Recommender Systems Project Specification

Barry Smyth, February 5th, 2016

COMP 30490/41440

This project specification is for students taking COMP 30490/41440 Collective Intelligence. It is a compulsory project for all students.

The overall project completion date is March 30th 2016 but there are a number of compulsory intermediate checkpoint submissions, which will be examined and graded during the course of the project. Failure to complete these intermediate submissions will result in a significant loss of marks (see Project Grading).

All project submissions will be via Moodle.

In what follows we will describe the detailed project specification with a week by week breakdown of what is required.

Aims & Objectives

The aim of the project is to build a working user-based collaborative filtering system that is capable of producing rating predictions for a well-known dataset (MovieLens). This dataset will be provided.

During the course of the next few weeks you will gradually build a working collaborative filtering system and conduct a series of experiments to evaluate its performance. You will learn about the practical aspects of building a recommender system: representing and understanding user and item data; making efficient predictions; evaluating prediction performance; running large-scale experiments.

Each week you will be submitting source code and a short report (typically 1-2 pages). Most weeks you will also submit one or more data-files with your results data. Your code submission each week should also include a README.txt file as a guide to your submission. This README must include clear instructions about how to compile and run your code.

It is important to note that each week builds on the work of the previous week. The tasks for each week will be evaluated and graded during the week that follows submission and feedback will be provide directly. It is therefore important for students to complete each weekly task in the timeframe indicated (typically 1 week). See Project Grading for information about the late submission policy.

This does not prevent those who are motivated to do so from getting ahead by working on tasks for future weeks as soon as they wish; this document provides a complete week-by-week breakdown of the tasks. However, it is important that you hold off on submitting the results of any accelerated work until the week that those tasks fall due.

This also does not prevent you from refactoring and improving your code from week to week. For example you might decide in week 3 that you need to better design some core data structures that formed the basis for your implementation in weeks 1 and 2. This is perfectly fine and you should not feel limited by your own legacy code in this regard. Improving your code in this way may improve your code-quality marks as the weeks go on but code submitted in previous weeks will not be regarded.

It is crucial that you focus on completing the core requirements each week but there is of course room to innovate and optional improvements can be included if they make sense and are supported with evidence.

At the same time don't overdo it. Do not get carried away. You need to manage your time carefully to avoid falling behind on the weekly tasks especially given the many competing deadlines that you will all be facing. There are only 50 marks available for this practical and you cannot get more no matter how much you do. In addition you will start your next practical, the GWAP project, in a couple of weeks and so your workload will quickly build.

The MovieLens Dataset

MovieLens (ML) is a well-known recommendation system produced by the GroupLens research group in the University of Minnesota. Briefly, the ML datasets provide researchers with access to *user-item ratings* data, which has been collected over an extended period of time by the ML group.

The ML data takes the form of a comma-separated-value (csv) file. Each line of the file refers to a single rating provided by some user for some item at a given time. As such each line of the file includes four values:

1. *user_id* – a unique numeric user id.
2. *item_id* – a unique numeric item (movie) id.
3. *rating* – a star rating value from 1 to 5 (1 = low, 5 = high).
4. *timestamp* – indicating the time the rating (not used in this practical).

For example, below is a sample of the first 5 lines of the *ratings.dat* file provided today. It shows 5 ratings for users 196, 186, 22, 244, and 166 and items, 242, 302, 377, 51, and 346. We can see ratings 3, 3, 1, 2, and 1 for these items, respectively. The final value in each line is the timestamp.

```
196,242,3,881250949
186,302,3,891717742
22,377,1,878887116
244,51,2,880606923
166,346,1,886397596
  :           :
  :           :
```

The MovieLens team provides access to other data-files. These describe further details of users (gender etc) and items (genre, year etc), but for the purpose of this collaborative filtering project they will not be needed.

There have been a number of different MovieLens dataset releases over the years. They differ in terms of the scale of the ratings data provided and the type of supplementary data that is offered. For the purpose of this project we will be using the 100k ratings dataset.

This is not quite big data scale but nor is it insignificant either. Your code will need to be reasonably efficient to run experiments on this dataset. Careful design decisions early on will likely pay dividends later.

Project Timetable and Milestones

Week 1 – Understanding your Data (February 10, 2016)

This first week of the project is about understanding your data. You will be reading in a ML data-file in a csv format (comma-separated-values). You will need to think about how best to represent the ratings data for use in collaborative filtering and you will start to plan and implement the core data structures that you will use for the remainder of the project.

The main tasks in week 1 include:

- Read in ratings data from ratings.csv.
- Design suitable data structures for storing and manipulating ratings data for collaborative filtering*.
- Generate the following key statistics from the data:
    - Total number of users, items, ratings.
    - Calculate a ratings density metric.
    - Mean, median, standard deviation, max, min ratings per user.
    - Mean, median, standard deviation, max, min ratings per item.
    - Total number of ratings for each of the 5 ratings classes.

