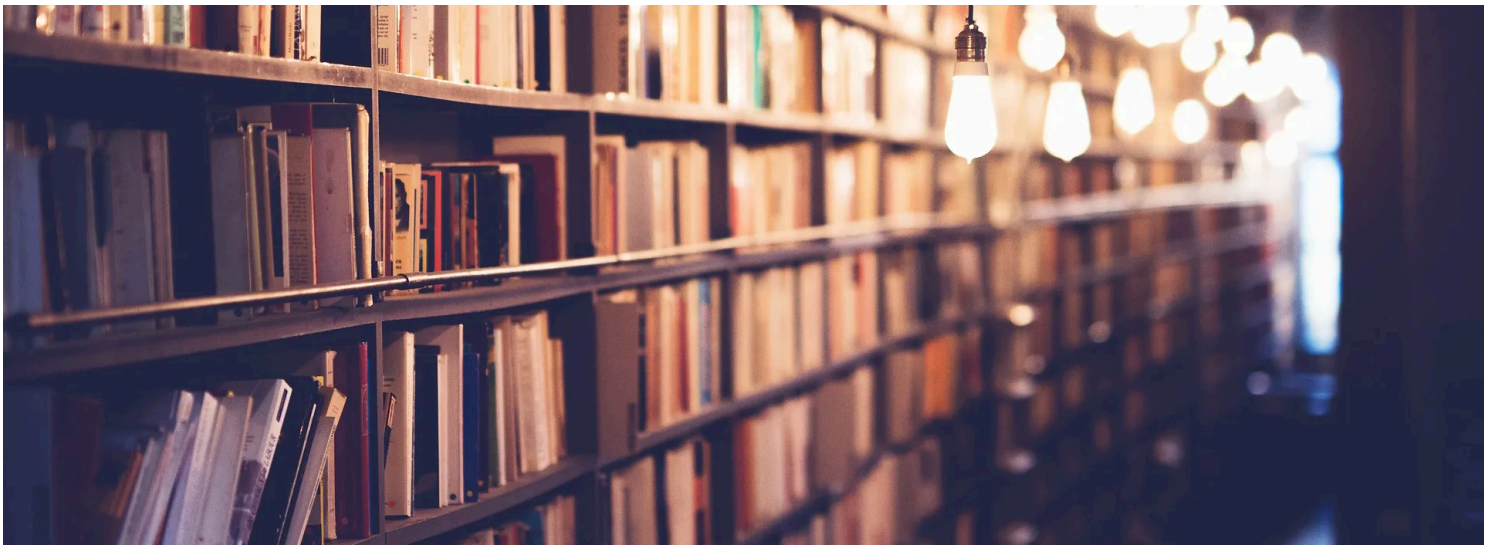
Introduction to recommender systems

Overview of some major recommendation algorithms.

Baptiste Rocca · Follow

Published in Towards Data Science · 22 min read · Jun 2, 2019

6K 18



Credit: StockSnap on Pixabay

This post was co-written with Joseph Rocca.

Introduction

During the last few decades, with the rise of Youtube, Amazon, Netflix and many other such web services, recommender systems have taken more and more place in our lives. From e-commerce (suggest to buyers articles that could interest them) to online advertisement (suggest to users the right contents, matching their preferences), recommender systems are today unavoidable in our daily online journeys.

In a very general way, recommender systems are algorithms aimed at suggesting relevant items to users (items being movies to watch, text to read, products to buy or anything else depending on industries).

Recommender systems are really critical in some industries as they can generate a huge amount of income when they are efficient or also be a way to stand out significantly from competitors. As a proof of the importance of recommender systems, we can mention that, a few years ago, Netflix organised a challenges (the “Netflix prize”) where the goal was to produce a recommender system that performs better than its own algorithm with a prize of 1 million dollars to win.

In this article, we will go through different paradigms of recommender systems. For each of them, we will present how they work, describe their theoretical basis and discuss their strengths and weaknesses.

Outline

In the first section we are going to overview the two major paradigms of recommender systems : collaborative and content based methods. The next two sections will then describe various methods of collaborative filtering, such as user-user, item-item and matrix factorization. The following section

will be dedicated to content based methods and how they work. Finally, we will discuss how to evaluate a recommender system.

Collaborative versus content

The purpose of a recommender system is to suggest relevant items to users. To achieve this task, there exist two major categories of methods : collaborative filtering methods and content based methods. Before digging more into details of particular algorithms, let's discuss briefly these two main paradigms.

Collaborative filtering methods

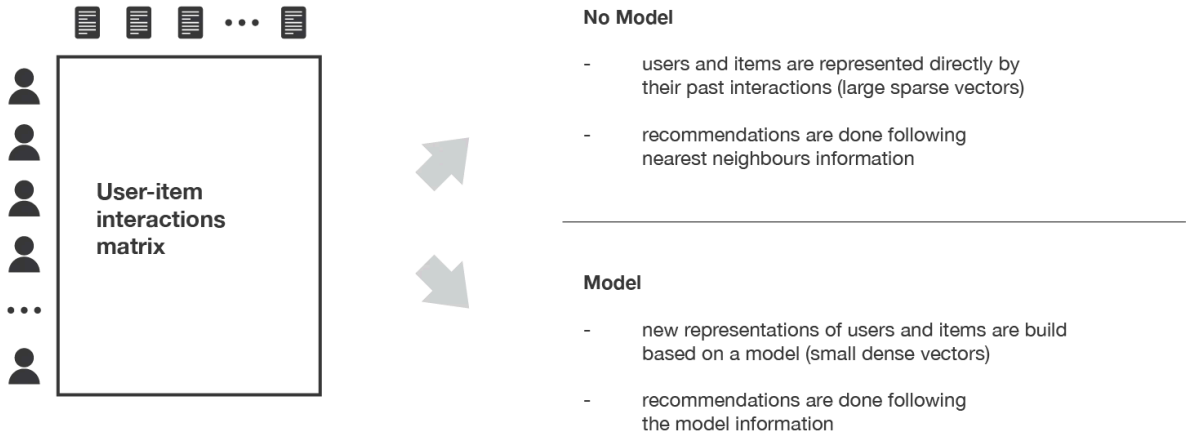
Collaborative methods for recommender systems are methods that are based solely on the past interactions recorded between users and items in order to produce new recommendations. These interactions are stored in the so-called "user-item interactions matrix".

Users	User-item interactions matrix	Items
suscribers	rating given by a user to a movie (integer)	movies
readers	time spent by a reader on an article (float)	articles
buyers	product clicked or not when suggested (boolean)	products
...		

Illustration of the user-item interactions matrix.

Then, the main idea that rules collaborative methods is that these past user-item interactions are sufficient to detect similar users and/or similar items and make predictions based on these estimated proximities.

The class of collaborative filtering algorithms is divided into two sub-categories that are generally called memory based and model based approaches. Memory based approaches directly works with values of recorded interactions, assuming no model, and are essentially based on nearest neighbours search (for example, find the closest users from a user of interest and suggest the most popular items among these neighbours). Model based approaches assume an underlying “generative” model that explains the user-item interactions and try to discover it in order to make new predictions.



Overview of the collaborative filtering methods paradigm.

The main advantage of collaborative approaches is that they require no information about users or items and, so, they can be used in many

situations. Moreover, the more users interact with items the more new recommendations become accurate: for a fixed set of users and items, new interactions recorded over time bring new information and make the system more and more effective.

