of a person's preference.

Challenges

Carly tried to import the data as usual, but kept coming across weird extra characters. After an hour and a single letter change to python's `open` command, the problem went away.

Shari forgot what a histogram was and had trouble getting the function to work, because she was trying to provide the wrong kinds of data. But we figured it out, and got very pretty rainbow graphs in the end.

Because the axes change each time we perform matrix factorization (randomness and local equilibria), we added a feature to save our matrices and reload them. This also saved on time because we didn't need to recompute everything.

Concluding Remarks

This project reminded us just how much we love Wallace & Gromit. After we're done with finals, we'll probably go watch that.

Our code is available at this github repo.