Submit a short report and source code supporting the above tasks. This submission (via Moodle) is due by the end of day on February 17th, 2016.

**Think carefully now about how you decide to implement the core data structures for these ratings. Conceptually you have a user x item matrix with each cell storing a rating. You will need to get easily able to access the rating for a given user-item pair so it's tempting to think about a 2D array for this. But the matrix is very sparse indeed so this is not a great idea. Equally you will need to be able to efficiently access the ratings of a given user and the ratings for a given item. It is fine for your data structures to evolve over the course of the project but the sooner you identify an efficient approach, the better.*

Week 2 – Prediction Testing (February 15, 2016, Double Lab Week)

The task for this week is to prepare for collaborative filtering by implementing a default prediction technique and by building the testing infrastructure. For this you will implement a *leave-one-out* (L1O) testing strategy as discussed in lectures. Briefly, you will use each (user, item, rating) tuple as a test target. This means that you will be using your various (future) prediction techniques to generate a predicted rating for the user-item pair. By comparing your predicted rating to the user-item's actual rating you can compute an error value for each prediction.

The main tasks for this week are:

- As a baseline prediction technique design a function/method *mean_item_rating(user_id, item_id)* which, for a given *user_id, item_id* pair will return the average rating for this item calculated across all ratings for this item, other that the rating given by *user_id*.*
- Can you use this technique to predict a rating for *every* existing user-item pair?
- Implement a L1O testing loop using the *root-mean-squared-error* (RMSE) error metric discussed in lectures.
- Use L1O to test predictions using the *mean_item_rating* method from last week.
- Create a csv data-file from the results of this L1O test. This csv should have a line per user-item pairing in the following elements: *user_id*, *item_id, actual rating, predicted rating, RMSE.*
- Calculate an overall RMSE value as an average of the individual RMSE values computed for the entire L1O test.
- Calculate an overall coverage value for the L1O test as the percentage of user-item pairs for which a recommendation can be calculated.
- Finally calculate the average runtime for 1 complete L10 test-cycle (that is a complete L1O test over all of the ratings data); to do this you should time say 10 separate L1O runs and get the average runtime.

Submit a short report, data-files and source code supporting the above tasks. This submission is due by the end of day on February 24th, 2016.

**Caution*: *when predicting a rating for user u and item i you need to make sure that the actual rating for user u and item i is not participating in the prediction. In other words, when making a prediction using mean_item_rating* the rating for item *i should be calculated as the mean rating over all users who have rated i, except user u.*

Week 3 – Collaborative Filtering 1, Distance-Based Sim. (February 24, 2016)

Over the last week you generated your first predictions, albeit using a simple popularity-based, average rating metric that was not particularly personalized for the user. This week you will implement your first true collaborative filtering technique using the *distance-based* metric discussed in lectures.

The main tasks for this week include:

- Implement distance-based user similarity metric discussed in lectures and use this as the basis for prediction*.
- Experiment with different approaches for computing the target user's neighbourhood as the basis for this distance-based prediction; e.g. by varying neighbourhood size, minimum neighbour overlap, or other suitable parameters. Test at least 4 different neighbourhood settings; for example if you decide to vary neighbourhood size then use 4 different sizes.
- Run a series of L1O tests to evaluate the impact of neighbourhood construction on overall RMSE. For example, if you are varying the size of the neighbourhood (*n*) then you will generate different RMSE and coverage values for different values of n. Produce a results csv file for each.
- In your report this week you should describe the setup and results of this experiment. Clearly describe the setup in terms of the dataset, prediction approach, how you varied the neighbourhoods. Present the RMSE and coverage results as graphs of overall RMSE and coverage versus different neighbourhood size (or whatever neighbourhood parameter you varied). Also include the average runtime for each L10 variation (averaged over 10 runs). Discuss what these results tell you about the relationship between neighbourhoods and error, coverage, runtime. Is it as you expected? If so why; if not why not? How do these predictions compare to the results you achieved for the average-rating metric from last week?
- Optionally, if you implement any improvements on the standard distance-based metric please make these clear in your report and provide supporting evidence as to their performance benefits.

Submit a short report, the data-files and source code supporting the above tasks. This submission is due by the end of March 2nd, 2016.

**Caution*: *as before, when predicting a rating for user u and item i you need to make sure that you do not corrupt the prediction process by allowing the user's own rating to participate in the prediction. In other words, when making a prediction for user u and* item *i,* item *i's* actual rating by *u should not participate in the prediction process. This is generally true for all of the prediction techniques that you will use.*

Week 4 – Collaborative Filtering 2, Cosine Similarity (March 2, 2016)

Last week you ran your first proper collaborative filtering experiment, comparing a simple user-based prediction technique to our average-rating baseline. This week you will try a different user similarity metric, *cosine similarity*, as discussed in class.