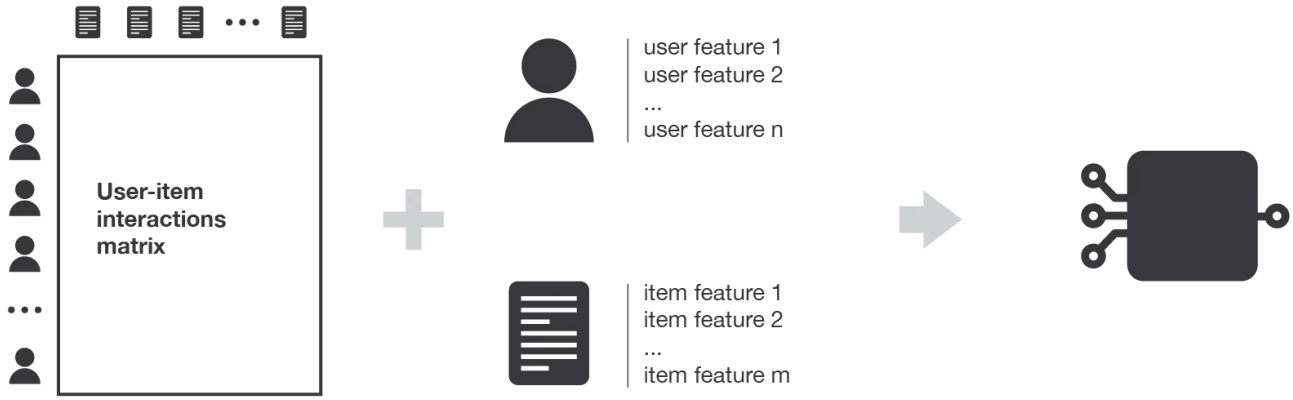
However, as it only consider past interactions to make recommendations, collaborative filtering suffer from the "cold start problem": it is impossible to recommend anything to new users or to recommend a new item to any users and many users or items have too few interactions to be efficiently handled. This drawback can be addressed in different way: recommending random items to new users or new items to random users (random strategy), recommending popular items to new users or new items to most active users (maximum expectation strategy), recommending a set of various items to new users or a new item to a set of various users (exploratory strategy) or, finally, using a non collaborative method for the early life of the user or the item.

In the following sections, we will mainly present three classical collaborative filtering approaches: two memory based methods (user-user and item-item) and one model based approach (matrix factorisation).

Content based methods

Unlike collaborative methods that only rely on the user-item interactions, content based approaches use additional information about users and/or items. If we consider the example of a movies recommender system, this additional information can be, for example, the age, the sex, the job or any other personal information for users as well as the category, the main actors, the duration or other characteristics for the movies (items).

Then, the idea of content based methods is to try to build a model, based on the available “features”, that explain the observed user-item interactions. Still considering users and movies, we will try, for example, to model the fact that young women tend to rate better some movies, that young men tend to rate better some other movies and so on. If we manage to get such model, then, making new predictions for a user is pretty easy: we just need to look at the profile (age, sex, ...) of this user and, based on this information, to determine relevant movies to suggest.



Open in app ↗ Sign up Sign in

Medium Search Write

Content based methods suffer far less from the cold start problem than collaborative approaches: new users or items can be described by their characteristics (content) and so relevant suggestions can be done for these new entities. Only new users or items with previously unseen features will logically suffer from this drawback, but once the system old enough, this has few to no chance to happen.

Later in this post, we will further discuss content based approaches and see that, depending on our problem, various classification or regression models can be used, ranging from very simple to much more complex models.

Models, bias and variance

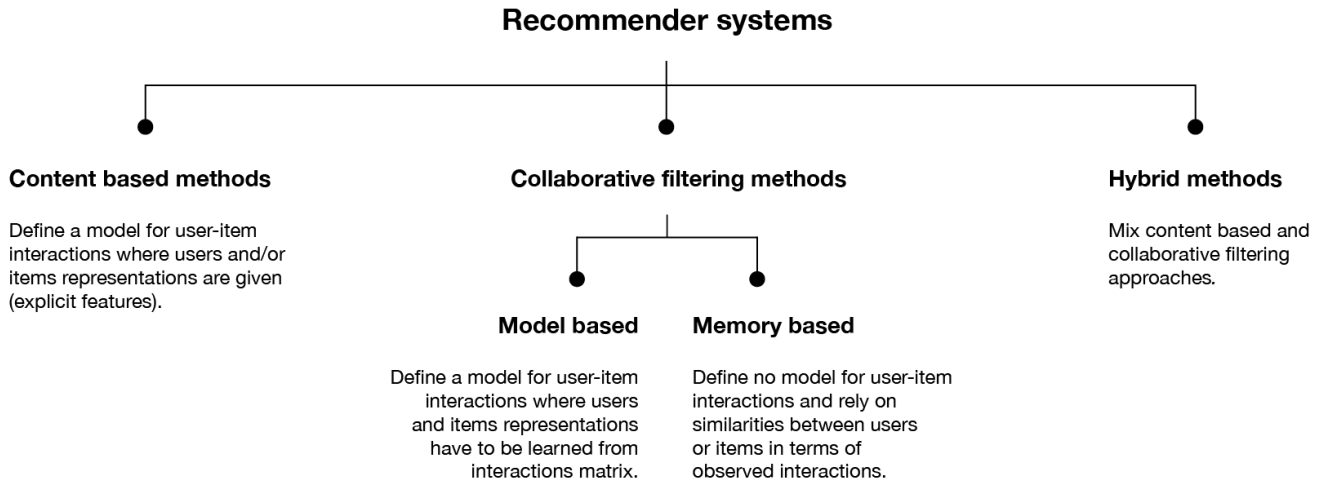
Let's focus a bit more on the main differences between the previously mentioned methods. More especially let's see the implication that the modelling level has on the bias and the variance.

In memory based collaborative methods, no latent model is assumed. The algorithms directly works with the user-item interactions: for example, users are represented by their interactions with items and a nearest neighbours search on these representations is used to produce suggestions. As no latent model is assumed, these methods have theoretically a low bias but a high variance.

In model based collaborative methods, some latent interaction model is assumed. The model is trained to reconstruct user-item interactions values from its own representation of users and items. New suggestions can then be done based on this model. The users and items latent representations extracted by the model have a mathematical meaning that can be hard to interpret for a human being. As a (pretty free) model for user-item interactions is assumed, this methods has theoretically a higher bias but a lower variance than methods assuming no latent model.

Finally, in content based methods some latent interaction model is also assumed. However, here, the model is provided with content that define the representation of users and/or items: for example, users are represented by given features and we try to model for each item the kind of user profile that likes or not this item. Here, as for model based collaborative methods, a

user-item interactions model is assumed. However, this model is more constrained (because representation of users and/or items are given) and, so, the method tends to have the highest bias but the lowest variance.



Summary of the different types of recommender systems algorithms.

Memory based collaborative approaches

The main characteristics of user-user and item-item approaches it that they use only information from the user-item interaction matrix and they assume no model to produce new recommendations.

User-user

In order to make a new recommendation to a user, user-user method roughly tries to identify users with the most similar “interactions profile” (nearest neighbours) in order to suggest items that are the most popular among these neighbours (and that are “new” to our user). This method is said to be “user-centred” as it represent users based on their interactions with items and evaluate distances between users.

