Candidate Generation

The purpose of candidate generation is to select the top k (let's say one-thousand) movies that you would want to consider showing as recommendations to the end-user. Therefore, the task is to select these movies from a corpus of more than a million available movies.

We'll cover the following	^

- Candidate generation techniques
  - Collaborative filtering
    - Method 1: Nearest neighborhood
    - Method 2: Matrix factorization
  - Content-based filtering
  - Generate embedding using neural networks/deep learning
- Techniques' strengths and weaknesses

In this lesson, we will be looking at a few techniques to generate media candidates that will match user interests based on the user's historical interaction with the system.

Candidate generation techniques#

The candidate generation techniques are as follows:

1. Collaborative filtering
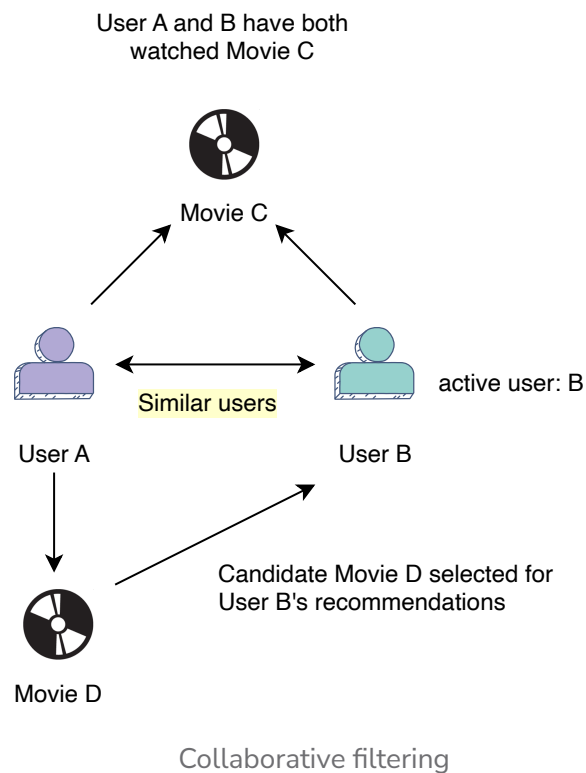2. Content-based filtering
3. Embedding-based similarity

Each method has its own strengths for selecting good candidates, and we will

combine all of them together to generate a complete list before passing it on

combine all of them together to generate a complete list before passing it on
to the ranked (this will be explained in the ranking lesson).

Collaborative filtering#

In this technique, you find users similar to the *active user* based on the
intersection of their historical watches. You, then, collaborate with similar
users to generate candidate media for the active user, as shown below.

User A and B have both
watched Movie C

Movie C

Similar users

active user: B

User A

User B

Candidate Movie D selected for
User B's recommendations

Movie D

Collaborative filtering

🔅 **Click to expand: Intuition behind collaborative filtering**

There are two methods to perform collaborative filtering:

