

Recommender Systems in Practice

How companies make product recommendations



Companies like Amazon, Netflix, LinkedIn, and Pandora leverage recommender systems to help users discover new and relevant items (products, videos, jobs, music), creating a delightful user experience while driving incremental revenue.

Here we provide a practical overview of recommender systems. First, three major systems are reviewed: content-based, collaborative filtering, and hybrid, followed by discussions on cold start, scalability, interpretability, and exploitation/exploration.

Content-based recommendation

At Pandora, a team of musicians labeled each music with more than 400 attributes. Then when a user selects a music station, songs that match the station's attributes will be added to the playlist (Music Genome Project|Pandora, Howe|Pandora).

This is content-based recommendation. Users or items have profiles describing their characteristics and the system would recommend an item to a user if the two profiles match. Stitch Fix's fashion box is another example of content-based recommendation. A user's attributes are collected (height, weight, etc.) and matched fashion products are put in a box delivered to the user (Stitch Fix|2013).

For Pandora, manual efforts/costs are needed to create music attributes, but there are many cases without such a need. Stitch Fix's customers provide their own preferences, LinkedIn users provide their own working experiences and skills, merchants on Amazon provide information about their product items, all are freely usable for content-based recommendations.

A straightforward way of matching users and items is keyword matching. For example, for job recommendations, one can match a job description to job seekers' resumes. Term frequency-inverse document frequency is often used to put more weights to keywords that are unique for an item or user.

A more systematic way is building a supervised model estimating the propensity of a user likes an unseen item. In the model, features are the attributes from users and items (e.g., an indicator variable whether a job and a job seeker is in the same industry), and the response variable is whether the user likes the item (e.g., whether the job seeker would apply for the job).

Content-based methods are computationally fast and interpretable. They can be easily adapted to new items or new users. However, some characteristics of items/users may not be easy to capture or describe explicitly. Stitch Fix addressed this by letting machine learning handle structured data, and human handle unstructured data (e.g., users' Pinterest board).

Collaborative filtering

Collaborative filtering systems make recommendations based on historic users' preference for items (clicked, watched, purchased, liked, rated, etc.). The preference can be presented as a user-item matrix. Here is an example of a matrix describing the preference of 4 users on 5 items, where p_{12} is the user 1's preference on item 2.

$$P = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} & p_{15} \\ p_{21} & p_{22} & p_{23} & p_{24} & p_{25} \\ p_{31} & p_{32} & p_{33} & p_{34} & p_{35} \\ p_{41} & p_{42} & p_{43} & p_{44} & p_{45} \end{bmatrix}$$

Although the entries can be numeric, e.g., Netflix's movie rating prediction challenge (rating ranges from 1 to 5), in most applications, they are binary (e.g., clicked, watched, purchased).

In reality, the user-item matrix can be more than millions * millions (e.g., Amazon, Youtube), and the majority of entries are missing — the goal of recommender systems is to fill those missing entries.

$$\begin{bmatrix} p_{11} & ? & p_{13} & ? & p_{15} \\ ? & ? & ? & p_{24} & ? \\ p_{31} & ? & p_{33} & ? & ? \\ ? & p_{42} & ? & p_{44} & p_{45} \end{bmatrix}$$

Here we describe three collaborative-filtering-related approaches, nearest-neighbor, and two methods creating a new latent space: matrix factorization and deep learning.

Nearest-neighbor

Nearest-neighbor-based methods are based on the similarity between pairs of items or users. Cosine similarity is often used for measuring the distance.

$$\text{similarity}(a, b) = \cos(a, b) = \frac{a \cdot b}{\|a\| * \|b\|}$$

The preference matrix can be represented as item vectors

$$P = [I_1 \quad I_2 \quad I_3 \quad I_4 \quad I_5]$$

The similarity between item $I1$ and item $I2$ is calculated as $\cos(I1, I2)$. The matrix can also be represented as user vectors

$$P = \begin{bmatrix} U_1 \\ U_2 \\ U_3 \\ U_4 \end{bmatrix}$$

The similarity between $U1$ and $U2$ is calculated as $\cos(U1, U2)$. Note, the missing values in the preference matrix are commonly filled with zeros.

For user_i, we can recommend the items liked by user_i's most similar users (user-to-user) or the most similar items of user_i's liked items (item-to-item).