Assume that we want to make a recommendation for a given user. First, every user can be represented by its vector of interactions with the different items (“its line” in the interaction matrix). Then, we can compute some kind of “similarity” between our user of interest and every other users. That similarity measure is such that two users with similar interactions on the same items should be considered as being close. Once similarities to every users have been computed, we can keep the k-nearest-neighbours to our user and then suggest the most popular items among them (only looking at the items that our reference user has not interacted with yet).

Notice that, when computing similarity between users, the number of “common interactions” (how much items have already been considered by both users?) should be considered carefully! Indeed, most of the time, we want to avoid that someone that only have one interaction in common with our reference user could have a 100% match and be considered as being “closer” than someone having 100 common interactions and agreeing “only” on 98% of them. So, we consider that two users are similar if they have interacted with a lot of common items in the same way (similar rating, similar time hovering…).

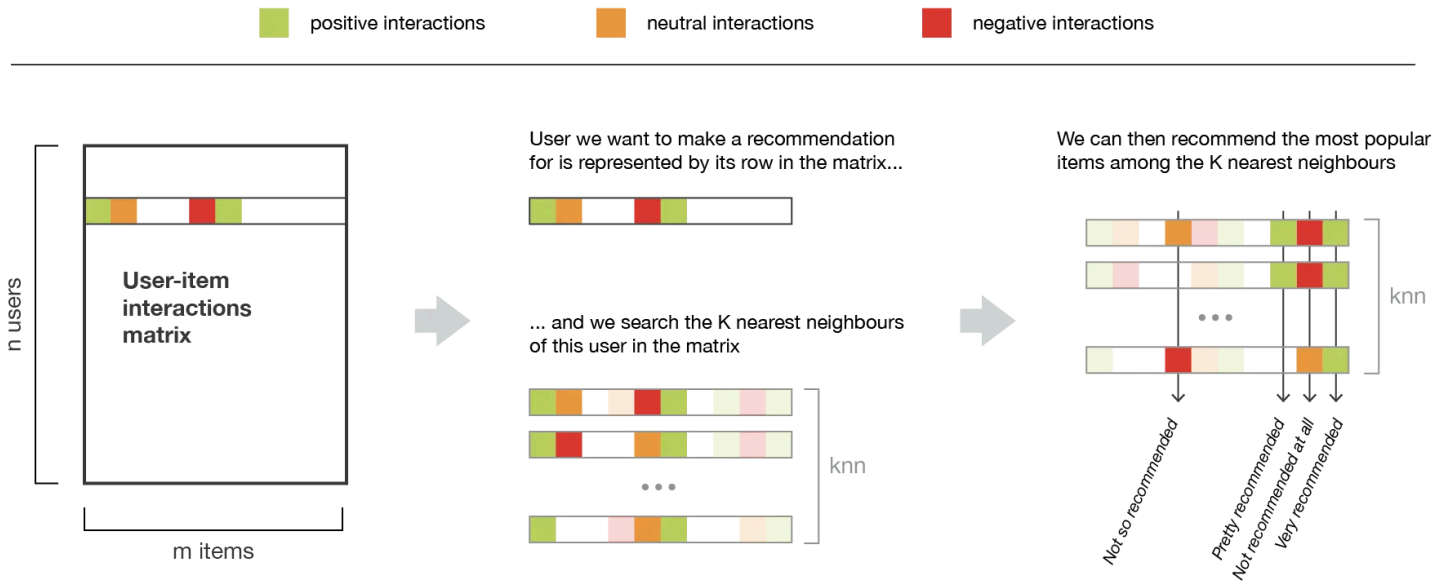


Illustration of the user-user method. The same colour code will be used in the remaining of the post.

Item-item

To make a new recommendation to a user, the idea of item-item method is to find items similar to the ones the user already "positively" interacted with. Two items are considered to be similar if most of the users that have interacted with both of them did it in a similar way. This method is said to be "item-centred" as it represent items based on interactions users had with them and evaluate distances between those items.

Assume that we want to make a recommendation for a given user. First, we consider the item this user liked the most and represent it (as all the other items) by its vector of interaction with every users ("its column" in the interaction matrix). Then, we can compute similarities between the "best item" and all the other items. Once the similarities have been computed, we can then keep the k-nearest-neighbours to the selected "best item" that are new to our user of interest and recommend these items.

Notice that in order to get more relevant recommendations, we can do this job for more than only the user's favourite item and consider the n preferred items instead. In this case, we can recommend items that are close to several of these preferred items.

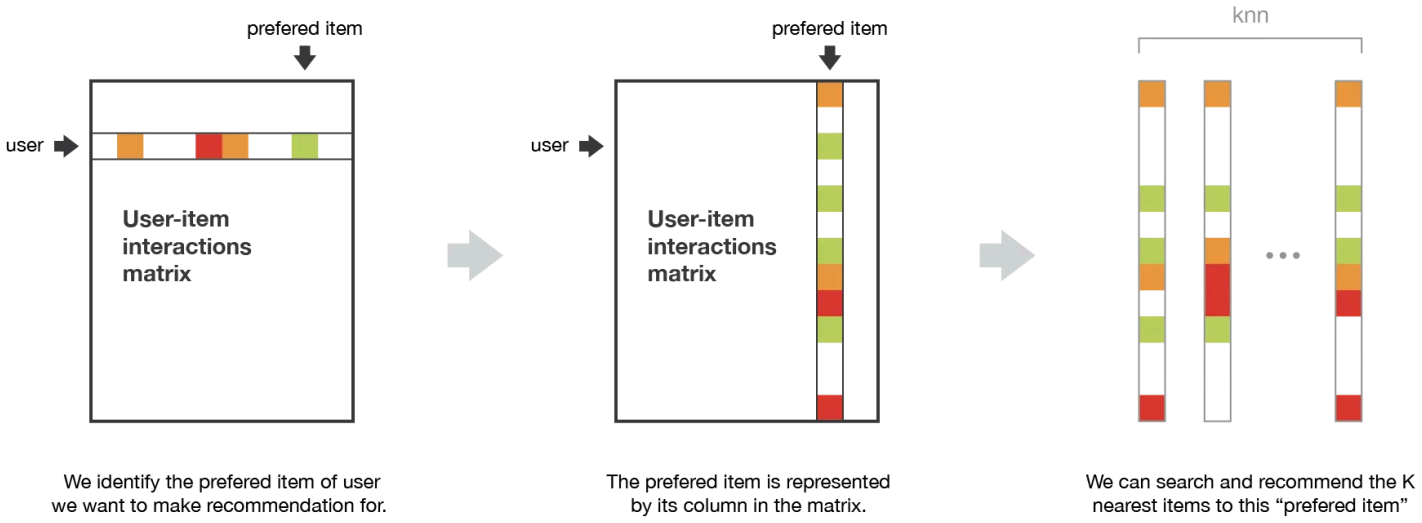


Illustration of the item-item method.

Comparing user-user and item-item

The user-user method is based on the search of similar users in terms of interactions with items. As, in general, every user have only interacted with a few items, it makes the method pretty sensitive to any recorded interactions (high variance). On the other hand, as the final recommendation is only based on interactions recorded for users similar to our user of interest, we obtain more personalized results (low bias).

Conversely, the item-item method is based on the search of similar items in terms of user-item interactions. As, in general, a lot of users have interacted with an item, the neighbourhood search is far less sensitive to single interactions (lower variance). As a counterpart, interactions coming from every kind of users (even users very different from our reference user) are

then considered in the recommendation, making the method less personalised (more biased). Thus, this approach is less personalized than the user-user approach but more robust.

