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 **9994 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. The resulting matrix should have **6041 rows** and **3953 columns**.

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 **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 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 **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 **D** (stored in `$d`) to 4 decimal places and (2) calculating 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.) 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 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

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

- 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.

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

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.

- 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.

Predicting 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 movies not present in the ratings 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.
- 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.

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

Predicting user careers

Similar to the movie genres, the dataset of users (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.