1. Nearest neighborhood
2. Matrix factorization
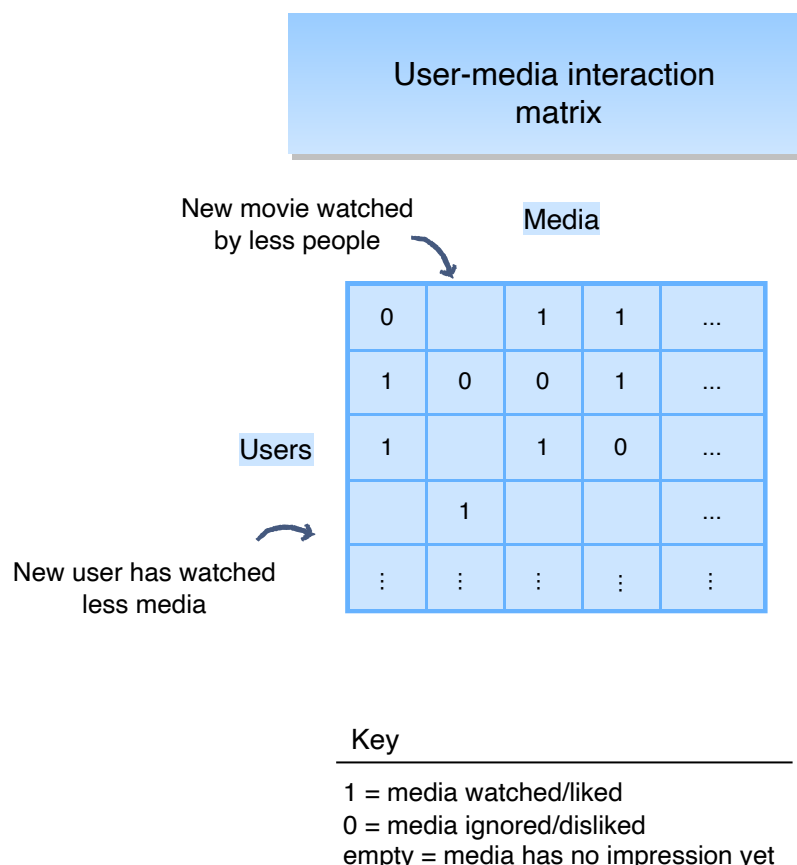
Method 1: Nearest neighborhood#

User A is similar to user B and user C as they have watched the movies Inception and Interstellar. So, you can say that user A's nearest neighbours are user B and user C. You will look at other movies liked by users B and C as candidates for user A's recommendations.

Nearest neighborhood
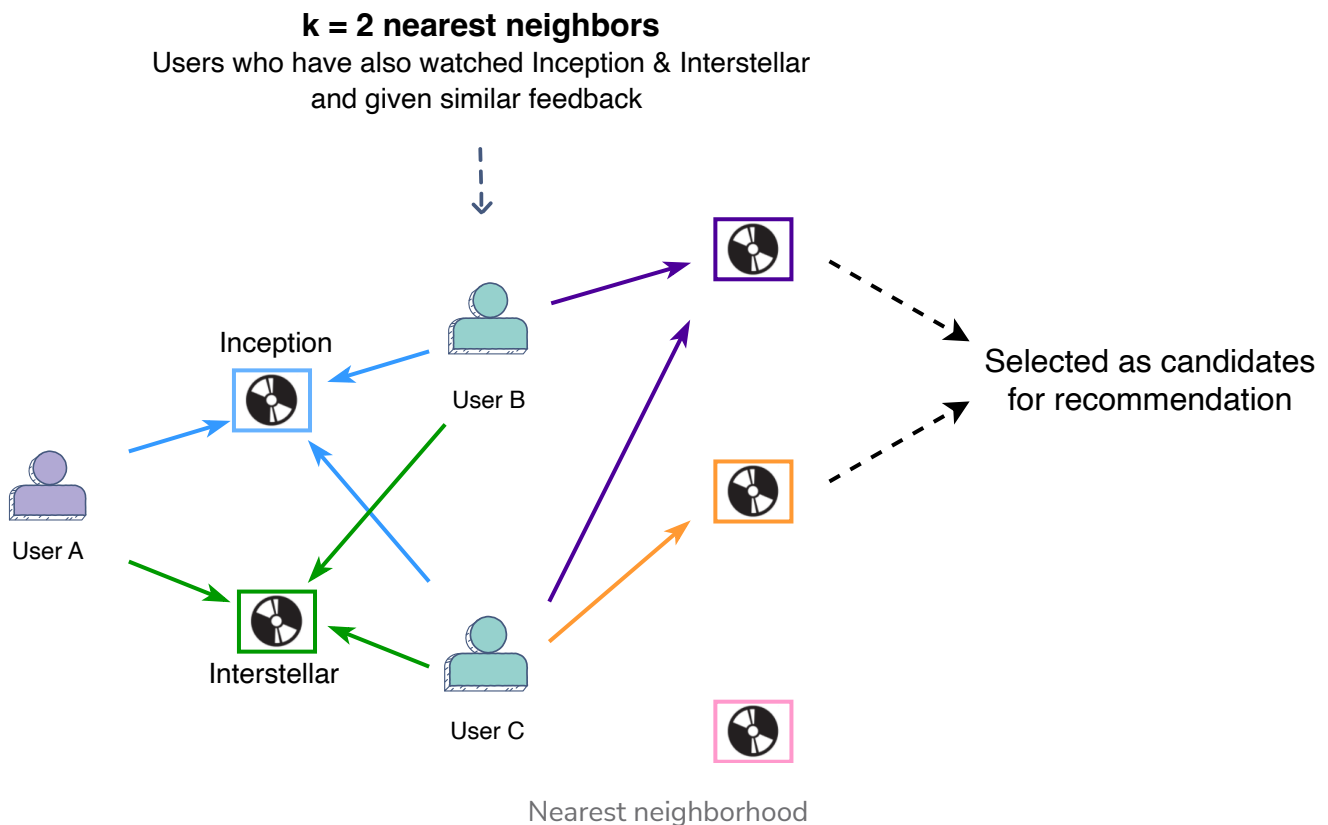
Let's see how this concept is realized. You have a (n x m) matrix of user u_i $(i = 1\ to\ n)$ and movie $m_{j\,(j\ =\ 1\ to\ m)}$. Each matrix element represents the feedback that the user i has given to a movie j. An empty cell means that user i has not watched movie j.