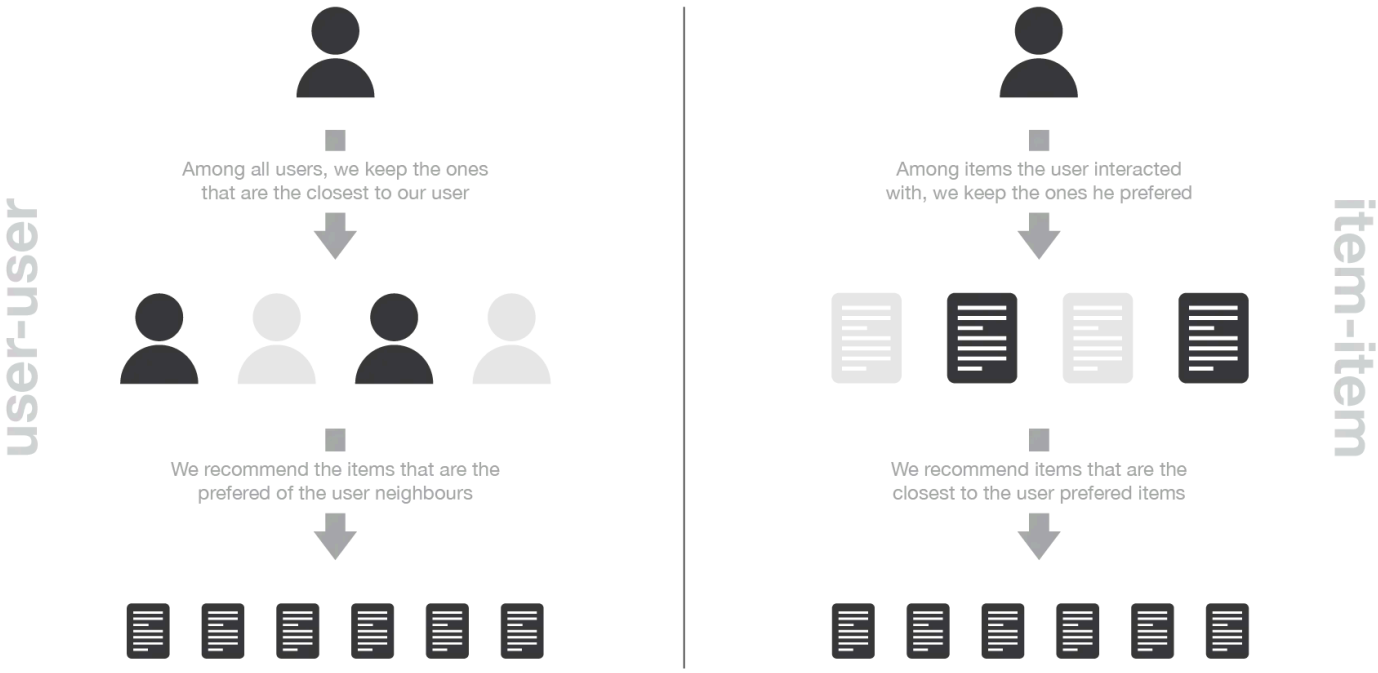


Illustration of the difference between item-item and user-user methods.

Complexity and side effect

One of the biggest flaw of memory based collaborative filtering is that they do not scale easily: generating a new recommendation can be extremely time consuming for big systems. Indeed, for systems with millions of users and millions of items, the nearest neighbours search step can become intractable if not carefully designed (KNN algorithm has a complexity of O(ndk) with n the number of users, d the number of items and k the number of considered neighbours). In order to make computations more tractable for huge systems, we can both take advantage of the sparsity of the interaction matrix when designing our algorithm or use approximate nearest neighbours methods (ANN).

In most of the recommendation algorithms, it is necessary to be extremely careful to avoid a “rich-get-richer” effect for popular items and to avoid getting users stuck into what could be called an “information confinement area”. In other words, we do not want that our system tend to recommend more and more only popular items as well as we do not want that our users only receive recommendations for items extremely close to the one they already liked with no chance to get to know new items they might like too (as these items are not “close enough” to be suggested). If, as we mentioned, these problems can arise in most of the recommendation algorithms, it is especially true for memory based collaborative ones. Indeed, with the lack of model “to regularise”, this kind of phenomenon can be accentuated and observed more frequently.

Model based collaborative approaches

Model based collaborative approaches only rely on user-item interactions information and assume a latent model supposed to explain these interactions. For example, matrix factorisation algorithms consists in decomposing the huge and sparse user-item interaction matrix into a product of two smaller and dense matrices: a user-factor matrix (containing users representations) that multiplies a factor-item matrix (containing items representations).

Matrix factorisation

The main assumption behind matrix factorisation is that there exists a pretty low dimensional latent space of features in which we can represent both users and items and such that the interaction between a user and an item can be obtained by computing the dot product of corresponding dense vectors in that space.

For example, consider that we have a user-movie rating matrix. In order to model the interactions between users and movies, we can assume that:

- there exists some features describing (and telling apart) pretty well movies.
- these features can also be used to describe user preferences (high values for features the user likes, low values otherwise)

However we don't want to give explicitly these features to our model (as it could be done for content based approaches that we will describe later). Instead, we prefer to let the system discover these useful features by itself and make its own representations of both users and items. As they are learned and not given, extracted features taken individually have a mathematical meaning but no intuitive interpretation (and, so, are difficult, if not impossible, to understand as human). However, it is not unusual to ends up having structures emerging from that type of algorithm being extremely close to intuitive decomposition that human could think about. Indeed, the consequence of such factorisation is that close users in terms of preferences as well as close items in terms of characteristics ends up having close representations in the latent space.