The main tasks for this week include:

- Implement the cosine similarity metric and used it to generate predictions for user-item pairs.
- Rerun your L1O tests from last week to compare the accuracy and coverage of cosine-based predictions to your distance-based predictions for different neighbourhoods. Produce a results csv file for each of the L1O runs.
- In your report, describe the results of this new experiment and include a comparison to last week's data. You can include the distance-based and cosine-based prediction error data on the same graph. Include the usual error, coverage, and timing data. Discuss what these results tell you.
- Optionally, if you implement any improvements on the standard distance-based or cosine metrics please make these clear in your report and provide supporting evidence as to their performance benefits.

Submit a short report and source code supporting the above tasks. This submission is due by the end of day on March 9th, 2016.

This is a good week to review your implementation to-date. At this stage you have tested 3 different prediction techniques and your code should able to accommodate different prediction functions/methods without the need for a lot of repetition. If you find that you are repeating a lot of code each time you implement a new prediction technique then it might be worth looking at some refactoring of your code.

Week 5 – Collaborative Filtering 3, Resnick's Formula (March 7, 2016, Double Lab Week)

This week you will implement the well-known *Resnick* prediction formula, discussed in lectures, and which uses correlation instead of similarity or distance as the basis for prediction.

This weeks tasks include:

- Implement the Resnick prediction technique and used it to generate predictions for user-item pairs in the usual way.
- Rerun your L1O tests to compare the accuracy and coverage of these new predictions to your previous predictions, again for different neighbourhoods. Produce a results csv file.
- In your report, describe the results of this new experiment and include this comparison to the previous weeks' data. You can include the distance-based and cosine-based prediction error data on the same graph as Resnick's technique and similar for coverage results. Include the usual error, coverage, and timing data. Discuss what these results tell you.
- Optionally, if you implement any improvements on the standard distance-based, cosine or Resnick metric please make these clear in your report and provide supporting evidence as to their performance benefits.

Submit a short report and source code supporting the above tasks via Moodle. This submission is due by the end of day on March 30th, 2016.

This completes the coding element of this project.

Project Grading

In total there are 50 marks going for this practical, for both 5-credit and 10-credit students. The workload is expected to take no more than 35 hours in total and should roughly break down into 5 weeks of 7 hours of effort per week.

The work in each of the 5 weeks is assigned 10 marks. Each week includes implementation work and some report writing, with 8 marks going for the implementation and 2 for the corresponding report.

Late submissions will incur a 10% penalty per day or part thereof (including weekends). Therefore a submission that is 3.5 days overdue will be marked out of a total of 6 (instead of 10 for a 40% penalty) marks for that week.

Implementation & Code Quality Marks

For the implementation work marks will be distributed in a manner that is proportionate to the effort involved for the various subtasks in a given week. In general 75% of the implementation marks will be given for any code that compiles and that performs the required task and produces the requested data in the right format. The remaining 25% will awarded based on the quality of the code that is produced.

There is not a lot of code to be written in reality but you will need to make good design decisions early on if you are to succeed in this regard. Think carefully about this as you begin the project because if you develop the right structure in the first 2 – 3 weeks the second half of the practical will be a lot more straightforward.

To achieve full marks for your implementation you must clearly demonstrate strong software engineering and good coding practice: you code must compile; it must be well designed, elegant, and efficient; it must be understandable as it relates to the project specification; it must be diligently commented.

Report Marks

Obviously it will be difficult if not impossible to turn in a good report unless you have completed the bulk of the implementation work for a given week. You are not required to complete a long or detailed report. Remember only 20% of the marks for a given week are going for the report, which therefore corresponds to a level of effort of approximately 90 minutes per week and the report format does not change from week to week so there are significant efficiencies to be gained, in subsequent weeks, if you develop a good reporting style from the start. It is likely that your weekly reports will be 1-2 pages in length and certainly no more than 3-4 pages in any given week. A clear and concise presentation style is an advantage.

Below is a copy of the standard evaluation report that the demonstrators will complete after each submission date. You will receive one of these within one week of the submission date to help you understand how you are doing and whether there are any areas for improvement.

Marking Report for Week x (date completed)

Student name: xxx

Days Late? 0

Total Marks: / 10

Final Mark: xxx (adjusted for late submission if relevant)

Implementation Mark: / 6

Comments to support the implementation mark.

- *Does the code compile and run?*
- *Identify any implementation objectives that were not satisfactorily achieved.*
- *Does the code produce the right results?*
- *Was a correct result file(s) produced? Identify any problems.*

Code Quality Mark: / 2

Comments to support the code quality mark.

- *Is there a clear README file?*
- *Is the code well organized?*
- *Is it adequately commented?*
- *How well designed does the code it appear to be?*
- *Comment on any obvious deficiencies or significant inefficiencies.*
- *Suggest ways to improve the code (if any).*

Report Mark: / 2

Comments to support the report mark.

- *Do the report contents adequately cover what was requested?*
- *Is the report competently written, well organised and properly presented?*
- *Are results presented in a scientific fashion with adequate description of the experimental setup and methodology?*
- *Does the student understand the implication of their results? Have they provided an insightful discussion of the core findings*
- *Suggest ways to improve the report (if any).*