User-media interaction
matrix

New movie watched
by less people

Media

0		1	1	...
1	0	0	1	...
1		1	0	...
	1			...
⋮	⋮	⋮	⋮	⋮

Users

New user has watched
less media

Key
_____

1 = media watched/liked
0 = media ignored/disliked
empty = media has no impression yet

To generate recommendations for user i, you need to predict their feedback for all the movies they haven't watched. You will collaborate with users similar to user i for this process. Their ratings for a movie, not seen by user i, would give us a good idea of how user i would like it.

So, you will compute the similarity (e.g. cosine similarity) of other users with user i and then select the top k similar users/nearest neighbours (KNN(u_i)).

Then, user i's feedback for an unseen *movie j* (f_{ij}) can be predicted by taking the weighted average of feedback that the top k similar users gave to movie j. Here the feedback by the nearest neighbour is weighted by their similarity with user i.

$$f_{ij} = \frac{\sum\limits_{v \,\in\, KNN(u_i)} Similarity(u_i, u_v) f_{vj}}{k}$$

The unseen movies with good predicted feedback will be chosen as

candidates for user *i's* recommendations.

> 💡 **Collaborative filtering by identifying similar media**

It is evident that this process will be computationally expensive with the increase in numbers of users and movies. The sparsity of this matrix also poses a problem when a movie has not been rated by any user or a new user has not watched many movies.

Method 2: Matrix factorization#

As explained above, you need to represent the *user-media interaction matrix* in a way that is scalable and handles sparsity. Here, matrix factorization helps us by factoring this matrix into two lower dimensional matrices:
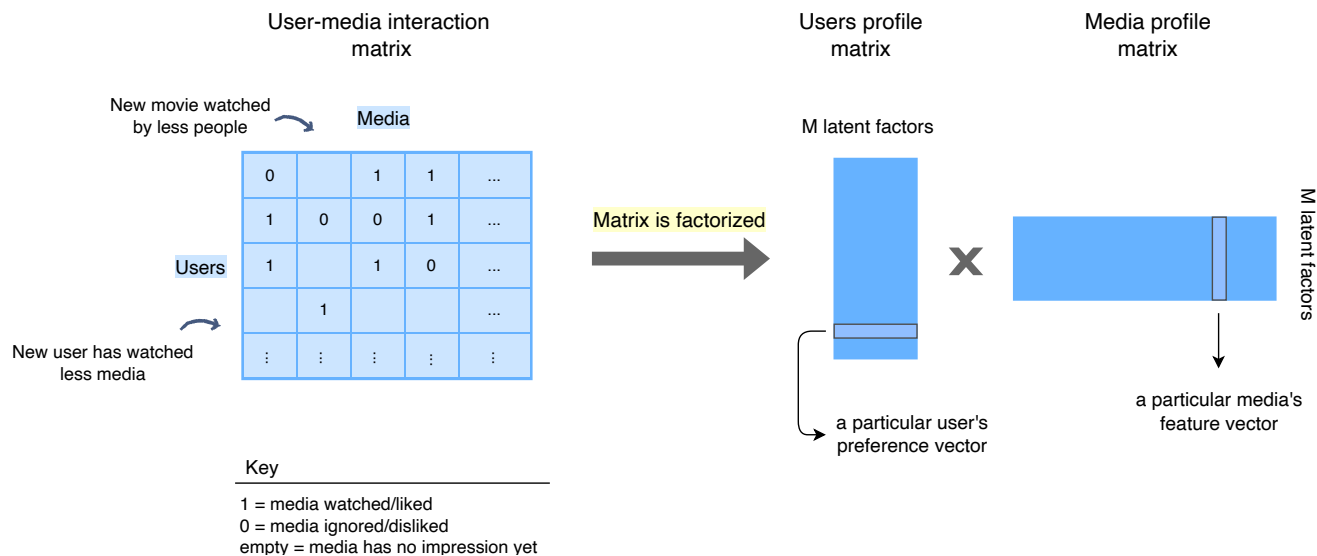
1. User profile matrix (n x M)

