Recommender Systems

Signal Data Science

In this assignment, we'll explore one way to make a recommender system, something which predicts the rating a user would give to some item. Specifically, we'll be using collaborative filtering on the MovieLens 1M Dataset, a set of one million different movie ratings. Collaborative filtering operates on the assumption that if one person A has the same opinion as another person B on item X, A is *also* more likely to have the same opinion as B on a *different* item Y than to have the opinion of a randomly chosen person on Y.

Collaborative filtering is a type of unsupervised learning and can serve as a *prelude* to dimensionality reduction (*e.g.*, with PCA or factor analysis) because filling in missing values is typically required for such methods. Specifically, we will be working with an imputation-based method of collaborative filtering, which infers *all* of the missing values from the given data.

In the following, write up your work in an R Markdown file with elaboration about *what* you're doing at each step and *why* you're doing it. Include interpretation of results as well whenever appropriate. Your goal should be to produce, at the end, an HTML (or PDF) file from the R Markdown writeup that gives a coherent and reasonably accessible description of the process you followed, the reasoning behind each step, and the results attained at the end.

Getting started

We'll first need to spend some time preparing the data before we can use any collaborative filtering methods.

- Download the MovieLens 1M Dataset. Read the associated `README.txt`, which describes the contents of the dataset.
- The first 5 lines of `ratings.dat` are:

  ```
  1::1193::5::978300760
  1::661::3::978302109
  1::914::3::978301968
  1::3408::4::978300275
  1::2355::5::978824291
  ```

Use `read.csv()` with the appropriate options to load the file into R. (Note that the `sep` parameter only accepts a single character.) The resulting data frame should have **1000209 rows** and **7 columns**.

- Restrict to the columns containing user IDs, movie IDs, and movie ratings. Name the columns appropriately.
- Compute the sets of `unique()` user IDs and movie IDs as well as the mean rating given. Compare the numbers of different user IDs and movie IDs with the *maximum* user ID and movie ID.
- Set the seed to **3** for consistency. Generate a training set using 80% of the data and a test set with the remaining 20%.

Because there are some movies which are rated by very few people and some people who rated very few movies, we have two corresponding problems: (1) there will be movies in the test set which were not rated by any people in the training set and (2) there will be people in the test set who do not show up in the training set. As such, we need to add to the training set a fake movie rated by every user and a fake user who rated every movie.

- Create two data frames corresponding to the above fake data using the previously calculated mean rating. (For the fake movie and user respectively, use a movie ID and user ID which are both 1 greater than their respective maximum values in the entire dataset.) When creating the fake user who has rated every movie, allow the movie IDs to range from 1 to the maximum movie ID in the dataset (which will include movie IDs not present in the dataset). The fake user should not have a rating for the fake movie.
- Perturb the ratings of the fake data slightly by adding a normally distributed noise term with mean 0 and standard deviation 0.01. Add your fake data to the training data frame, which should increase in size by **9992 rows**.

Next, we need to create a matrix containing rating data for (user, movie) pairs. We can store this as a *sparse* matrix, which is a special data structure designed for handling matrices where only a minority of the entries are filled in (because each user has only rated a small number of movies).

- Use `Incomplete()` to generate a sparse ratings matrix with one row per user ID and one column per movie ID (including the perturbed fake data). The resulting matrix should have **6041 rows** and **3953 columns**. (*Hint:* The arguments of `Incomplete()` should be three vectors of the same length such that the *i*th value of each of the three vectors corresponds to a particular (user ID, movie ID, rating) data point in the training set.)

Using collaborative filtering

We will proceed to use the method of alternating least squares (ALS) via `softImpute()` to fill in the missing entries of the sparse ratings matrix. See *Notes on Alternating Least Squares* for an exposition of the technique.

Preparing the data

First, we need to prepare our data and calculate what values of the regularization parameter λ we'll search over.

- Use `biScale()` to scale both the columns and the rows of the sparse ratings matrix with `maxit=5` and `trace=TRUE`. You can ignore the resulting warnings (increasing the number of maximum iterations doesn't improve the outcome, which you can verify for yourself).

`lambda0()` will calculate the lowest value of the regularization parameter which gives a zero matrix for $\mathbf{D}$, *i.e.*, drives all rating estimates to zero.

- Use `lambda0()` on the scaled matrix and store the returned value as `lam0`.
- Create a vector of λ values to test by (1) generating a vector of 20 *decreasing* and uniformly spaced numbers from `log(lam0)` to `log(1)` and then (2) calculating e^x with each of the previously generated values as x. You should obtain a vector where entries 1 and 5 are respectively 103.21 and 38.89.

Finally, we need to initialize some data structures to store the results of our computations.

- Initialize a data frame `results` with three columns: `lambda`, `rank`, and `rmse`, where the `lambda` column is equal to the previously generated sequence of values of λ to test. Initialize a list `fits` as well to store the results of alternating least squares for each value of λ.