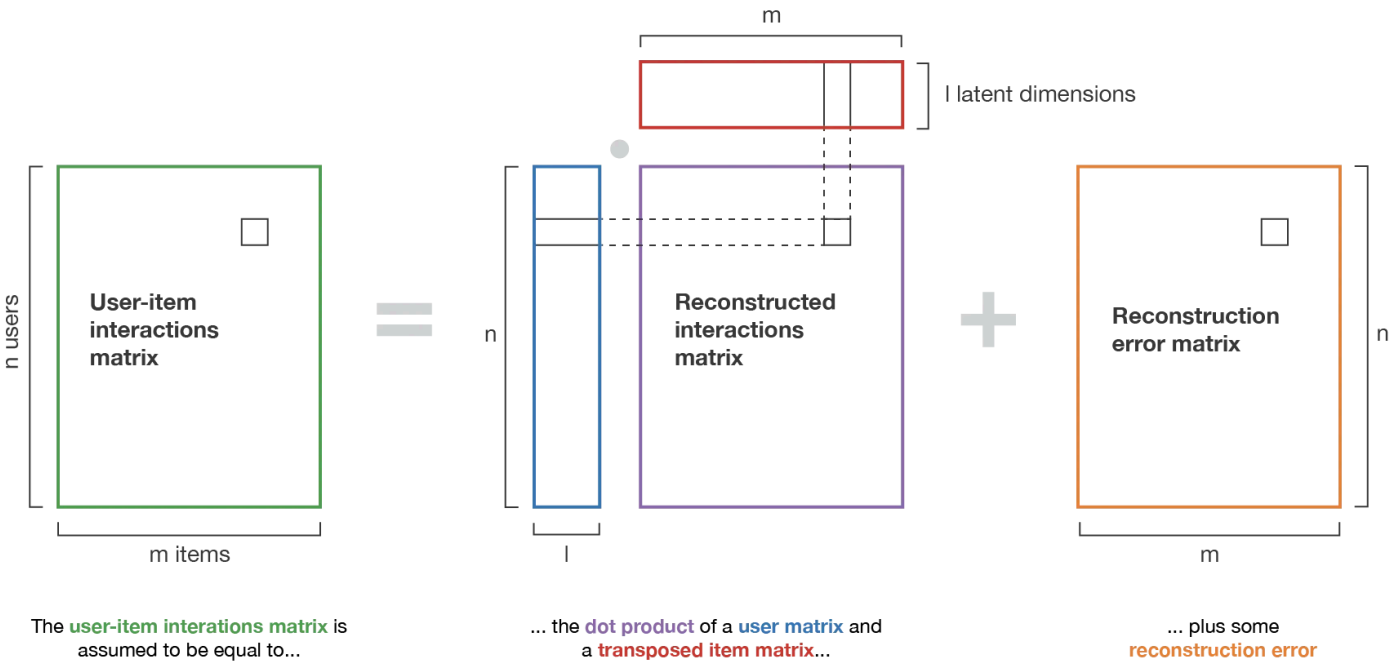


Illustration of the matrix factorization method.

Mathematics of matrix factorisation

In this subsection, we will give a simple mathematical overview of matrix factorization. More especially, we describe a classical iterative approach based on gradient descent that makes possible to obtain factorisations for very large matrices without loading all the data at the same time in computer's memory.

Let's consider an interaction matrix M (nxm) of ratings where only some items have been rated by each user (most of the interactions are set to None to express the lack of rating). We want to factorise that matrix such that

$$M \approx X.Y^T$$

where X is the “user matrix” (nxl) whose rows represent the n users and where Y is the “item matrix” (mxl) whose rows represent the m items:

$$user_i \equiv X_i \qquad \forall i \in \{1, ..., n\}$$
$$item_j \equiv Y_j \qquad \forall j \in \{1, ..., m\}$$

Here l is the dimension of the latent space in which users and item will be represented. So, we search for matrices X and Y whose dot product best approximate the existing interactions. Denoting E the ensemble of pairs (i,j) such that M_ij is set (not None), we want to find X and Y that minimise the “rating reconstruction error”

$$(X, Y) = \underset{X,Y}{argmin} \sum_{(i,j) \in E} [(X_i)(Y_j)^T - M_{ij}]^2$$

Adding a regularisation factor and dividing by 2, we get

$$(X, Y) = \underset{X,Y}{argmin} \frac{1}{2} \sum_{(i,j) \in E} [(X_i)(Y_j)^T - M_{ij}]^2 + \frac{\lambda}{2}(\sum_{i,k}(X_{ik})^2 + \sum_{j,k}(Y_{jk})^2)$$

The matrices X and Y can then be obtained following a gradient descent optimisation process for which we can notice two things. First, the gradient do not have to be computed over all the pairs in E at each step and we can consider only a subset of these pairs so that we optimise our objective function “by batch”. Second, values in X and Y do not have to be updated simultaneously and the gradient descent can be done alternatively on X and Y at each step (doing so, we consider one matrix fixed and optimise for the other before doing the opposite at the next iteration).

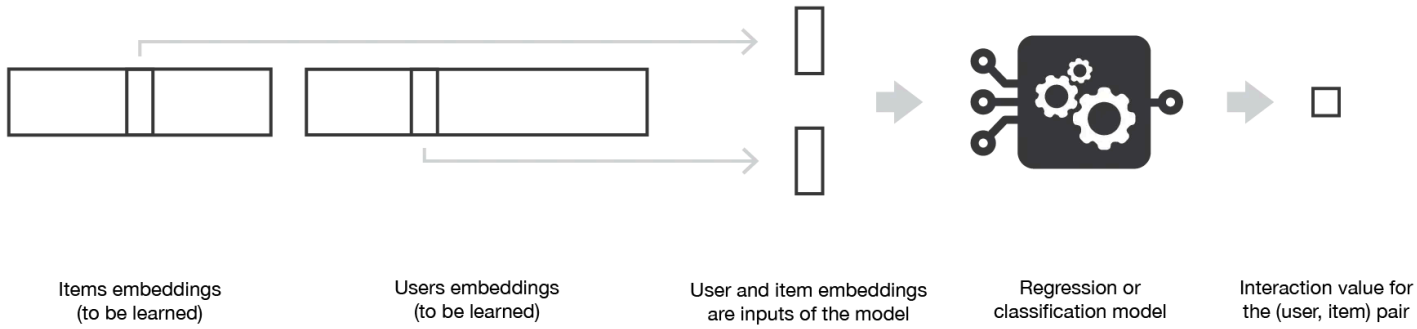
Once the matrix has been factorised, we have less information to manipulate in order to make a new recommendation: we can simply multiply a user vector by any item vector in order to estimate the corresponding rating. Notice that we could also use user-user and item-item methods with these new representations of users and items: (approximate) nearest neighbours searches wouldn't be done over huge sparse vectors but over small dense ones making some approximation techniques more tractable.

Extensions of matrix factorisation

We can finally notice that the concept of this basic factorisation can be extended to more complex models with, for example, more general neural network like "decomposition" (we cannot strictly speak about "factorisation" anymore). The first direct adaptation we can think of concerns boolean interactions. If we want to reconstruct boolean interactions, a simple dot product is not well adapted. If, however, we add a logistic function on top of that dot product, we get a model that takes its value in [0, 1] and, so, better fit the problem. In such case, the model to optimise is

$$(X, Y) = \underset{X,Y}{argmin} \frac{1}{2} \sum_{(i,j) \in E} [f((X_i)(Y_j)^T) - M_{ij}]^2 + \frac{\lambda}{2} (\sum_{i,k} (X_{ik})^2 + \sum_{j,k} (Y_{jk})^2)$$

with f(.) our logistic function. Deeper neural network models are often used to achieve near state of the art performances in complex recommender systems.



Matrix factorization can be generalized with the use of a model on top of users and items embeddings.

Content based approaches

In the previous two sections we mainly discussed user-user, item-item and matrix factorisation approaches. These methods only consider the user-item interaction matrix and, so, belong to the collaborative filtering paradigm. Let's now describe the content based paradigm.

Concept of content-based methods

In content based methods, the recommendation problem is casted into either a classification problem (predict if a user "likes" or not an item) or into a regression problem (predict the rating given by a user to an item). In both cases, we are going to set a model that will be based on the user and/or item features at our disposal (the "content" of our "content-based" method).

If our classification (or regression) is based on users features, we say the approach is item-centred: modelling, optimisations and computations can be done "by item". In this case, we build and learn one model by item based

on users features trying to answer the question “what is the probability for each user to like this item?” (or “what is the rate given by each user to this item?”, for regression). The model associated to each item is naturally trained on data related to this item and it leads, in general, to pretty robust models as a lot of users have interacted with the item. However, the interactions considered to learn the model come from every users and even if these users have similar characteristic (features) their preferences can be different. This mean that even if this method is more robust, it can be considered as being less personalised (more biased) than the user-centred method thereafter.