Item-to-item approaches are adapted commonly in practice, by Amazon (Amazon|2003), Youtube (Youtube|2010), LinkedIn (LinkedIn|2014), etc. When a customer likes an item, an item-to-item system can quickly surface items similar to it (similar items for each item are pre-calculated and saved in a key-value data store). In addition, item-to-item recommendations can be more interpretable than user-to-user recommendations, e.g., the systems can explain why an item is recommended because "you liked X".

It is possible that the number of items similar to an item is too small (after applying a threshold on the similarity scores). One could expand the similar item list by including similar items' similar items (Youtube|2010).

After getting the most similar items, a post-processing step can be useful. (Youtube|2010) ranked the similar items according to video quality (e.g., measured by rating), diversity (e.g., limited the recommendations from one channel), and user specificity (e.g., similar videos to a video with more watch time by the user should be ranked higher). The three elements were combined with a linear model, providing a final ranking.

Latent-factor methods


Latent-factor methods create a new and usually reduced feature space of the original user or item vectors, leading to reduced noise and faster computations in real-time.

In the following, we introduce two latent-factor methods — matrix factorization and deep learning.

Matrix factorization

Matrix factorization was popularly used during the Netflix recommendation challenge, especially singular value decomposition and a more practical version for recommender systems.

Singular value decomposition (SVD) decomposes the preference matrix as



$$P_{m \times n} = U_{m \times m} \Sigma_{m \times n} V_{n \times n}$$

U and V are unitary matrices. For 4 users and 5 items, it looks like

$$P_{4 \times 5} = \begin{bmatrix} u_{11} & u_{12} & u_{13} & u_{14} \\ u_{21} & u_{22} & u_{23} & u_{24} \\ u_{31} & u_{32} & u_{33} & u_{34} \\ u_{41} & u_{42} & u_{43} & u_{44} \end{bmatrix} \cdot \begin{bmatrix} \sigma_1 & 0 & 0 & 0 & 0 \\ 0 & \sigma_2 & 0 & 0 & 0 \\ 0 & 0 & \sigma_3 & 0 & 0 \\ 0 & 0 & 0 & \sigma_4 & 0 \end{bmatrix} \cdot \begin{bmatrix} v_{11} & v_{12} & v_{13} & v_{14} & v_{15} \\ v_{21} & v_{22} & v_{23} & v_{24} & v_{25} \\ v_{31} & v_{32} & v_{33} & v_{34} & v_{35} \\ v_{41} & v_{42} & v_{43} & v_{44} & v_{45} \\ v_{51} & v_{52} & v_{53} & v_{54} & v_{55} \end{bmatrix}$$

where $\sigma_1 > \sigma_2 > \sigma_3 > \sigma_4$.

The preference of the first user for the first item can be written as

$$p_{11} = \sigma_1 u_{11} v_{11} + \sigma_2 u_{12} v_{21} + \sigma_3 u_{13} v_{31} + \sigma_4 u_{14} v_{41}$$

This can be presented as vectors

$$p_{11} = \vec{\sigma} \circ \vec{u}_1 \bullet \vec{v}_1$$

An entrywise product is applied between the sigma vector and the first user vector, and then a dot product with the first item vector. It can be seen u and v have the same length, i.e., they are in the same latent feature space. The sigma vector represents the importance of each feature.

Now let's select the top two features based on the sigmas

$$p_{11} \approx \sigma_1 u_{11} v_{11} + \sigma_2 u_{12} v_{21}$$

which can be presented as the item and user vectors each has a length of two.

Simon Funk's SVD

Lots of entries in the preference matrix can be missing and the regular SVD has the following issues (1) how missing values are imputed can have an undesirable impact on the outcome. (2) computational complexity for training can be high with all the entries considered.

During the Netflix challenge, Simon Funk came up with a practical solution (Funk|2006)

$$\min_{\vec{u}_i, \vec{v}_j} \sum_{p_{ij}} (p_{ij} - \vec{u}_i \cdot \vec{v}_j)^2$$

In the formula, only the non-missing entries p_{ij} are considered. The estimated score for the j^{th} item from i^{th} user is

$$\vec{u}_i \cdot \vec{v}_j$$

Note, user and item vectors do not have unit lengths as in SVD, but it doesn't matter as the sum of squares for error is minimized. Funk's approach had great success in the Netflix challenge, and the idea was implemented by Netflix (Netflix|2012).

