CS6120 Continuous Assessment, 2013–2014

Derek Bridge

13th February, 2014

1 Introduction

You are given four files of data, as follows:

- `movies.txt`: Each line of the file describes a movie, including its title and genre(s).
- `users.txt`: Each line of the file describes a user, including some demographic data.
- `debug.txt`: Each line of the file contains one rating. This file contains only 20 ratings. You should use this file to test the Python that you write.
- `ratings.txt`: Each line of the file contains one rating. This file contains 100,000 ratings. You should use this file to run experiments.

You are also given one file of Python, as follows:

- `ims.py`: This contains some Python which you can freely use to make it hugely easier for you to run experiments.

Do *not* edit these files.

Write Python to run experiments on the data. Ideally, your experiments will make *comparisons*, e.g.:

- between different algorithms (e.g. user-based, item-based, other);
- between different formulae within the algorithms (e.g. the three formulae for making user-based predictions presented in lecture 6, or other formulae you find in the research literature);
- between different parameterisations (e.g. different values for k);
- between different evaluation criteria (i.e. criteria other than just MAE); and
- between predictions and recommendations.

Present your results in an *iPython notebook*. Give precise, concise explanations of what is being compared, what is being measured, and your experimental methodology.

Review the relevant research literature and discuss it in your notebook. For example, you might discuss the usefulness of MAE as a measure of the quality of a prediction algorithm; or the difficulty of evaluating a recommendation algorithm; or other ways of evaluating prediction/recommendation systems. Your research may even help you to design new experiments (different algorithms, different formulae, different evaluation criteria, etc.) that you can run and whose results you can include in your notebook.

Those of you who are less comfortable with programming can, to a limited degree, compensate for less coding & experimentation by including a deeper and broader discussion, with more extensive use of the research literature. Similarly, those of you who are less comfortable with analytic writing can, to a limited degree, compensate for less discussion by including more programming and rigorous experimentation.

All elements (the Python you write, the experimental results, and discussion of the research literature) must be present in your notebook. You will receive no credit for elements that are not part of your notebook.

When to start? Start now!

You may be running dozens and dozens of experiments. Once you are using the full dataset of 100,000 ratings, each may take 10 minutes or more to run. So it's unwise to leave everything to the last minute. There is no way you can hope to run experiments and bring it all together in the final few hours before the deadline.

2 Format

The only acceptable format for your work is an iPython notebook.

Your notebook should comprise the following:

1. a heading that includes your name and student id;
2. a short abstract (200 words or so);
3. an introduction;
4. one or more sections and subsections presenting your discussion of the research literature, your Python code, and your experimental results;
5. a final section that offers conclusions and ideas for future work; and
6. a list of references, i.e. sources cited in the body of your notebook.

Feel free to use tables, diagrams, charts and graphs to make the presentation of your work more vivid.

Note that the references section is not a list of things you've read. It is a list of things you've cited in the body of your notebook.

3 Academic Integrity

This is an *individual* assignment. The work you submit must be your own. In no way, shape or form should you submit work as if it were your own when some or all of it is not.

Collusion: Given how much freedom there is in the assignment, everybody's work will be different. It will be obvious if there is collusion. All parties to collusion will be penalized.

Deliberate plagiarism: You must not plagiarise the programs, results, writings or other efforts of another student or any other third-party. Plagiarism will meet with severe penalties, which can include exclusion from the University.

Inadvertent plagiarism: In reporting your exploration of the research literature be careful to avoid inadvertent plagiarism (e.g where 'paraphrases' of the source material are too close to the original).

Small amounts of material may be quoted directly, where the exact wording of the original needs to be conveyed in your paper. But in these cases, the material must be presented within quotation marks; the quoted material must be followed by an immediate citation to your references section; and the work must be listed in your references section.

Even when not quoting directly, be scrupulous to use citations to acknowledge the influence of the research literature and to add support to claims that you make. Here again a citation should be given immediately, and the work must be listed in your references section.

Falsification and fabrication: The experimental results reported in your notebook must come from the experiments that you have run. Do not falsify or fabricate results.

Your notebook will be checked for signs of collusion, plagiarism, falsification and fabrication. You may be called to discuss your submission with me and this will inform the grading, any penalties and any disciplinary actions.

You may, of course, ask me questions. I may share questions and answers with the class, if I feel they are general matters, for example, of clarification. But I will also discuss with you questions that relate to your own Python, your experiments and your reading and, in the interest of giving you proper credit for your endeavours, these will not be shared with the class.