If we are working with items features, the method is then user-centred: modelling, optimisations and computations can be done “by user”. We then train one model by user based on items features that tries to answer the question “what is the probability for this user to like each item?” (or “what is the rate given by this user to each item?”, for regression). We can then attach a model to each user that is trained on its data: the model obtained is, so, more personalised than its item-centred counterpart as it only takes into account interactions from the considered user. However, most of the time a user has interacted with relatively few items and, so, the model we obtain is a far less robust than an item-centred one.

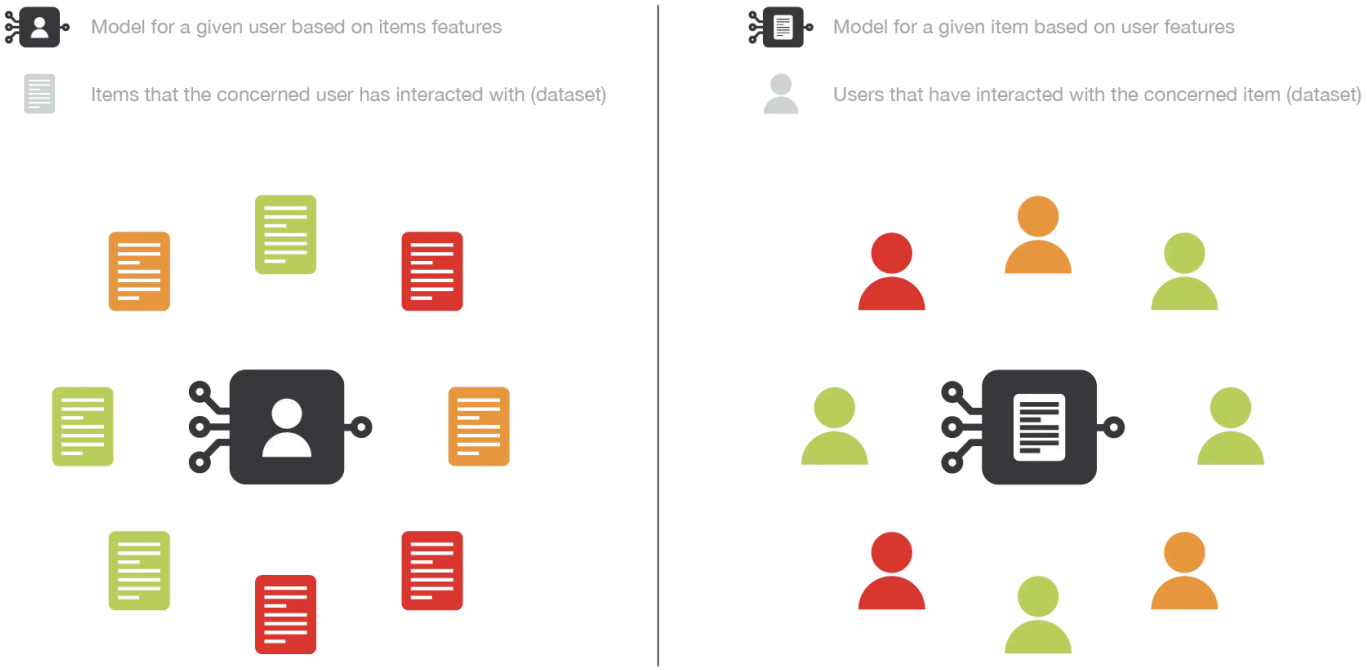


Illustration of the difference between item-centred and user-centred content based methods.

From a practical point of view, we should underline that, most of the time, it is much more difficult to ask some information to a new user (users do not want to answer too much questions) than to ask lot of information about a new item (people adding them have an interest in filling these information in order to make their items recommended to the right users). We can also notice that, depending on the complexity of the relation to express, the model we build can be more or less complex, ranging from basic models (logistic/linear regression for classification/regression) to deep neural networks. Finally, let's mention that content based methods can also be neither user nor item centred: both informations about user and item can be used for our models, for example by stacking the two features vectors and making them go through a neural network architecture.

Item-centred Bayesian classifier

Let's first consider the case of an item-centred classification: for each item we want to train a Bayesian classifier that takes user features as inputs and output either "like" or "dislike". So, to achieve the classification task, we want to compute

$$\frac{\mathbb{P}_{item}(like|user\_features)}{\mathbb{P}_{item}(dislike|user\_features)}$$

the ratio between the probability for a user with given features to like the considered item and its probability to dislike it. This ratio of conditional probabilities that defines our classification rule (with a simple threshold) can be expressed following the Bayes formula

$$\mathbb{P}_{item}(like|user\_features) = \frac{\mathbb{P}_{item}(user\_features|like) \times \mathbb{P}_{item}(like)}{\mathbb{P}_{item}(user\_features)}$$

$$\mathbb{P}_{item}(dislike|user\_features) = \frac{\mathbb{P}_{item}(user\_features|dislike) \times \mathbb{P}_{item}(dislike)}{\mathbb{P}_{item}(user\_features)}$$

$$\frac{\mathbb{P}_{item}(like|user\_features)}{\mathbb{P}_{item}(dislike|user\_features)} = \frac{\mathbb{P}_{item}(user\_features|like) \times \mathbb{P}_{item}(like)}{\mathbb{P}_{item}(user\_features|dislike) \times \mathbb{P}_{item}(dislike)}$$

where

$$\mathbb{P}_{item}(like) \qquad \text{and} \qquad \mathbb{P}_{item}(dislike)(= 1 - \mathbb{P}_{item}(like))$$

are priors computed from the data whereas

$$\mathbb{P}_{item}(.|like) \qquad \text{and} \qquad \mathbb{P}_{item}(.|dislike)$$

are likelihoods assumed to follow Gaussian distributions with parameters to be determined also from data. Various hypothesis can be done about the covariance matrices of these two likelihood distributions (no assumption, equality of matrices, equality of matrices and features independence) leading to various well known models (quadratic discriminant analysis, linear discriminant analysis, naive Bayes classifier). We can underline once more that, here, likelihood parameters have to be estimated only based on data (interactions) related to the considered item.

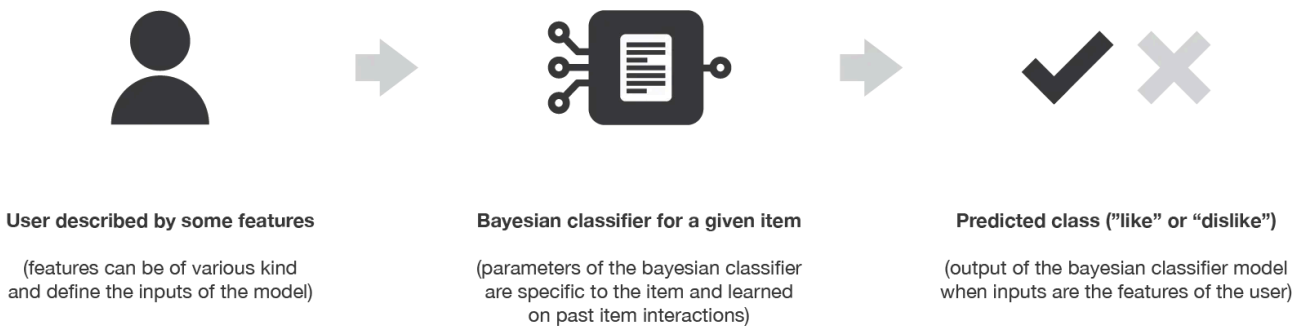


Illustration of the item-centred content based Bayesian classifier.