   Each user in the user profile matrix is represented by a row, which is a *latent vector of M dimensions*.

2. Media profile matrix (M x m)

   Each movie in the movie profile matrix is represented by a column, which is a latent vector of M dimensions.

The dimension M is the number of latent factors we're using to estimate the user-media feedback matrix. *M* is much smaller than the actual number of users and number of media.
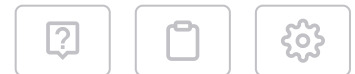
Factorization of user-media interaction matrix into two smaller factors (user profile matrix and media profile matrix)

The representation of users and media in the form of a *vector of latent factors* aims to explain the reason behind a user's particular feedback for a media. Latent vectors can also be thought of as features of a movie or a user. For example, in the diagram below, two of the latent factors might loosely represent the amount of comedy and quirkiness.

> 📝 Note that we say the word "might" and "loosely" for latent factors because we don't know exactly what each dimension means, it is "hidden" from us.

User A's vector has their preferences: 1 for the "comedy" and 0 for "quirkiness". Movie B's vector contains the presence/absence of the two factors in the movie, e.g., 1 and 0. The *dot product of these two vectors* will tell how much movie B aligns with the preferences of the user A; hence, the feedback.
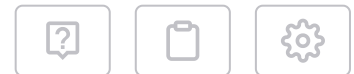
Movie B had a particular combination of latent factors (**might** be comedy, quirkiness) which was similar to user A's liking hence explaining his positive feedback for Movie B
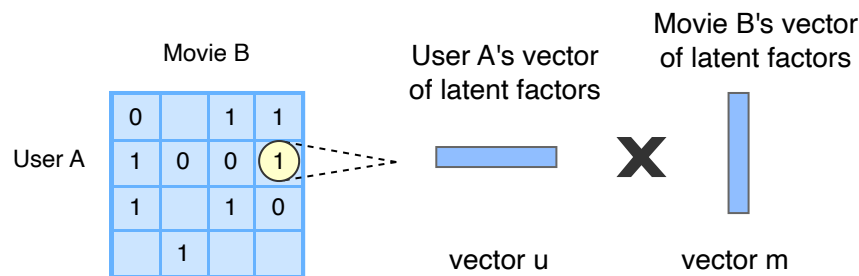
User A's feedback for movie B is captured in the form of alignment between A's preferences and B's characteristics

The first step is to create the user profile and movie profile matrices. Then, you can generate good candidates for movie recommendation by predicting user feedback for unseen movies. This prediction can be made simply by computing the dot product of the user vector with the movie vector.

Now, let's go over the process of how to learn the latent factor matrices for users and media.

You will initialize the user and movie vectors randomly. For each known/historical user-movie feedback value f_{ij}, you will predict the movie feedback by taking the dot product of the corresponding user profile vector u_i and movie profile vector m_j. The difference between the actual (f_{ij}) and the predicted feedback ($u_i.m_j$) will be the error (e_{ij}).

$$e_{ij} = f_{ij} - u_i.m_j$$

You will use stochastic gradient descent to update the user and movie latent vectors, based on the error value. As you continue to optimize the user and movie latent vectors, you will get a semantic representation of the users and movies, allowing us to find new recommendations that are closer in that space.

📝 The user profile vector will be made based on the movies that the user has given feedback on. Similarly, the media/movie profile vector will be made based on its user feedbacks. By utilizing these user and movie profile vectors, you will generate candidates for a given user based on their predicted feedback for unseen movies.

Content-based filtering#

Content-based filtering allows us to make recommendations to users based on the characteristics or attributes of the media they have already interacted with.