Deep learning embedding

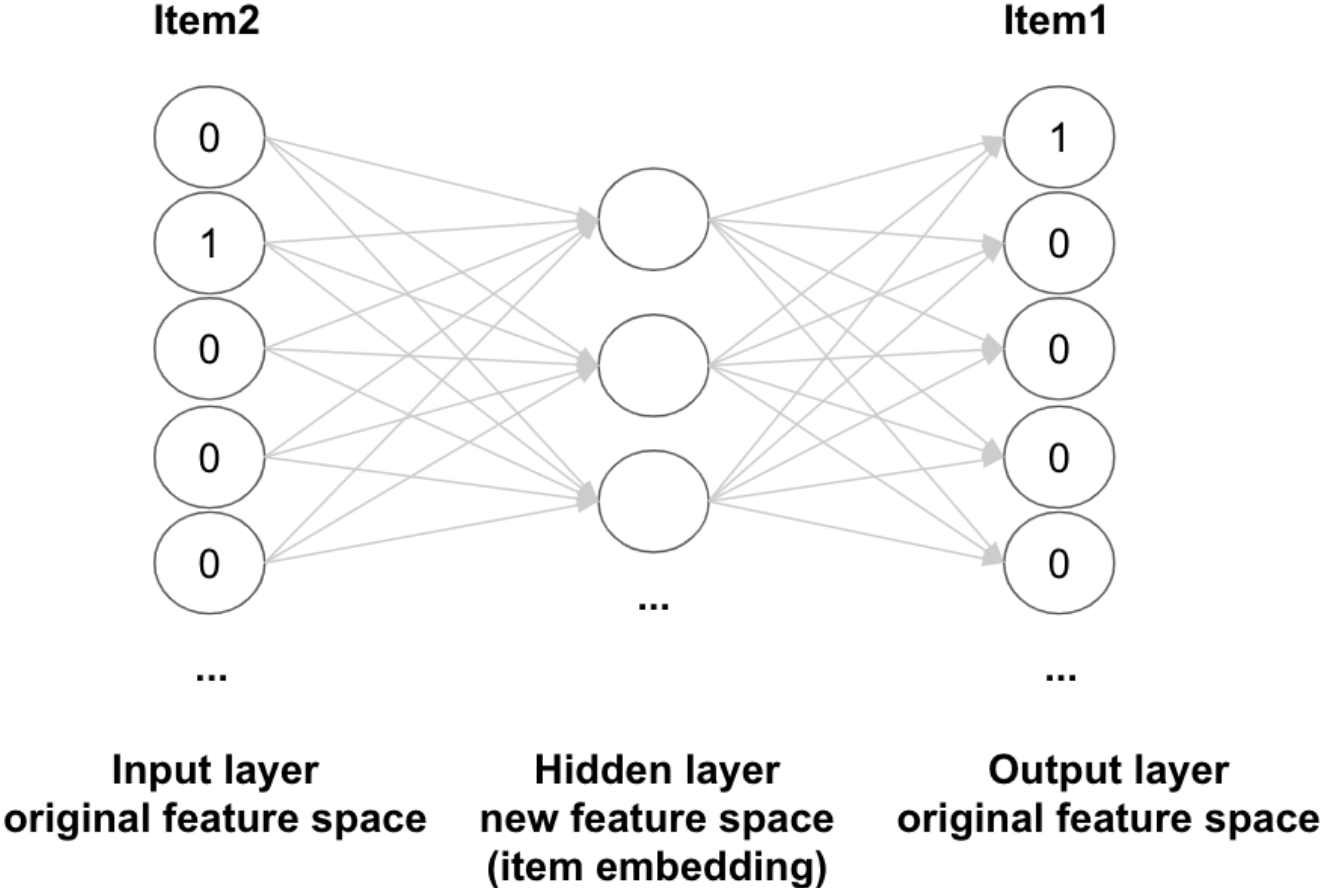
Deep learning is more flexible (than matrix factorization) in including various factors into modeling and creating embeddings. For example, deep learning was used to model sequential information by leveraging the skip-gram model, originally used for calculating word similarity. (Airbnb|2018, Zillow|2018)

Say, a user's item sequence is item1 -> item2 -> item 3 -> item4 -> ... The intuition is to use each item in the sequence to predict its neighboring items, formulated as a classification problem, where each item is one class. The training data include the neighboring K items of each item (the left K and right K items). The following figure illustrates the pairs of items with K = 1.

