James Henegan
Fall 2019

Introduction to Recommender Systems, with an Emphasis on Collaborative Filtering Methods

Recommender systems are based on the assumption that it is possible to infer a consumer's interests by utilizing various sources of data related to the consumer (e.g., purchasing history, preferences, etc.). Once the interests of the consumer have been estimated, recommendations can be given. We say that the *user* is the person who receives the recommendation. We say that the *item* is the product that is being recommended. Data are often given to the recommender system by the user in the form of feedback. For example, a user may indicate their attitude towards an item by providing a rating for that item. Intuitively, if a user gives an item a high rating, then this indicates that the item is of high value to the user. A good recommender system can help drive sales for a business, and it can help a user find (heretofore unknown) items that will be of high value to the user.

There are two main ways of formulating the recommendation problem. In one version of the problem, the goal is to predict the rating value for each user-item combination. Here, we assume that some training data is available, indicating user preferences for (past, previously encountered) items. This problem is sometimes referred to as the *matrix completion problem*. Often times, it is neither necessary nor practical to predict ratings for all of the possible user-item combinations. Instead, a recommender system may recommend the top-k items for a particular user. This version of the problem is called the *top-k recommendation problem*.

Here are sine of the basic models of recommender systems:

- **Collaborative Filtering Methods**. Based on information provided by several users. We will discuss these methods in more detail shortly.
- **Content-Based Recommender Systems**. Based on descriptive attributes of items. *Example.* If a user likes the movie *Predator* [attributes: ("action", "Schwarzenegger")], then the user might also like *The Terminator*.
- **Knowledge-Based Methods**. These are often based on *user requirements*. Useful for items that are not bought very often. *Example.* I need a heating technician in the Jackson-metro area who can work with floor furnaces.
- **Hybrid and Ensemble-Based Recommender Systems**. The methods mentioned above can be combined.

For the rest of the report, we will focus on Collaborative Filtering Methods. These methods are based on data involving user-item interactions. The data may be represented in the form of a *ratings matrix*.

	Item 1	Item 2	Item 3	Item 4
User 1	5	0	3	4
User 2	1	4	?	1
User 3	4	1	4	3
User 4	0	?	1	0
User 5	5	1	3	?

These may be difficult to implement because, often times, the underlying ratings matrices are sparse. The ratings which have been specified are called observed ratings. Meanwhile, we can use the terms unobserved or missing to refer to the the unspecified ratings. Broadly speaking, Collaborative Filtering Methods can either be *Neighborhood-based* or *Model-based*. Whether neigborhood based or model based, the main idea behind collaborative filtering methods is that the unobserved ratings can be predicted by taking advantage of the fact that the observed ratings are often highly correlated across various users and items.

For neighborhood-based methods, the ratings of user-item combinations are predicted on the basis of their so-called *neighborhoods*. These neighborhoods could be constructed in terms of either users or items.

- **User-based collaborative filtering**. To predict ratings for a *target user* A, we would first gather up all the ratings provided by a *peer group*: users who are "similar" to A. We would then use these ratings to predict the ratings for A.
- **Item-based collaborative filtering**. Suppose we wanted to predict the rating of item B by user A. We would gather up all the ratings that A gave for items that were similar to B. We would then use these ratings to predict the ratings for A.

For an example of a model-based method, we can apply decision trees to the collaborative filtering problem, but, when we do so, we are faced with two issues.

- There is no clear distinction between the dependent and independent variables.
- The ratings matrix is extremely sparse.

It is easy to work around the first issue by constructing a decisions tree for each item (or for each user). Meanwhile, it is possible to work around the second issue by taking advantage of certain dimensionality reduction methods. These can take a sparse ratings matrix M and return a dense, low-dimensional representation of M in terms of latent factors. This low-dimensional representation can be computed with either Principal Component Axis-type methods or Singular Value Decomposition-type methods.

Comments. An example of item-based collaborative filtering is provided on the next page. Most of the information in this report is based on Aggarawal's *Recommender Systems*, which was published by Springer in 2016.

Example of Item-based collaborative filtering. The data sets for this example may be found at the following website :

`https://www.kaggle.com/rounakbanik/the-movies-dataset`

For our purposes, we will need to download the following files: `ratings_small.csv` and `movies_metadata.csv` from the website given above.

The following R code will create a dataset containing movie titles, user ID's, and user ratings for various movies, based on the data mentioned above.

```
library(tidyverse)

movies_metadata <-
  read_csv("Data/movies_metadata.csv")

ids_and_titles <-
  movies_metadata %>%
  select(id, original_title)

ratings_small <-
  read_csv("Data/ratings_small.csv") %>%
  select(-timestamp)

titles_users_ratings <-
  ids_and_titles %>%
  full_join(ratings_small,
            by = c("id" = "movieId"))  %>%
  drop_na() %>%
  select(original_title, userId, rating)

titles_users_ratings

# A tibble: 44,994 x 3
   original_title userId rating
   <chr>           <dbl>  <dbl>
 1 Heat               23    3.5
 2 Heat              102    4
 3 Heat              232    2
 4 Heat              242    5
 5 Heat              263    3
 6 Heat              311    3
 7 Heat              363    4
 8 Heat              387    5
 9 Heat              452    4.5
10 Heat              505    3.5
# ... with 44,984 more rows
```

We can now export this data to a `.csv` file, which can be analyzed in Python.

```
titles_users_ratings %>%
  write_csv("demo.csv")
```

The following Python code will create a ratings matrix based on the given reviews.

```
import pandas as pd
df = pd.read_csv("demo.csv")
userRatings = df.pivot_table(index=['userId'],
                             columns=['original_title'],
                             values='rating')
```

The following code computes the correlations for user-provided ratings for each pair of movies. The option `min_periods` lets us ignore movies with a small number of ratings.

```
corrMatrix = userRatings.corr(
                method='pearson',
                min_periods=25)
```

Now suppose that that a user has provided some ratings for some movies. For an example, we can take a user's ratings from our ratings matrix.

```
myRatings = userRatings.loc[6].dropna()
```

Now, we are going to (i) retrieve movies similar to the ones rated by the user, (ii) scale each movie's similarity score (correlation) according to how well the user rated the movie, and (iii) add the score to a list. Below, the first line of code initializes this list.

```
simCandidates = pd.Series()
for i in range(0, len(myRatings.index)):
    # get movies similar to ones rated by user
    sims = corrMatrix[myRatings.index[i]].dropna()
    # scale similarity score according to user rating
    sims = sims.map(lambda x: x * myRatings[i])
    # add to list
    simCandidates = simCandidates.append(sims)
```

Next, we add up the results, sort, and remove the movies the user has already seen.

```
# Add up scores
simCandidates = simCandidates.groupby(
    simCandidates.index).sum()
# Sort
simCandidates.sort_values(
    inplace = True,
     ascending = False)
# Remove Movies Already Seen by user
key_diff = set(
    simCandidates.index).difference(
        myRatings.index)
where_diff = simCandidates.index.isin(
    key_diff)
recommendations = simCandidates[where_diff]
# Print top 5 from list of recommendations
recommendations.head(5)
```

Here are the results:

```
Rope                            20.263889
Blood: The Last Vampire         19.572526
The Man with the Golden Arm     19.418102
Arlington Road                  17.227141
Bridge to Terabithia            15.643330
dtype: float64
```

References.

1. Aggarwal, Charu C. *Recommender Systems.* Springer, 2016.
2. Kane, Frank. *Machine Learning, Data Science, and Deep Learning with Python.* Udemy Course.