Collaborative filtering
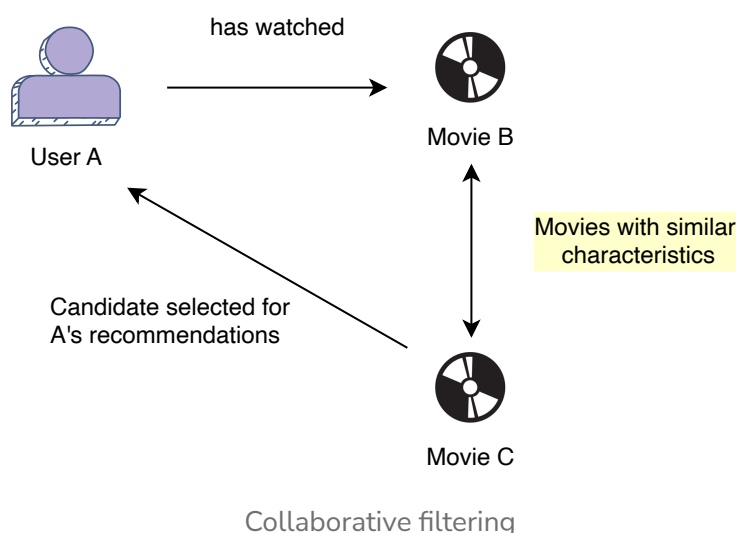
As such the recommendations tend to be relevant to users' interest. The characteristics come from metadata (e.g., genre, movie cast, synopsis, director, etc.) information and manually assigned media-descriptive-tags (e.g., visually striking, nostalgic, magical creatures, character development, winter season, quirky indie rom-com set in Oregon, etc.) by Netflix taggers. The media is represented as a vector of its attributes. The following explanation takes a subset of the attributes for ease of understanding.

Media Attributes

	Director	Genre	Tags
Inception	Christopher Nolan	Action, Adventure, Sci fi	Psychological manipulation, Mind bender ...
Interstellar	Christopher Nolan	Adventure, Sci fi, Drama	Time paradox ...
Baby's day out	Patrick Read Johnson	Drama, Comedy ...	1990s

Attributes are preprocessed (stop words removed, words converted to lowercase, tags with multiple words joined) to extract features

Attribute preprocessing

Extracted features

	Patrick Read Johnson	Christopher Nolan	Action	Adventure	Sci fi	Drama	Comedy	Psychological manipulation	Mind Bender	Time Paradox	1990s
Inception	0	1	1	1	1	0	0	1	1	0	0
Interstellar	0	1	0	1	1	1	0	0	0	1	0
Baby's day out	1	0	0	1	0	1	1	0	0	0	1

Term Frequencies (Binary Representation)

Media represented as a vector of attributes

Initially, you have the media's attributes in raw form. They need to be preprocessed accordingly to extract features. For instance, you need to remove stop words and convert attribute values to lowercase to avoid duplication. You also have to join the director's first name and last name to identify unique people since there maybe two directors with the first name Christopher. Similar preprocessing is required for the tags. After this, you are able to represent the movies as a vector of attributes with elements depicting the term frequency (TF) (binary representation of attribute's presence (1) or absence (0)).
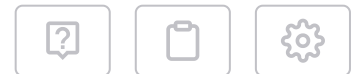
Feature Vector Representation of Media (TF)

	Patrick Read Johnson	Christopher Nolan	Action	Adventure	Sci fi	Drama	Comedy	Psychological manipulation	Mind Bender	Time Paradox	1990s		# of features present in media
Inception	0	1	1	1	1	0	0	1	1	0	0		6
Interstellar	0	1	0	1	1	1	0	0	0	1	0		5
Baby's day out	1	0	0	1	0	1	1	0	0	0	1		5

Normalize feature occurrence with the length of vector ,i.e. sqrt(# of features present in the media)

Normalise TF vector

	Patrick Read Johnson	Christopher Nolan	Action	Adventure	Sci fi	Drama	Comedy	Psychological manipulation	Mind Bender	Time Paradox	1990s
Inception	0.00	1/sqrt(6) = 0.41	0.41	0.41	0.41	0.00	0.00	0.41	0.41	0.00	0.00
Interstellar	0.00	0.45	0.00	0.45	0.45	0.45	0.00	0.00	0.00	0.45	0.00
Baby's day out	0.45	0.00	0.00	0.45	0.00	0.45	0.45	0.00	0.00	0.00	0.45