4 Submission

- Ensure the name of your notebook comprises your name and student id, e.g. `sam_smiles_113123456.ipynb`;
- Create a folder, also labelled with your name and student id (`sam_smiles_113123456`);

- Place your notebook (e.g. `sam_smiles_113123456.ipynb`) and any image files into this folder;
- Compress the folder, which, on the Macs in the multimedia laboratories, is done as follows:
  - select the folder;
  - select *Compress* from the *File Menu* thus generating a new file with the `.zip` extension (e.g. `sam_smiles_113123456.zip`);
- Submit the compressed folder via Moodle.

When grading your work, I will use the *original* versions of `movies.txt`, `users.txt`, `debug.txt`, `ratings.txt` and `ims.py`.

The assignment must be submitted by **5p.m. Thursday 27th March, 2014**.

Where work is submitted up to and including 7 days late, 5% of the total marks available shall be deducted from the mark achieved. Where work is submitted up to and including 14 days late, 10% of the total marks available shall be deducted from the mark achieved. Work submitted 15 days late or more shall be assigned a mark of zero.

5 Acknowledgement

I am grateful to the those associated with the GroupLens Project (`www.grouplens.org`) for making the data available.

Appendix

Here is a description of some functions that I have written for your use. They are all in the file called `ims.py`. (Technically speaking, for the benefit of those with a greater knowledge of programming, what I have done is define a class and what I am describing here are methods, not functions.)

Before you can use these functions in your notebook, you must include a Code cell that contains the following:

```
import ims
rec = ims.Recommender()
```

This reads in data about the movies (from `movies.txt`) and the users (from `users.txt`) and does some other housekeeping.

5.1 Loading the ratings

5.1.1 `rec.load_ratings(filename, test_percentage = 30, seed = None)`

This reads in ratings from a file. It partitions the ratings randomly into a training set and a test set.

`filename` is a string: the name of the file that contains the ratings. Use `"debug.txt"` while debugging your Python; use `"ratings.txt"` for your experiments. `test_percentage` is a non-negative integer: the percentage of the ratings that it will place in the test set; the rest go into the training set. Note that it is approximate: the test set may not contain *exactly* this proportion of the ratings. `seed` is a non-negative integer. It is optional. Use it while debugging your Python. It enables you to get the same random split of the data on multiple runs of the program.

For example, this is what you might use while debugging your Python. It reads in the small ratings file and it ensures the same random split every time you run it:

```
rec.load_ratings("debug.txt", 30, 1)
```

And this, for example, is what you might use when you are ready to run experiments:

```
rec.load_ratings("ratings.txt", 30)
```

5.1.2 `rec.get_test_ratings()`

Gets all the test ratings.

The result is a list, containing each rating from the test set, in no particular order.

Each rating is a dictionary, whose keys are as follows:

- `user_id`: the user
- `movie_id`: the movie
- `rating`: the rating

5.2 User-Based Collaborative Recommenders

5.2.1 `rec.get_k_nearest_users(similarity_function, k, active_user_id, candidate_movie_id = None)`

Gets the `k` nearest users to `active_user_id`. It uses `similarity_function` to compute the similarity of users to `active_user_id`. Optionally, if `candidate_movie_id` is not `None`, the set of neighbours is confined to those who have rated `candidate_movie_id`.

The result is a list containing the nearest neighbours, in no particular order. The length of this list will be no more than `k` and may be less than `k` if there are insufficient users who both have movies in common with `active_user_id` and have rated `candidate_movie_id`.

Each neighbour in the result list is represented as a dictionary, whose keys are as follows:

- `user_id`: the neighbour's id
- `sim`: the degree of similarity between `active_user_id` and `user_id`

- `rating`: `user_id`'s rating for `candidate_movie_id`

The `rating` is only included if `candidate_movie_id` was not `None`.

5.2.2 `rec.get_thresholded_nearest_users(similarity_function, threshold, active_user_id, candidate_movie_id = None)`

This is identical to the function in 5.2.1 except it has `threshold` instead of `k`. It gets all users whose degree of similarity to `active_user_id` exceeds `threshold` and who, optionally, have rated `candidate_movie_id`.

5.2.3 `rec.get_k_thresholded_nearest_users(similarity_function, k, threshold, active_user_id, candidate_movie_id = None)`

This is identical to the function in 5.2.1 except it has *both* `k` and `threshold`. It gets the `k` nearest users to `active_user_id` provided their similarity to `active_user_id` exceeds `threshold` and who, optionally, have rated `candidate_movie_id`. Again, the result list may contain fewer than `k` neighbours.

5.3 Item-Based Collaborative Recommenders

5.3.1 `rec.get_k_nearest_movies(similarity_function, k, candidate_movie_id, active_user_id = None)`

Gets the `k` nearest movies to `candidate_movie_id`. It uses `similarity_function` to compute the similarity of movies to `candidate_movie_id`. Optionally, if `active_user_id` is not `None`, the set of neighbours is confined to those movies which have been rated by `active_user_id`.