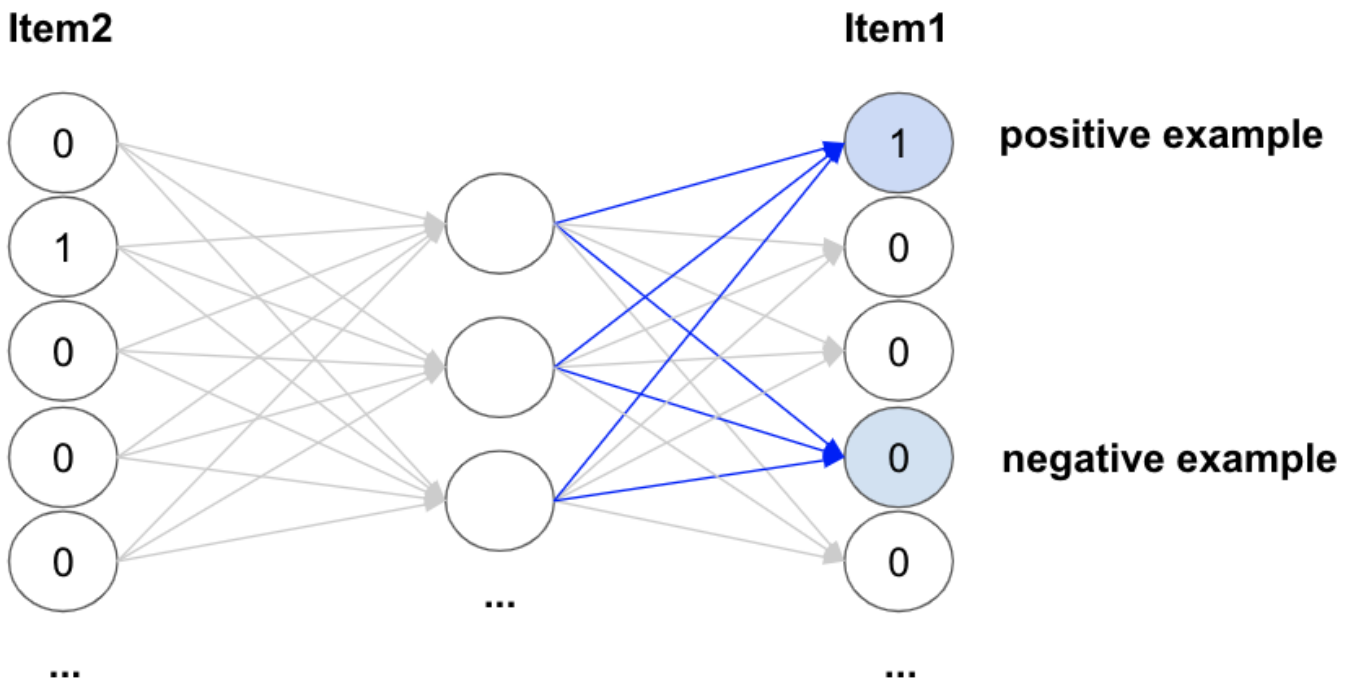


Furthermore, each item is represented as a one-hot vector that has a length equal to the number of items. A neural network takes an item one-hot vector as the input and output the vector of one of its similar items, illustrated in the following figure, using (Item2, Item1) as the training example.

The hidden layer is the new feature space (or latent space), and each item can be transferred to the new feature space using the weights between the input layer and the hidden layer (essentially a linear combination of the original features).



In reality, there can be millions of items, and billions of examples are used to train the network. To simplify the calculation, the *negative sampling* idea can be applied. The idea is to update only the weights of the output item (Item 1) and a small number of other items randomly sampled. In the following, we highlight the items and the weights need to be updated. It makes the calculation much faster.



Once each item is represented in the new feature space, the similarity between items can be calculated, and recommendations can be made based on similarity scores.

In some cases, users visit a sequence of items before conversion, e.g., an Amazon user makes a purchase after a sequence of page views; An Airbnb user books a listing after viewing a few listings. This information can be included by adding the purchased item to every item’s training pair (Airbnb|2018), shown in the figure below. Items recommended in this way could improve the conversion rates.



Hybrid approaches

Hybrid approaches use information from both user-item interactions and users/items' characteristics.

The "companies you may want to follow" feature at LinkedIn used both content and collaborative filtering information (LinkedIn|2014). To determine whether a company a user may want to follow, a logistic regression classifier is built on a set of features. The collaborative filtering information is included in a feature indicating whether the company is similar to the ones a user already followed. The content information includes whether the industry, location, etc. match between the user and the company.