→ Tf vector of each movie

DF = feature occurrence frequency in movies

2	3	5	7	6	5	5	4	3	3	3

← Frequencies assumed w.r.t 10 movies

$IDF = log_{10}\left(\frac{no\ of\ movies\ in\ corpus}{DF}\right)$

0.69	0.52	0.30	0.15	0.22	0.30	0.30	0.39	0.52	0.52	0.52

→ IDF vector

Assuming that corpus has 10 movies

Find TF-IDF (TF*IDF)

TF-IDF vector

	Patrick Read Johnson	Christopher Nolan	Action	Adventure	Sci fi	Drama	Comedy	Psychological manipulation	Mind Bender	Time Paradox	1990s
Inception	0.00	0.21	0.12	0.06	0.09	0.00	0.00	0.16	0.21	0.00	0.00
Interstellar	0.00	0.23	0.00	0.07	0.10	0.14	0.00	0.00	0.00	0.23	0.00
Baby's day out	0.31	0.00	0.00	0.07	0.00	0.14	0.14	0.00	0.00	0.00	0.23

0.45 x 0.69

Click to enlarge: making TF-IDF vector representations of media

Now, you have the movies represented as vectors containing the TF (term/attribute frequency) of all the attributes. This is followed by normalizing the term frequencies by the length of the vectors. Finally, you multiply the movies' normalized TF vectors and the IDF vector element-wise to make TF-IDF vectors for the movies.

Click to expand: TF-IDF

Given the TF-IDF representation of each movie/show, you have two options for recommending media to the user:
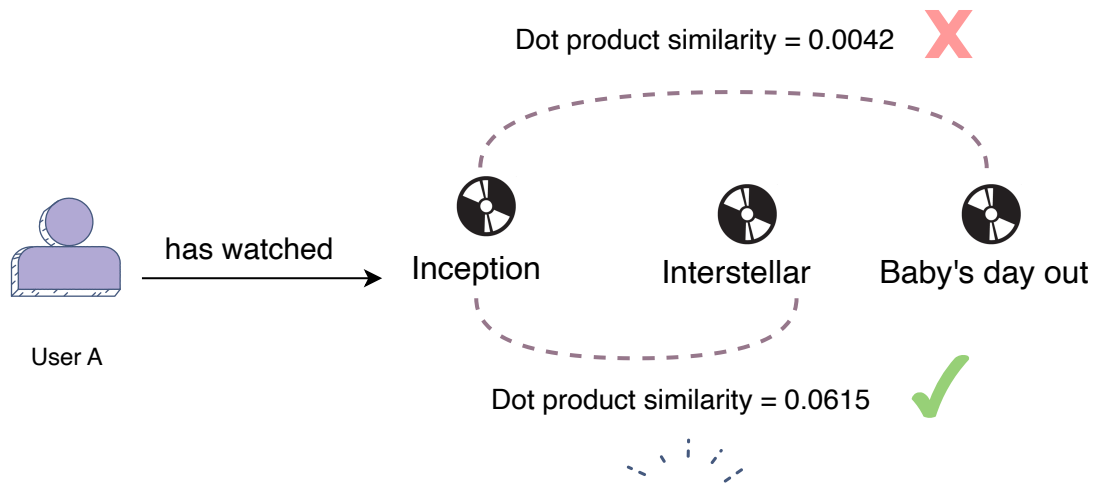
1. Similarity with historical interactions

You can recommend movies to the user similar to those they have

interacted (seen) with in the past. This can be achieved by computing the dot product of movies.