User-centred linear regression

Let’s now consider the case of a user-centred regression: for each user we want to train a simple linear regression that takes item features as inputs and output the rating for this item. We still denote M the user-item interaction matrix, we stack into a matrix X row vectors representing users coefficients to be learned and we stack into a matrix Y row vectors representing items features that are given. Then, for a given user i, we learn the coefficients in X_i by solving the following optimisation problem

$$X_i = \underset{X_i}{argmin} \frac{1}{2} \sum_{(i,j) \in E} [(X_i)(Y_j)^T - M_{ij}]^2 + \frac{\lambda}{2} (\sum_k (X_{ik})^2)$$

where one should keep in mind that i is fixed and, so, the first summation is only over (user, item) pairs that concern user i. We can observe that if we solve this problem for all the users at the same time, the optimisation problem is exactly the same as the one we solve in “alternated matrix factorisation” when we keep items fixed. This observation underlines the link we mentioned in the first section: model based collaborative filtering approaches (such as matrix factorisation) and content based methods both assume a latent model for user-item interactions but model based collaborative approaches have to learn latent representations for both users and items whereas content-based methods build a model upon human defined features for users and/or items.

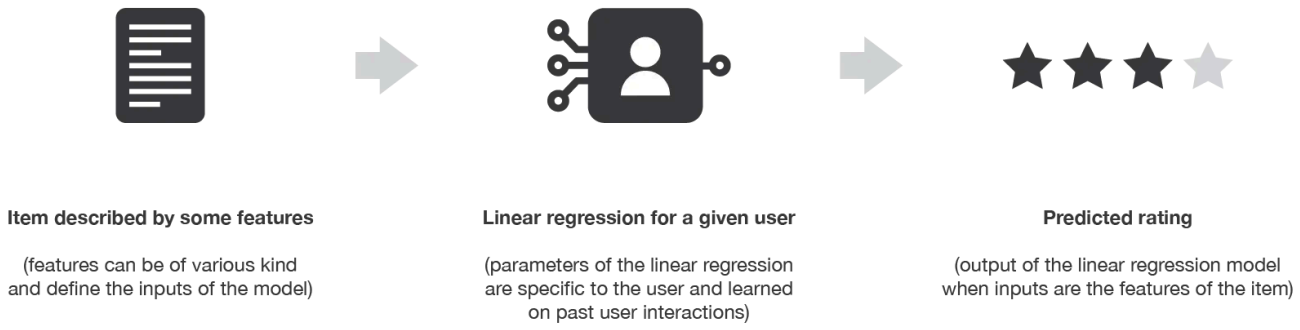


Illustration of the user-centred content based regression.

Evaluation of a recommender system

As for any machine learning algorithm, we need to be able to evaluate the performances of our recommender systems in order to decide which algorithm fit the best our situation. Evaluation methods for recommender

systems can mainly be divided in two sets: evaluation based on well defined metrics and evaluation mainly based on human judgment and satisfaction estimation.

Metrics based evaluation

If our recommender system is based on a model that outputs numeric values such as ratings predictions or matching probabilities, we can assess the quality of these outputs in a very classical manner using an error measurement metric such as, for example, mean square error (MSE). In this case, the model is trained only on a part of the available interactions and is tested on the remaining ones.

Still if our recommender system is based on a model that predicts numeric values, we can also binarize these values with a classical thresholding approach (values above the threshold are positive and values bellow are negative) and evaluate the model in a more “classification way”. Indeed, as the dataset of user-item past interactions is also binary (or can be binarized by thresholding), we can then evaluate the accuracy (as well as the precision and the recall) of the binarized outputs of the model on a test dataset of interactions not used for training.

Finally, if we now consider a recommender system not based on numeric values and that only returns a list of recommendations (such as user-user or item-item that are based on a knn approach), we can still define a precision like metric by estimating the proportion of recommended items that really suit our user. To estimate this precision, we can not take into account recommended items that our user has not interacted with and we should only consider items from the test dataset for which we have a user feedback.

Human based evaluation

When designing a recommender system, we can be interested not only to obtain model that produce recommendations we are very sure about but we can also expect some other good properties such as diversity and explainability of recommendations.

As mentioned in the collaborative section, we absolutely want to avoid having a user being stuck in what we called earlier an information confinement area. The notion of "serendipity" is often used to express the tendency a model has or not to create such a confinement area (diversity of recommendations). Serendipity, that can be estimated by computing the distance between recommended items, should not be too low as it would create confinement areas, but should also not be too high as it would mean that we do not take enough into account our users interests when making recommendations (exploration vs exploitation). Thus, in order to bring diversity in the suggested choices, we want to recommend items that both suit our user very well and that are not too similar from each others. For example, instead of recommending a user "Start Wars" 1, 2 and 3, it seems better to recommend "Star wars 1", "Start trek into darkness" and "Indiana Jones and the raiders of the lost ark": the two later may be seen by our system as having less chance to interest our user but recommending 3 items that look too similar is not a good option.

Explainability is another key point of the success of recommendation algorithms. Indeed, it has been proven that if users do not understand why they had been recommended as specific item, they tend to loose confidence into the recommender system. So, if we design a model that is clearly explainable, we can add, when making recommendations, a little sentence stating why an item has been recommended ("people who liked this item also liked this one", "you liked this item, you may be interested by this one", ...).

Finally, on top of the fact that diversity and explainability can be intrinsically difficult to evaluate, we can notice that it is also pretty difficult to assess the quality of a recommendation that do not belong to the testing dataset: how to know if a new recommendation is relevant before actually recommending it to our user? For all these reasons, it can sometimes be tempting to test the model in "real conditions". As the goal of the recommender system is to generate an action (watch a movie, buy a product, read an article etc...), we can indeed evaluate its ability to generate the expected action. For example, the system can be put in production, following an A/B testing approach, or can be tested only on a sample of users. Such processes require, however, having a certain level of confidence in the model.

Takeaways

The main takeaways of this article are:

- recommendation algorithms can be divided in two great paradigms: collaborative approaches (such as user-user, item-item and matrix factorisation) that are only based on user-item interaction matrix and content based approaches (such as regression or classification models) that use prior information about users and/or items
- memory based collaborative methods do not assume any latent model and have then low bias but high variance ; model based collaborative approaches assume a latent interactions model that needs to learn both users and items representations from scratch and have, so, a higher bias but a lower variance ; content based methods assume a latent model build around users and/or items features explicitly given and have, thus, the highest bias and lowest variance

- recommender systems are more and more important in many big industries and some scales considerations have to be taken into account when designing the system (better use of sparsity, iterative methods for factorisation or optimisation, approximate techniques for nearest neighbours search...)
- recommender systems are difficult to evaluate: if some classical metrics such that MSE, accuracy, recall or precision can be used, one should keep in mind that some desired properties such as diversity (serendipity) and explainability can't be assessed this way ; real conditions evaluation (like A/B testing or sample testing) is finally the only real way to evaluate a new recommender system but requires a certain confidence in the model

We should notice that we have not discussed hybrid approaches in this introductory post. These methods, that combine collaborative filtering and content based approaches, achieves state-of-the-art results in many cases and are, so, used in many large scale recommender systems nowadays. The combination made in hybrid approaches can mainly take two forms: we can either train two models independently (one collaborative filtering model and one content based model) and combine their suggestions or directly build a single model (often a neural network) that unify both approaches by using as inputs prior information (about user and/or item) as well as "collaborative" interactions information.