The result is a list containing the nearest neighbours, in no particular order. The length of this list will be no more than `k` and may be less than `k` if there are insufficient movies who both have users in common with `candidate_movie_id` and have been rated by `active_user_id`.

Each neighbour in the result list is represented as a dictionary, whose keys are as follows:

- `movie_id`: the neighbour's id
- `sim`: the degree of similarity between `candidate_movie_id` and `movie_id`
- `rating`: `active_user_id`'s rating for `candidate_movie_id`

The `rating` is only included if `active_user_id` was not `None`.

5.3.2 `rec.get_thresholded_nearest_movies(similarity_function, threshold, candidate_movie_id, active_user_id = None)`

This is identical to the function in 5.3.1 except it has `threshold` instead of `k`. It gets all movies whose degree of similarity to `candidate_movie_id` exceeds `threshold` and who, optionally, have been rated by `active_user_id`.

5.3.3 rec.get_k_thresholded_nearest_movies(similarity_function, k, threshold, candidate_movie_id, active_user_id = None)

This is identical to the function in 5.3.1 except it has *both* k and threshold. It gets the k nearest movies to candidate_movie_id provided their similarity to candidate_movie_id exceeds threshold and who, optionally, have been rated by active_user_id. Again, the result list may contain fewer than k neighbours.

5.4 Other

5.4.1 rec.get_user_movie_rating(user_id, movie_id)

Gets user_id's rating for movie_id from the training set or None if this user has no rating for this movie in the training set.

5.4.2 rec.get_user_ratings(user_id)

Gets all of user_id's ratings from the training set as a list. If this user has no ratings in the training set, an empty list is the result. Each rating in the result list is represented as a dictionary, whose keys are as follows:

- movie_id: the movie id
- rating: user_id's rating for movie_id

5.4.3 rec.get_user_mean_rating(user_id)

Gets the mean of user_id's ratings from the training set. If this user has no ratings in the training set, the mean is None.

5.4.4 rec.get_user_demographics(user_id)

Gets all of user_id's demographic data. The result is a dictionary, whose keys are as follows:

- age: a positive integer
- gender: either 'M' or 'F'
- occupation: one of 'administrator', 'artist', 'doctor', 'educator', 'engineer', 'entertainment', 'executive', 'healthcare', 'homemaker', 'lawyer', 'librarian', 'marketing', 'none', 'other', 'programmer', 'retired', 'salesman', 'scientist', 'student', 'technician', or 'writer'
- zipcode: a string of digits

5.4.5 `rec.get_demographic_ratings(age = None, gender = None, occupation = None, zipcode = None)`

Gets all ratings from the training set for users whose demographics matches the values in the arguments. For example, the following puts a list of all ratings from female students into variable `rs`:

```
rs = rec.get_demographic_ratings(gender = 'F', occupation = 'student')
```

The result is a list, containing each rating, in no particular order. Each rating is a dictionary, whose keys are the same as for 5.1.2.

5.4.6 `rec.get_movie_ratings(movie_id)`

Gets all of `movie_id`'s ratings from the training set as a list. If this movie has no ratings in the training set, an empty list is the result. Each rating in the result list is represented as a dictionary, whose keys are as follows:

- `user_id`: the user id
- `rating`: `user_id`'s rating for `movie_id`

5.4.7 `rec.get_movie_mean_rating(movie_id)`

Gets the mean of `movie_id`'s ratings from the training set. If this movie has no ratings in the training set, the mean is `None`.

5.4.8 `rec.get_movie_descriptors(movie_id)`

Gets all of `movie_id`'s descriptors. The result is a dictionary, whose keys are as follows:

- `title`: a string
- `release_date`: a string
- `video_release_date`: a string
- `url`: a string containing the URL of this movie in the IMDb
- `genres`: a list of 19 integers, each either 0 or 1. The 19 genres are: unknown, Action, Adventure, Animation, Children's, Comedy, Crime, Documentary, Drama, Fantasy, Film-Noir, Horror, Musical, Mystery, Romance", Sci-Fi, Thriller, War, and Western. Note that many movies have more than one genre, i.e. more than one of the 19 integers will be set to 1.

5.4.9 `rec.get_genre_ratings(genre)`

Gets all ratings from the training set for movies of the given genre. For example, the following puts a list of all ratings for Horror movies into variable `rs`:

```
rs = rec.get_genre_ratings('Horror')
```

The result is a list, containing each rating, in no particular order. Each rating is a dictionary, whose keys are the same as for 5.1.2.