Recommend **one** movie to user A

Dot product similarity = 0.0042 ❌

User A — has watched → Inception   Interstellar   Baby's day out
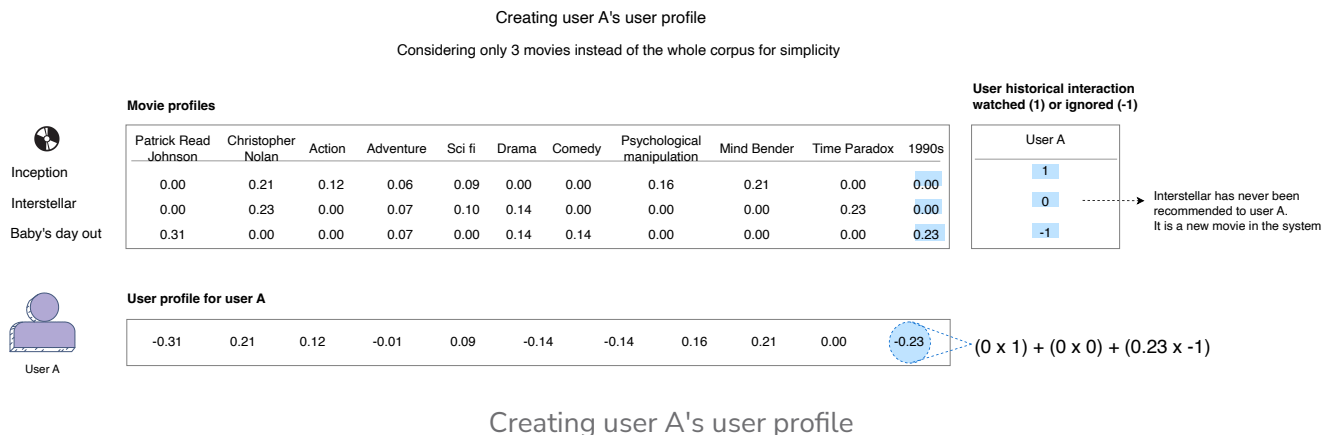
Dot product similarity = 0.0615 ✓

Candidate movie selected for recommendation

Interstellar is selected as a candidate for recommendation to user A based on their historical watch (Inception)

2. Similarity between media and user profiles

The media's TF-IDF vectors can be viewed as their *profile*. Based on user preferences, which can be seen from its historical interactions with media, you can build *user profiles* as well as shown below.

Creating user A's user profile

Considering only 3 movies instead of the whole corpus for simplicity

Movie profiles

	Patrick Read Johnson	Christopher Nolan	Action	Adventure	Sci fi	Drama	Comedy	Psychological manipulation	Mind Bender	Time Paradox	1990s
Inception	0.00	0.21	0.12	0.06	0.09	0.00	0.00	0.16	0.21	0.00	0.00
Interstellar	0.00	0.23	0.00	0.07	0.10	0.14	0.00	0.00	0.00	0.23	0.00
Baby's day out	0.31	0.00	0.00	0.07	0.00	0.14	0.14	0.00	0.00	0.00	0.23

User historical interaction watched (1) or ignored (-1)

User A: 1, 0, -1

Interstellar has never been recommended to user A. It is a new movie in the system

User profile for user A

| -0.31 | 0.21 | 0.12 | -0.01 | 0.09 | -0.14 | -0.14 | 0.16 | 0.21 | 0.00 | -0.23 |

(0 x 1) + (0 x 0) + (0.23 x -1)

Creating user A's user profile

Now, instead of using past watches to recommend new ones, you can just compute the similarity (dot product) between the user's profile and media

profiles of unseen movies to generate relevant candidates for user A's recommendations.

Choosing two candidate movies for user A based on his profile

Considering only 3 movies instead of the whole corpus for simplicity

Movie profiles

Movie X	0.00	0.00	0.22	0.02	0.01	0.30	0.20	0.00	0.00	0.00	0.00
Interstellar	0.00	0.23	0.00	0.07	0.10	0.14	0.00	0.00	0.00	0.23	0.00
Movie Y	0.00	0.00	0.32	0.03	0.20	0.01	0.00	0.00	0.00	0.30	0.23

User profile for user A

-0.31	0.21	0.12	-0.01	0.09	-0.14	-0.14	0.16	0.21	0.00	-0.23

Matching: dot product similarity

Movie X -0.0429 ✗
Movie Y 0.0547 ✓
Insterstellar 0.0370 ✓
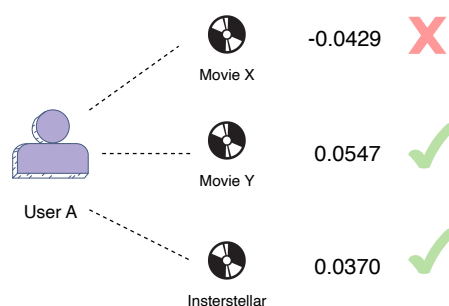
Choosing two candidate movies for user A based on their profile

Generate embedding using neural networks/deep learning#

Given the historical feedback (u,m), i.e., user u's feedback for movie m, you can use the power of deep learning to generate latent vectors/embeddings to represent both movies and users. Once you have generated the vectors, you will utilize KNN (k nearest neighbours) to find the movies that you would want to recommend to the user. This method is similar to generating latent vectors, that you saw in matrix factorization but is much more powerful because using NNs allows us to use all the sparse and dense features in the data to generate user and movie embedding.