Imputation via alternating least squares

We are now ready to impute the training data with alternating least squares. For each value of λ, we will obtain as a result of `softImpute()` factor scores for every movie and every user. As described above, we can then use those to *impute* the ratings in the test set and calculate a corresponding RMSE to evaluate the quality of the imputation in order to determine the optimal amount of regularization.

- Iterate through the calculated values of λ. For each one, do the following:

- Use `softImpute()` with the current value of λ on the scaled sparse ratings matrix. In order to reduce computation time and find a low-dimensionality solution, constrain the rank of $\mathbf{D}$ to a maximum of 30. `rank.max=30` to restrict solutions to a maximum rank of 30 and `maxit=1000` to control the number of iterations allowed. For all but the first call of `softImpute()`, pass into the `warm.start` parameter the *previous* result of calling `softImpute()` to reduce the required computation time via a "warm start". Read the documentation for details on what these parameters mean.
- Calculate the *rank* of the solution by (1) rounding the values of the diagonal matrix $\mathbf{D}$ (stored in `$d`) to 4 decimal places and (2) determining the number of nonzero entries in the rounded matrix.
- Use `impute()` to calculate ratings for the test set using the results of `softImpute()`. (Pass in to `impute()` the calculated matrix decomposition as well as the user and movie ID columns in the test set.) You don't need to worry about unscaling the predictions: `impute()` will automatically take care of that. Calculate the corresponding RMSE between the predicted ratings and the actual ratings.
- Store the output of `softImpute()` in the previously initialized list `fits` as well as the calculated rank and RMSE in the corresponding row of the `results` data frame. Print out the results of the current iteration as well.

You should find that the minimum RMSE is attained at approximately $\lambda \approx 20$ with an RMSE of approximately 0.858.

- Store the best-performing soft-thresholded SVD (*i.e.*, the result of the best `softImpute()` call) into a variable called `best_svd`.

Evaluation metrics for collaborative filtering

Previously, we used the RMSE to evaluate the quality of our predicted ratings. We'll briefly explore several other ways to evaluate the output of a collaborative filtering algorithm.[1]

Aside from RMSE, we can also look at the mean absolute error (MAE), defined as

$$\text{MAE}(\mathbf{x}, \mathbf{y}) = \frac{1}{n} \sum_{i=1}^{n} |x_i - y_i|.$$

[1] See Lee *et al.* (2012), A Comparative Study of Collaborative Filtering Algorithms and the `softImpute` vignette for more detail.

- Add a column `mae` to `results` with the MAE corresponding to each value of λ. Which value of λ minimizes the MAE?

The RMSE and MAE are the two most commonly used evaluation metrics for collaborative filtering algorithms. In practice, only one of the two is chosen, since they yield fairly similar results.

We can also *classify* ratings that exceed a certain threshold as corresponding to movie to recommend to the user and those which do not as corresponding to movies to *not* recommend, turning our regression problem into a classification problem. This allows us to use precision and recall as evaluation metrics.

Precision, also known as the positive predictive value, is defined as the fraction of all recommended items which were correctly recommended, whereas **recall**, also known as sensitivity, is defined as the fraction of liked items which were actually recommended.

- Using the mean rating in the entire training set as the threshold value, add `precision` and `recall` columns to `results`. Which values of λ maximize the precision and recall?

Finally, we can use an *asymmetric cost function*, motivated by the thought that it is substantially worse to highly recommend a bad movie than to underrate a good movie (because in the former case the user may suffer through the movie whereas in the latter case they don't know what they're missing). Since ratings are given on a 1–5 scale, we can define such a cost function as, *e.g.*,

$$L(t,p) = \mathbf{L}_{t,p} \text{ given } \mathbf{L} = \begin{pmatrix} 0 & 0 & 0 & 7.5 & 10 \\ 0 & 0 & 0 & 4 & 6 \\ 0 & 0 & 0 & 1.5 & 3 \\ 3 & 2 & 1 & 0 & 0 \\ 4 & 3 & 2 & 0 & 0 \end{pmatrix},$$

where t is the true rating, p is the predicted rating rounded to the closest integer from 1–5, and $\mathbf{L}_{t,p}$ denotes the entry in the tth row and pth column of the matrix $\mathbf{L}$. (Given a vector of predicted ratings, we take the sum of $L(t,p)$ evaluated for each value.) We see that the cost of predicting a rating of 5 for a movie which was actually rated 1 is 10, the highest cost in the entire matrix, whereas the cost of predicting a rating of 1 for a movie which was actually rated 5 is only 4, less than half of the cost for the other way around.

- Add a column `asym` to `results` with the asymmetric cost function described above. Which value of λ minimizes the aymmetric cost?

