TÉCNICO LISBOA

Mathematics for Machine Learning

2022-2023

Project 1 - Matrix Factorization and Collaborative Filtering

We are all familiar with recommendation systems: we encounter them while browsing Netflix, Youtube or Amazon. There are in fact two types of algorithms for recommendation systems: Content-based filtering and collaborative filtering. Content-based filtering uses information about the entity and the user: For instance, if you watched a movie made by a certain director, or from a certain genre, it will recommend other films with that director/genre. On the other hand, collaborative filtering uses similarities between users and items to provide recommendations via ratings. In fact it detects the item features (and user preferences) implicitly. And all that with just Matrix Factorization! Pretty neat, right?

In this project you will use a real-world database and build from scratch a recommendation system. It will also serve as an introduction to the framework we will use on this course: Jupyter Labs. For this project you will need a Jupyter notebook and Python. You can either install locally on your machine, or use a web browser.

You have to make by **March 6th** a progress presentation (see below) and deliver by the deadline of **March 20th** a 6 to 8 page report including explanation of what you did, graphics, analysis, and discussion of your results, together with the commented code of your project. This assignment is **individual**.

1- Start by reading (Y. Koren, R. Bell 2009) for a nice and simple introduction to the types of techniques you will be using in this project. The list of references is at the end.

2- Go to https://jupyter.org/ and follow the instructions to install a version of Jupyter IDE with Python, either locally or remotely. You will also need to install the Python Numpy library, and a library for the plots, like Matplotlib. **No other libraries should be used!**

3- (4 points) Create a an algorithm for low rank approximation using svd-based factorization of a matrix. You can use *numpy.linalg.svd* for this. Calculate the reconstruction error in different matrix metrics, and in the sum of squared error of all the known ratings, as k increases. Try your algorithm on small matrices (from 2 x 2 up to 6 x 6 - not necessarily squared), and check it works. Show plots of the error depending on the k. On **March 6** be prepared to show your algorithm working in the Lab. **This is a graded exercise!**

4- Now we go to the real world. You will need to obtain a ratings dataset. A possibility is the Movie Lens Small Latest Dataset (https://www.kaggle.com/shubhammehta21/movie-lens-small-latest-dataset) which contains 100836 ratings for 9742 movies. These data were created by 610 users between March 29, 1996 and September 24, 2018, with each user having rated at least 20 movies. It also contains other data, such as film tags. If you want to explore a different ratings dataset, it is perfectly possible. Note: When importing the dataset, notice that the movie ids are not sequential, and you should **not** use them directly as index for the matrix entries. You need to also randomly remove 10% of the given ratings and use them later as a test set to check the accuracy of your model predictions…

5- In the dataset you have data to build a Ratings $m \times n$ matrix R. This will be a sparse matrix, where the entry R_{ij} represents what rating user j gave to item i . Our objective is, given a small k, to find (dense) matrices $M_{m \times k}$ and $U_{k \times n}$ such that

$$R = M\,U.$$

The idea is to "predict" the missing values, so that it is possible to recommend to a user a movie our model predicts she will like. Of course, such a pair of factors might not exist. In that case we want to find a pair that minimizes the error $|| R - M\,U ||$ for some kind of metric.

Start by writing some functions to import the csv file and build the matrix from the list of ratings. The missing values can be represented as 0. Then create a first recommendation system from a svd-based factorization algorithm you implemented earlier (2 points).

6- While the original ratings matrix has non-negative entries, the factors can have negative entries. There are good arguments for trying to obtain non-negative factors (check the references below). In that case, we are in the domain of non-negative matrix factorization. (8 points) Choose and Implement two different Algorithms for non-negative matrix factorization (one of them alternating least squares) and apply it to the dataset, creating a second and third recommendation systems. (2 points) How do these ones compare with the previous one?

7- (4 points) When you have a factorization, you can think of the linear space between the matrices as the latent or embedding space. For instance, the Matrix M represents a "Movies" to "Movie Latent Features" operator. So, the projection of an existing movie into the latent space is just the corresponding line of M (which is of dimension k). This is the same as multiplying from the left a vector of dimension ~9000, with zeros everywhere, except a single "1" corresponding to the index of that movie. A new film (not in the dataset) would be represented as a linear combination of the known films, always as a vector of dimension ~9000. Multiplying on the right by M would project it to the Latent Space. The same with a user u is projected to the same latent space via the multiplication $U\,u$. If $k = 2$ our latent space has dimension 2 and we can represent the entities in this space. Project

50 movies you know with their names into the latent space, with the various recommendation systems you obtained, and check what insights it is possible to get.

8- Note that what you learned here is not constrained to recommendation systems. Matrices can be used to represent many other types of dyadic data (relating two types of entities), such as a term-document matrix in a document corpus or a pixel value-picture matrix for an image corpus. Also in these cases factoring the matrix can lead to the extraction of important information, used for document or image clustering, for example.

References

Rasmus Bro and Sijmen De Jong. 1997. A fast non-negativity-constrained least squares algorithm. Journal of Chemometrics 11, 5 (1997), 393–4

F. Maxwell Harper and Joseph A. Konstan. 2015. The MovieLens Datasets: History and Context. ACM Trans. Interact. Intell. Syst. 5, 4, Article 19 (Dec. 2015), 19 pages. https://doi.org/10.1145/2827872

Patrik O. Hoyer. 2004. Non-negative Matrix Factorization with Sparseness Constraints. J. Mach. Learn. Res. 5 (Dec. 2004), 1457–1469. http://dl.acm.org/citation.cfm?id=1005332.1044709

Y. Koren, R. Bell, and C. Volinsky. 2009. Matrix Factorization Techniques for Recommender Systems. Computer 42, 8 (Aug 2009), 30–37. https://doi.org/10.1109/MC.2009.263

Daniel D. Lee and Sebastian Seung. 2000. Algorithms for Non-negative Matrix Factorization. In Proceedings of the 13th International Conference on Neural Information Processing Systems (NIPS'00). MIT Press, Cambridge, MA, USA, 535–541. http://dl.acm.org/citation.cfm?id=3008751.3008829

Pentti Paatero. 1997. Least squares formulation of robust non-negative factor analysis. Chemometrics and Intelligent Laboratory Systems 37, 1 (1997), 23 – 35. https://doi.org/10.1016/S0169-7439(96)00044-5

Further Reading

Shulong Chen and Yuxing Peng. 2018. Matrix factorization for recommendation with explicit and implicit feedback. Knowledge-Based Systems 158 (2018), 109 – 117. https://doi.org/10.1016/j.knosys.2018.05.040