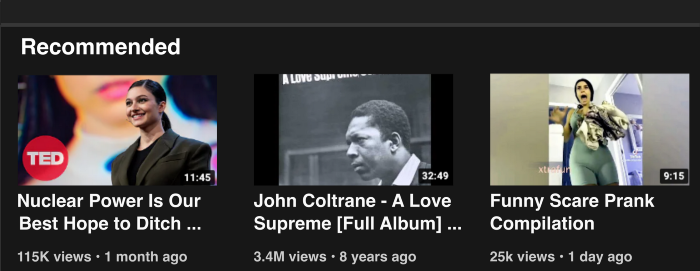
**Recommender Systems — A Complete Guide to Machine Learning Models**

**Leveraging data to help users discovering new content**

**Recommender Systems: Why And How?**

[**Recommender systems**](https://en.wikipedia.org/wiki/Recommender_system) are algorithms providing personalized suggestions for items that are most relevant to each user. With the massive growth of available online contents, users have been inundated with choices. It is therefore crucial for web platforms to offer recommendations of items to each user, in order to increase user satisfaction and engagement.



YouTube recommends videos to users, to help them discover and watch content relevant to them in the middle of a huge number of available contents. (Image by Author)

The following list shows examples of well-known web platforms with a **huge number of available contents**, which need efficient recommender systems to keep users interested.

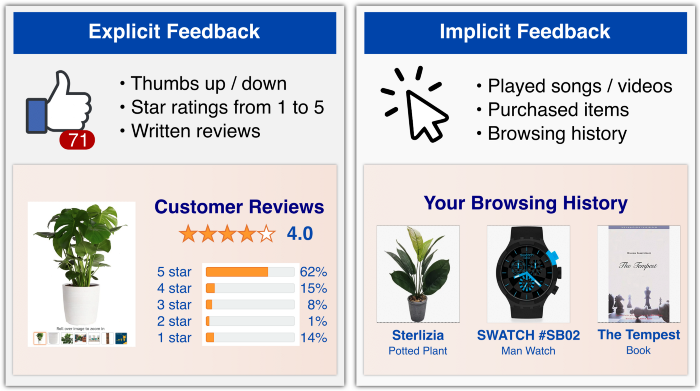
1. [**Youtube**](https://www.youtube.com/)**.** Every minute people upload [500 hours of videos](https://www.oberlo.com/blog/youtube-statistics), i.e. it would take 82 years to a user to watch all videos uploaded just in the last hour.
2. [**Spotify**](https://www.spotify.com/)**.** Users can listen to ore than [80 million song tracks and podcasts](https://newsroom.spotify.com/company-info/).
3. [**Amazon**](https://www.amazon.com/)**.** Users can buy more than [350 million different products](https://www.retailtouchpoints.com/resources/how-many-products-does-amazon-carry).

All these platforms use powerful machine learning models in order to generate relevant recommendations for each user.

**Explicit Feedback vs. Implicit Feedback**

In recommender systems, machine learning models are used to predict the **rating *rᵤᵢ* of a user *u* on an item *i****.* At inference time, we recommend to each user *u* the items *l* having highest predicted rating *rᵤ****ᵢ***.

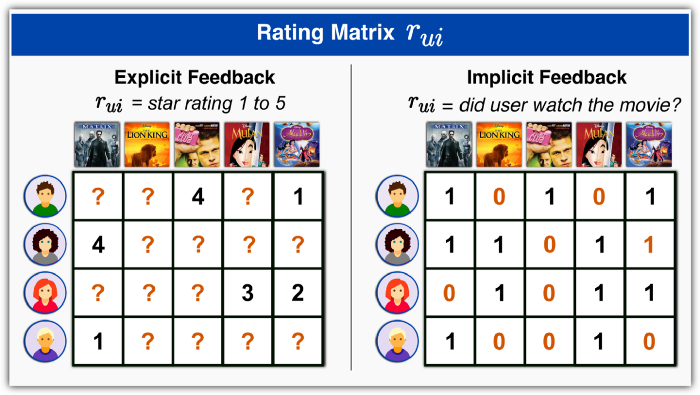
We therefore need to collect user feedback, so that we can have a ground truth for training and evaluating our models. An important distinction has to be made here between **explicit feedback** and **implicit feedback**.