Deep learning models can be powerful in combining collaborative-filtering and content-based information. The Youtube recommender system (Youtube|2016) built deep learning models to predict users' watch given their previous activities (search queries and videos watched) and static information (gender, location, etc.). Watched videos and queries are represented as embeddings. Since neural networks typically take fixed-length inputs, a user's watched videos or queries vectors are averaged, concatenated with other static features. It is recommended that features with multiple categories should be embedded into a much smaller space (roughly proportional to the logarithm of the number of unique values), and continuous features should be normalized between 0 and 1 (Youtube|2016).

Hybrid methods can depend on content-based recommendations when a user/item has no or little activity and become more accurate as more data are available.

Cold start, Scalability, Interpretability, and Exploitation-Exploration

Accurate recommendations cannot be made for new users/items with no or little information. This is referred to as the **cold start** problem. This is a typical issue for collaborative filtering systems that rely on user-item interactions. Some heuristics can be used. For a new user, the most popular items in the user's area could be recommended. For a new item, some rule-based similarity criteria can be defined. For example, Airbnb used the average of 3 geographically closest listings of the same type and price range to approximate a new listing (Airbnb|2018).

Scalability is a key factor when determining which type of recommender systems to use. More complex systems need more people, potentially harder to hire, to

build/maintain with a larger hardware cost. It can be a long-term commitment, and so business should understand the incremental business gain vs. the increased cost. With that being said, here are a few key elements of building scalable systems.

Offline batch computing and online serving. With a large number of users and items, one has to calculate easily fetchable recommendations offline by batch. For example, LinkedIn used Hadoop for batch processing the user-item event data, and then the recommendations are loaded to a key-value store for low-latency queries in real-time. (LinkedIn|2014)

Sampling. When dealing with millions of users and items, sampling can be considered, randomly sampling items or users, or removing items without significant user engagement.

Leveraging sparsity. In recommender systems, the user-item preference matrix is often very sparse with the majority of the entries being missing. Leveraging the sparsity can significantly reduce computational complexity (Amazon|2003).

Multi-phase modeling. The Youtube recommender system divided the modeling process into two steps. In the first phase, only user-item activity data are used to select hundreds of candidates out of millions. In the second phase, it is then feasible to use more information about the candidate videos for further selection and ranking. (Youtube|2016)

Scale deep networks. Although softmax or other functions are used in the output layer for training, during real-time serving time, the probability doesn't have to be calculated, and the nearest-neighbor approach can be used on the output from the last hidden layer. Negative sampling mentioned earlier can also be considered so that for each training example, only a small number of classes' weights are updated.

Interpretability. From the customer side, it can be helpful to state why a recommendation is made. When recommending a video to a user, Youtube added a link to the video that the user watched and triggered the recommendation (Youtube|2010).

From the modeling side, interpretability helps developers understand and debug the system. Content-based approaches are easy to interpret, while collaborative filtering models are harder to understand, especially in latent space. One can cluster the items or users based their original feature space or latent space (matrix factorization and deep learning), and check whether the objects from the same cluster share similar characteristics.

In addition, the t-SNE algorithm (Maaten|2018) can be used to project a high-dimensional space to 2-dimensional space for visualization (Zillow|2018). It can also be useful to have a tool so that one can quickly view recommendations as a sanity check, e.g., Airbnb developed an internal exploration tool for validating the recommendations (Airbnb|2018).

Exploitation-Exploration. Recommender systems should not overfit historical user-item preference data (exploitation), to avoid getting stuck in a local optimal.

First, one should avoid the training data being fully impacted by previous recommendations. Youtube includes videos embedded in other sites for training. The videos watched outside Youtube site are not from the recommender system, and can effectively surface new content (Youtube|2016). One may also consider injecting randomness into the system (e.g., making random recommendations).

Simple rules can be added to the system to increase the diversity of recommendations. For example, in Youtube recommendation (Youtube|2010), videos too similar to each other are removed, and the number of videos coming from the same channel is limited.

Methods from multi-armed bandits may also be applied. The *upper confidence bound* was applied by Uber eats to increase the diversity of recommended restaurants/dishes (Uber|2018). The idea of upper confidence bound is using the upper bound of the estimated success rate (e.g., order-rate, click-rate, watch-rate). When a new item arrives without any information, the confidence interval is $[0,1]$, and so the upper bound is 1. Therefore, the new item would have a high chance of being recommended. As the item gets more impressions, the estimation would be more accurate, and the upper bound would be closer to its actual value.

Closing

This article discusses methodologies and key perspectives for building a recommender system. In practice, companies should make choices based on multiple factors like accuracy, complexity and business impact, under realistic constraints on resources (e.g., engineers and software/hardware costs)