An alternative method is to look at *implied rankings*. For a given set of predicted ratings, we can calculate for each user a value corresponding to how well the ranking of movies implied by the predicted ratings matches up with the ranking of movies implied by the user's actual ratings. For instance, we could calculate Spearman's rank correlation coefficient between the two sets of rankings for each user and take the average of the rank correlations. However, we won't be exploring this method here because the high number of users increases the computation time required and the fact that users haven't rated very many movies on average decreases its overall effectiveness.

Analyzing the results

Now that we have good results from running alternating least squares, we can do some further analysis of the MovieLens dataset. There are many aspects of the data to explore, so feel free to perform your own analyses if you have any ideas of your own!

Predicting user careers and movie genres

We'll begin by using the computed "factors" to look at different movie genres.

- As with the ratings dataset, load the movies dataset (in `movies.dat`) and name the columns appropriately.
- How many different genres are listed in the dataset? (You may find `strsplit()` helpful.) There is a single genre which is obviously the result of a data entry error. Add an appropriately named column for all of the *other* genres to serve as an *indicator variable* for whether each movie belongs to a particular genre. Fill in the entries of those columns accordingly.
- Restrict to movies which were listed at least once in the ratings dataset.

Examine the dimensions of the calculated matrix $\mathbf{V}$ in `best_svd`. The ith row corresponds to the movie with ID i and the jth column represents the "scores" for the jth "movies factor" (loosely speaking). We're interested in analyzing these "factors". To that end:

- Subset `best_svd$v` with the movie IDs in the movie dataset which remain after removing rows corresponding to movie IDs not present in the movies dataset. (Pay attention to the data type of the movie ID column, which is loaded in as a *factor*.) After doing so, add the factor columns to the data frame created from the movies dataset.[2]

Next, we'll illustrate one possible path of analysis by looking at the "Drama" genre.

- Examine the correlation between the indicator variable for movies tagged as dramas and the factor columns. Using `glm()`, run an unregularized logistic regression of the indicator variables against the factors.

[2] Something like `movies = cbind(movies, best_svd$v[as.numeric(as.character(movies$mid)),])`. (Be sure to understand what this code does!)

- Use `CVbinary()` (from the DAAG package) on the resulting model to generate *cross-validated probability predictions* for the whole dataset (stored in `CVbinary(fit)$cvhat`). Plot the associated ROC curve and calculate the AUC.

We now have a *probability* for each movie corresponding to how likely it is to be a drama or not given the information stored in the factor variables.

- Create a new data frame with (1) movie titles, (2) the indicator variable for dramas, and (3) the predicted probability for each movie. Order the rows from largest to smallest probability. Which movies are the most likely to be dramas and which movies are the most unlikely to be dramas? How well does this correspond with the actual genre labeling in the dataset?

- Repeat the above analysis for 3 other genres of your choice.

Similar to the movie genres, the users dataset (in `users.dat`) includes information about the *occupation* of each movie rater.

- Restrict to users 35 or older. Among those users, restrict to the 4 most common careers excluding "other" and "self-employed". Use unregularized multinomial logistic regression to predict career in terms of the factors for each user in $\mathbf{U}$. Run principal component analysis on the resulting log-odds values; plot and interpret the loadings of the principal components.

Estimating different careers' genre preferences

In the previous section, we were able to make career and genre predictions in terms of the factor variables. We'll begin a more complex analysis by attempting to calculate a sort of "vector of characteristic factor scores" for each movie genre.

- With the previously created indicator variables for movie genres, run an unregularized logistic regression for each genre against the factor variables.

- Set the seed for consistency. For each of the fitted models, use `CVbinary()` to generate cross-validated probability estimates of genre membership for each movie in the dataset.

- Convert the estimated probabilities to log odds via $L = \log(P/(1-P))$.

- For each genre, calculate a *linear combination* of the vectors of factor scores over the entire movies datasets with the coefficients being each movie's log odds of genre membership. That is, (1) multiply the factor scores corresponding to each movie by that movie's log odds and (2) take the sum of all the scaled vectors of factor scores.

  More precisely, suppose that we have m movies and f factor variables, that the ith movie's factor scores are given by $\mathbf{s}_i = (s_{i,1}, s_{i,2}, \ldots, s_{i,f})$, and that the ith movie's predicted log odds of inclusion into a particular genre is given by l_i. Then our goal is to calculate the sum $\sum_i l_i \mathbf{s}_i$, a process which we repeat for each of the different genres.

- Bind all of the vectors calculated in the previous step into a single data frame, with one column per genre and one row per factor. Set the column names to match the corresponding genres. Call the resulting data frame `genre_scores`.

By combining the factor scores of individual movies in the fashion described above, we have obtained factor scores for each *genre*.