Explicit vs. implicit feedback for recommender systems. (Image by Author)

[**Explicit feedback**](https://en.wikipedia.org/wiki/Rating_scale#Rating_scales_used_online)is a rating explicitly given by the user to express their satisfaction with an item. Examples are: number of stars on a scale from 1 to 5 given after buying a product, thumb up/down given after watching a video, etc. This feedback provides **detailed information** on how much a user liked an item, but it is **hard to collect** as most users typically don’t write reviews or give explicit ratings for each item they purchase.

[**Implicit feedback**](https://en.wikipedia.org/wiki/Implicit_data_collection), on the other hand, assume that user-item interactions are an indication of preferences. Examples are: purchases/browsing history of a user, list of songs played by a user, etc. This feedback is **extremely abundant**, but at the same time it is **less detailed** and **more noisy** (e.g. someone may buy a product as a present for someone else). However, this noise becomes negligible when compared to the sheer size of available data of this kind, and **most modern Recommender Systems tend to rely on implicit feedback**.

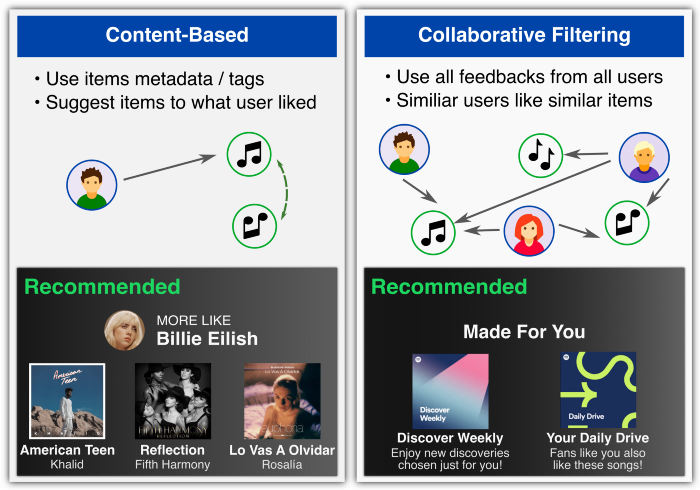


User-item rating matrix for explicit feedback and implicit feedback datasets. (Image by Author)

Once we have collected explicit or implicit feedbacks, we can create the **user-item rating matrix *rᵤᵢ***. For explicit feedback, each entry in *rᵤᵢ* is a numerical value—e.g. *rᵤᵢ =* “stars given by *u* to movie *i*”—or “?” if user *u* did not rate item *i*. For implicit feedback, the values in *rᵤᵢ* are a boolean values representing presence or lack of interaction—e.g. *rᵤᵢ =* “did user *u* watch movie *i*?”. Notice that the matrix *rᵤᵢ* is very sparse, as users interact with few items among all available contents, and they review even fewer items!

**Content-Based vs. Collaborative Filtering Approaches**

Recommender system can be classified according to the kind of information used to predict user preferences as **Content-Based** or **Collaborative Filtering.**



Content-Based vs. Collaborative Filtering approaches for recommender systems. (Image by author)

**Content-Based Approach**

[**Content-based methods**](https://en.wikipedia.org/wiki/Recommender_system#Content-based_filtering) describe **users and itemsby** **their known metadata**. Each item *i* is represented by a set of relevant tags—e.g. movies of the [IMDb platform](https://www.imdb.com/search/keyword/) can be tagged as*“action”, “comedy”,* etc. Each user *u* is represented by a user profile, which can created from known user information—e.g. sex and age—or from the user’s past activity.