As we mentioned in the introduction of this post, recommender systems are becoming essential in many industries and, hence, have received always more attention in the recent years. In this article, we have introduced basics notions required for a better understanding of the questions related to these

systems, but we highly encourage interested readers to explore this field further... without giving, ironically, any specific reading recommendations!

Thanks for reading!

Last articles written with Joseph Rocca:

Ensemble methods: bagging, boosting and stacking

Understanding the key concepts of ensemble learning.

towardsdatascience.com

A brief introduction to Markov chains

Definitions, properties and PageRank example.

towardsdatascience.com

Machine Learning | Artificial Intelligence | Deep Learning | Data Science | Towards Data Science

Published in Towards Data Science

767K Followers · Last published 4 hours ago

Follow

Your home for data science. A publication sharing concepts, ideas and codes.

Written by Baptiste Rocca

1.6K Followers · 3 Following

Follow

Data scientist at ManoMano

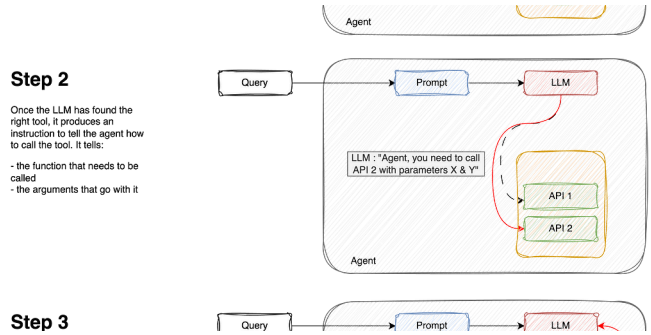
More from Baptiste Rocca and Towards Data Science

In Towards Data Science by Baptiste Rocca

Handling imbalanced datasets in machine learning

What should and should not be done when facing an imbalanced classes problem?

Jan 27, 2019 6K 36

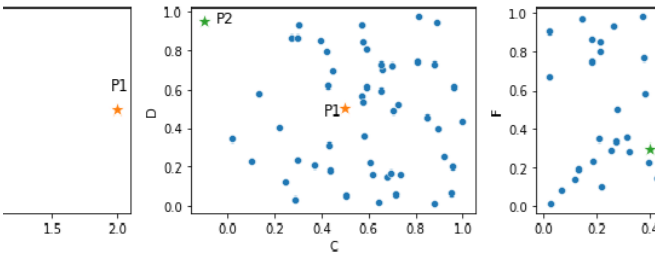


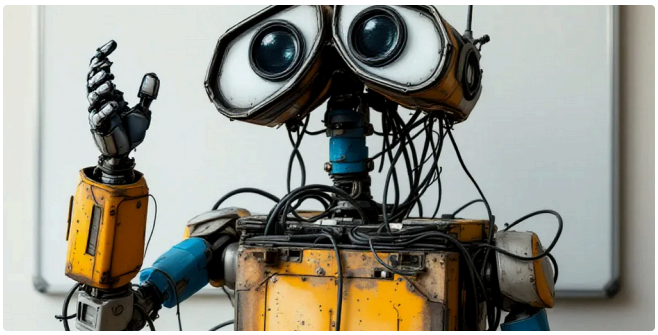
In Towards Data Science by Benjamin Etienne

Build your Personal Assistant with Agents and Tools

Learn how to build your personal assistant using LangChain agents and Gemini by...

1d ago 125





In Towards Data Science by W Brett Kennedy

Perform Outlier Detection More Effectively Using Subsets of...

Identify relevant subspaces: subsets of features that allow you to most effectively...

20h ago 379 9

In Towards Data Science by Heiko Hotz

Productionising GenAI Agents: Evaluating Tool Selection with...

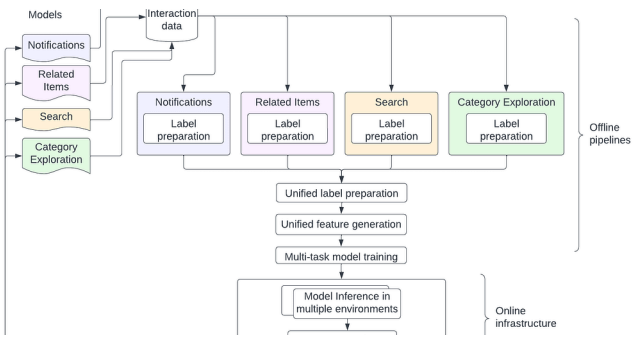
How to create reliable and scalable GenAI Agents for real-world applications

2d ago 190 2

See all from Baptiste Rocca

See all from Towards Data Science

Recommended from Medium



Netflix Technology Blog

Lessons Learnt From Consolidating ML Models in a Large Scale...

by Roger Menezes, Rahul Jha, Gary Yeh, and Sudarshan Lamkhede

Aug 24, 2023 718 4

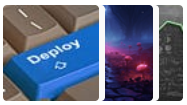
In Towards Data Science by Dr. Robert Kübler

Introduction to Embedding-Based Recommender Systems

Learn to build a simple matrix factorization recommender in TensorFlow

Jan 25, 2023 654 5

Lists



Predictive Modeling w/ Python

20 stories · 1685 saves

Natural Language Processing

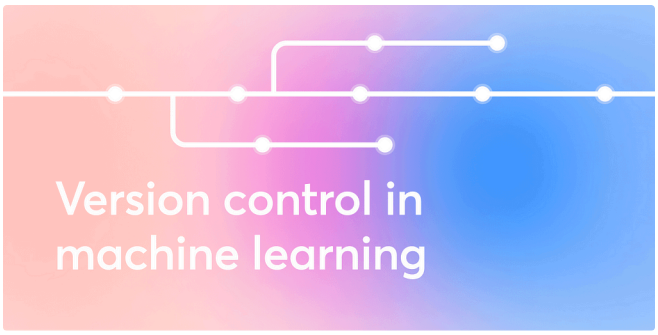
1830 stories · 1450 saves

Practical Guides to Machine Learning

10 stories · 2047 saves

data science and AI

40 stories · 288 saves

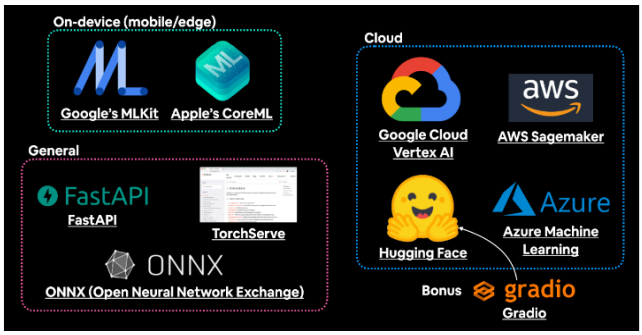


Salman Anwaar

Effective Model Version Management in Machine Learnin...

In machine learning (ML) projects, one of the most critical components is version...

Sep 13 1K



Ling Huang

MLOps—ML/DL Model Deployment

Model deployment is the process of making machine learning model accessible to...

Jul 17

In Stackademic by Abdur Rahman

Python is No More The King of Data Science

5 Reasons Why Python is Losing Its Crown

Oct 22 8.2K 32

ZIRU

Mastering Machine Learning System Design Interviews

With the rise of machine learning (ML) in industries across the globe, companies are o...

Aug 22 2

See more recommendations