- For the users dataset, use `dummy.data.frame()` (in the `dummies` package) to expand out the column of career codes into a set of indicator variables such that a career coded as n corresponds to a column titled `career_n`.
- Repeat the above process for the users dataset to obtain vectors of factor scores for each *career*. Call the resulting data frame `career_scores`. (Remember to give the columns descriptive names.)

Recall that predicted ratings are generated in the following fashion: if a user has factor scores $\mathbf{u} = (u_1, u_2, \ldots, u_f)$, a movie has factor scores $\mathbf{m} = (m_1, m_2, \ldots, m_f)$, and the values on the diagonal of $\mathbf{D}$ are given by d_i, our predicted rating is given by

$$r = \sum_{i=1}^{f} u_i d_i m_i.$$

We can perform the exact same calculation for the factor scores we have calculated for genres and careers.

- Initialize a matrix `pairings` with one row per career and one column per genre.
- Using the factor scores calculated for genres and careers, the matrix decomposition in `best_svd`, and the above equation, fill in each entry of the `pairings` with the corresponding predicted rating.
- Plot the values of `pairings` with `corrplot()` and interpret the results.

We can adjust for both (1) the differences in the mean ratings given to each genre and (2) the differences in the mean ratings given out by members of each career by scaling both the columns and the rows of `pairings` to have mean 0.

- Use `biScale()` to scale the columns and rows of `pairings`, setting `row.scale=FALSE` and `col.scale=FALSE` to preserve the original variances. Plot the scaled matrix with `corrplot()` and interpret the results as well as differences with the previous plot.

It's also possible to scale the variances of the rows and columns as well to make the output of `corrplot()` prettier, although in doing so we throw away yet even more information about how genres or careers differ from one another.

- Use `biScale()` to scale the rows and columns of `pairings` to have mean 0 and variance 1. Plot the scaled matrix with `corrplot()` and interpret the results as well as differences with previous plots.
- How does this method of analysis differ from just calculating summary statistics from the unimputed data, *i.e.*, looking at the average rating given by members of career X to movies in genre Y for each (X, Y) pair?

Estimating each career's specific movie preferences

More insight into our results so far can be gained by focusing in on specific careers and looking at which movies members of that career particularly like or dislike. Let's begin by just looking at writers.

- Use `complete()` to generate the fully imputed matrix $\mathbf{Z}$.[3]
- With the users dataset, run an unregularized logistic regression for the indicator variable corresponding to writers using the factor variables as predictors.
- Use `CVbinary()` to generate a cross-validated probability estimate for each user being a writer. Convert the probabilities into log odds.

Next, we want to calculate a single rating for each movie that represents how much *writers specifically* like that movie. Unlike in the previous section, we'll have to do some additional normalization (by dividing by the *sum* of the log odds for each linear combination of ratings) to keep our values in the correct numerical range.

- Calculate a *linear combination* of the rows of $\mathbf{Z}$ with the coefficients being the previously obtained log odds. That is, (1) multiply the ith row of $\mathbf{Z}$ by the ith log odds and (2) take the sum of all of the scaled rows. Next, divide each linear combination by the *sum* of the log odds used as its coefficients. Call this linear combination `writer_ratings`.

We're interested in which movies writers *disproportionately* like or dislike. As such, we have to correct for the fact that each movie has a different average rating. (For example, movies which are extraordinarily bad will be disliked by *every* career, but we're more interested in those which writers specifically dislike more than others.) To that end:

- Subtract the mean of each column of $\mathbf{Z}$ from `writer_ratings`.

[3] The function call should look something like `complete(Incomplete(df$uid, df$mid), best_svd)`.

One last logistical concern: `writer_ratings` has a value for *every* possible movie ID from 1 to the maximum, but not every movie ID has a corresponding movie in the movies dataset.

- Subset `writer_ratings` by the column of movie IDs in the movies dataset. Again, pay attention to the data type of the movie IDs column, taking care to handle factor conversion correctly.

Finally, we're ready to see which movies writers especially love or hate!

- Create a new data frame `writer_prefs` by combining the movies dataset with `writer_ratings` as a new column. Remove all columns aside from movie title, movie genre, and `writer_ratings`. Order the rows of `writer_prefs` from highest to lowest values of `writer_ratings`.
- Examine the top 10 and bottom 10 rows of `writer_prefs`. Do your results here correspond to the results obtained in the previous section?[4]
- Calculate the mean of `writer_ratings`. What is the interpretation of this value (both its magnitude and its sign)?

We can, of course, repeat the above process for other careers.

- Write a function which takes as input an integer `career_code` and performs the above analysis for the corresponding career, returning a reordered data frame of movies with three columns (movie title, movie genre, estimated career-specific rating).
- Choose 3 other careers to analyze in a similar matter. Interpret the results.

[4]The top movie for writers should be "North by Northwest" and the bottom movie should be "Toy Story".