To train a Machine Learning model with this approach we can use a [**k-NN model**](https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm)**.** For instance, if we know that user *u* bought an item *i*, we can recommend to *u* the available items with features most similar to *i*.

The **advantage** of this approach is that items metadata are known in advance, so we can also apply it to [Cold-Start scenarios](https://en.wikipedia.org/wiki/Cold_start_(recommender_systems)) where a new item or user is added to the platform and we don’t have user-item interactions to train our model. The **disadvantages** are that we don’t use the full set of known user-item interactions (each user is treated independently), and that we need to know metadata information for each item and user.

**Collaborative Filtering Approach**

[**Collaborative filtering methods**](https://en.wikipedia.org/wiki/Recommender_system#Collaborative_filtering)do not use item or user metadata, but try instead to **leverage the feedbacks or activity history of all users** in order to predict the rating of a user on a given item by inferring interdependencies between users and items from the observed activities.

To train a Machine Learning model with this approach we typically try to cluster or factorize the rating matrix *rᵤᵢ* in order to make predictions on the unobserved pairs (*u,i*), i.e. where *rᵤᵢ* =“?”. In the following on this article we present the [**Matrix Factorization algorithm**](https://en.wikipedia.org/wiki/Matrix_factorization_(recommender_systems)), which is the most popular method of this class.

The **advantage** of this approach is that the whole set of user-item interactions (i.e. the matrix *rᵤᵢ*) is used, which typically allows to obtain higher accuracy than using Content-Based models. The **disadvantage** of this approach is that it requires to have a few user interactions before the model can be fitted.

**Hybrid Approaches**

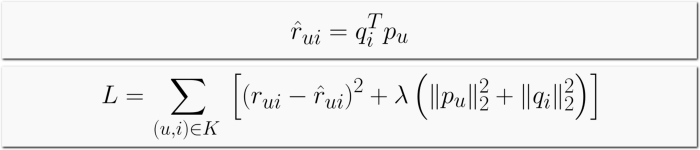
Finally, there are also [**hybrid methods**](https://en.wikipedia.org/wiki/Recommender_system#Hybrid_recommendations_approaches) that try to use both the known metadata and the set of observed user-item interactions. This approach **combines advantages** of both Content-Based and Collaborative Filtering methods, and allow to obtain the best results. Later in this article we present [**LightFM**](https://github.com/lyst/lightfm), which is the most popular algorithm of this class of methods.

**Collaborative Filtering: Matrix Factorization**

[**Matrix factorization algorithms**](https://datajobs.com/data-science-repo/Recommender-Systems-%5bNetflix%5d.pdf) are probably the most popular and effective collaborative filtering methods for recommender systems. Matrix factorization is a [latent factor model](https://en.wikipedia.org/wiki/Latent_variable_model) assuming that for each user *u* and item *i* there are **latent vector representations *pᵤ, qᵢ*** ∈ **R**ᶠ s.t. *rᵤᵢ* can be uniquely expressed— i.e. “factorized” — in terms of *pᵤ* and *qᵢ*. The Python library [**Surprise**](https://github.com/NicolasHug/Surprise)provides excellent implements of these methods.

**Matrix Factorization for Explicit Feedback**

The simplest idea is to model user-item interactions through a **linear model**. To learn the values of *pᵤ* and *qᵢ*, we can minimize a regularized [MSE loss](https://en.wikipedia.org/wiki/Mean_squared_error) over the set *K* of pairs (*u*, *i*) for which *rᵤᵢ* is known. The algorithm so obtained is called [**probabilistic matrix factorization (PMF)**](https://proceedings.neurips.cc/paper/2007/hash/d7322ed717dedf1eb4e6e52a37ea7bcd-Abstract.html).



Probabilistic matrix factorization: model for rᵤᵢ *and loss function.*

The loss function can be minimized in two different ways. The first approach is to use [**stochastic gradient descent (SGD)**](https://en.wikipedia.org/wiki/Stochastic_gradient_descent)**.** SGD is easy to implement, but it may have some issues because both *pᵤ* and *qᵢ* are both unknown and therefore the loss function is not convex. To solve this issue, we can alternatively fix the value *pᵤ* and *qᵢ* and obtain a convex linear regression problem that can be easily solved with [ordinary least squares (OLS)](https://en.wikipedia.org/wiki/Ordinary_least_squares). This second method is known as [**alternating least squares (ALS)**](https://spark.apache.org/docs/2.2.0/ml-collaborative-filtering.html) and allows significant parallelization and speedup.

The PMF algorithm was later generalized by the [**singular value decomposition (SVD)**](https://sifter.org/simon/journal/20061211.html) algorithm, which introduced **bias terms** in the model. More specifically, ***bᵤ*** and ***bᵢ*** measure observed rating deviations of user *u* and item *i*, respectively, while ***μ***is the overall average rating. These terms often explain most of the observed ratings *rᵤᵢ*, as some items widely receive better/worse ratings, and some users are consistently more/less generous with their ratings.

https://miro.medium.com/max/700/1*dopKPhNNW7PD63ci3My9_w.png

SVD algorithm, a generalization of probabilistic matrix factorization.

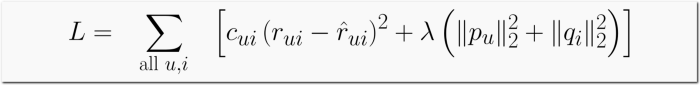
**Matrix Factorization for Implicit Feedback**

The **SVD** **method** **can** **be** [**adapted to implicit feedback datasets**](https://ieeexplore.ieee.org/document/4781121). The idea is to look at implicit feedback as an **indirect measure of confidence**. Let’s assume that the implicit feedback ***tᵤᵢ*** measures the percentage of movie *i* that user *u* has watched — e.g. *tᵤᵢ* = 0 means that *u* never watched *i*, *tᵤᵢ* = 0.1 means that he watched only 10% of it, *tᵤᵢ* = 2 means that he watched it twice. Intuitively, a user is more likely to be interested into a movie they watched twice, rather than in a movie they never watched. We therefore define a **confidence matrix *cᵤᵢ*** and a **rating matrix *rᵤᵢ***as follows.



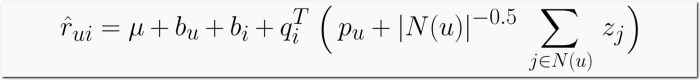
Confidence matrix and rating matrix for implicit feedback.

Then, we can model the observed *rᵤᵢ* using the same linear model used for SVD, but with a slightly different loss function. First, we compute the loss over all (*u*, *i*) pairs — unlike the explicit case, if user *u* never interacted with *i* we have *rᵤᵢ =* 0 instead of *rᵤᵢ =*“?” . Second, we weight each loss term by the confidence *cᵤᵢ* that *u* likes *i.*



Loss function for SVD for implicit feedback.

Finally, the [**SVD++**](https://dl.acm.org/doi/abs/10.1145/1401890.1401944)algorithm can be used when we have access to both explicit and implicit feedbacks. This can be very useful, because typically users interact with many items (= implicit feedabck) but rate only a small subset of them (= explicit feedback). Let’s denote, for each user *u*, the set *N(u)* of items that *u* has interacted with. Then, we assume that an implicit interaction with an item *j* is associated with a new latent vector *zⱼ* ∈ **R***ᶠ*. The SVD++ algorithm modifies the linear model of SVD by including into the user representation a weighted sum of these latent factors *zⱼ.*



SVD++ for mixed (explicit + implicit) feedback

**Hybrid Approach: LightFM**

Collaborative filtering methods based on matrix factorization often produce excellent results, but in [**cold-start scenarios**](https://en.wikipedia.org/wiki/Cold_start_(recommender_systems))—where little to no interaction data is available for new items and users—they cannot make good predictions because they lack data to estimate the latent factors. **Hybrid approaches** solve this issue by leveraging known item or user metadata in order to improve the matrix factorization model. The Python library [**LightFM**](https://github.com/lyst/lightfm) implements one of the most popular hybrid algorithms.

In LightFM, we assume that for each user *u* we have collected a set of tag annotations *Aᵁ(u)* — e.g. *“male”*, *“age < 30”*, … — and similarly each item *i* has a set of annotations *Aᴵ(i)* — e.g. *“price > 100 $”*, *“book”*, … Then we model each user tag by a **latent factor *xᵁₐ* ∈** **R**ᶠ and by a **bias term *bᵁₐ* ∈** **R**, and we assume that the user vector representation *pᵤ* and its associated bias *bᵤ* can be expressed simply as the sum of these terms *xᵁₐ* and *bᵁₐ*, respectively. We take the same approach to item tags, using latent factors *xᴵₐ* ∈ Rᶠ and bias terms *bᴵₐ* ∈ R. Once we have defined *pᵤ, qᵢ, bᵤ, bᵢ* using these formulas, we can use the same linear model of SVD to describe the relationship between these terms and *rᵤᵢ*.



LightFM: user/item embeddings and biases are the sum of latent vectors associated to each user/item.

Notice that there are three interesting cases of this hybrid approach of LightFM.

1. **Cold start.** If we have a new item *i* with known tags *Aᴵ(i)*, then we can use the latent vectors *xᴵₐ* (obtained by fitted our model on the previous data) to compute its embedding *qᵢ*, and therefore estimate for any user *u* its rating *rᵤᵢ*.
2. **No available tags.** If we don’t have any known metadata for items or users, the only annotation we can use is an indicator function, i.e. a different annotation *a* for each user and each item. Then, user and item feature matrices are identity matrices, and LightFM reduces to a classical collaborative filtering method such as SVD.
3. **Content-based vs. Hybrid.** If we only used user or item tags without indicator annotations, LightFM would almost be a content-based model. So in practice, to leverage user-item interactions, we also add to known tags an indicator annotation *a* different from each user and item.

**TL;DR – Conclusions**

* **Recommender systems** leverage machine learning algorithms to help users inundated with choices in discovering relevant contents.
* **Explicit vs. implicit feedback**: the first is easier to leverage, but the second is way more abundant.
* **Content-based** models work well in cold-start scenarios, but require to know user and item **metadata**.
* **Collaborative filtering** models typically use matrix factorization: **PMF, SVD, SVD for implicit feedback, SVD++.**
* **Hybrid models** take the best of content-based and collaborative filtering. **LightFM** is a great example of this approach.

**References**

* [Wikipedia, *Recommender System*](https://en.wikipedia.org/wiki/Recommender_system)*.*
* [“Surprise”, *Python package documentation*.](https://surprise.readthedocs.io/en/stable/)
* [(S. Funk 2006), *Netflix Update: Try This at Home.*](https://sifter.org/simon/journal/20061211.html)
* [(R. Salakhutdinov 2007), *Probabilistic matrix factorization.*](https://proceedings.neurips.cc/paper/2007/hash/d7322ed717dedf1eb4e6e52a37ea7bcd-Abstract.html)
* ([Y. Hu 2008), *Collaborative Filtering for Implicit Feedback Datasets*](https://ieeexplore.ieee.org/document/4781121)*.*
* [(Y. Koren 2009), *Matrix Factorization Techniques for Recommender Systems*](https://ieeexplore.ieee.org/document/5197422)*.*
* [(Y. Koren 2008) *Factorization meets the neighborhood: a multifaceted collaborative filtering model.*](https://dl.acm.org/doi/abs/10.1145/1401890.1401944)
* ([M. Kula 2015), *Metadata Embeddings for User and Item Cold-start Recommendations*](https://arxiv.org/pdf/1507.08439.pdf)*.*