Embedding generation

You set up the network as two towers with one tower feeding in *media only* sparse and dense features and the other tower feeding in *user-only* sparse and dense features. The activation of the first tower's last layer will form the
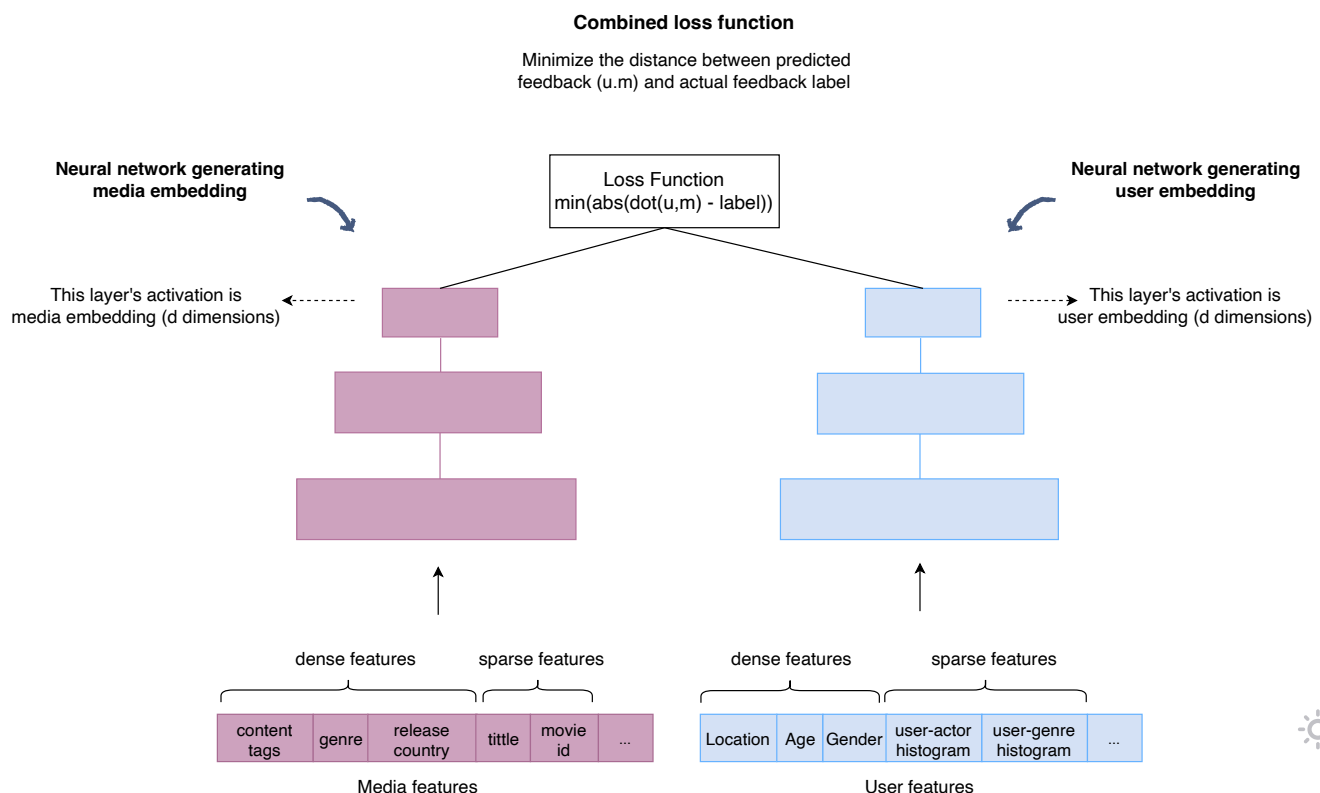
media's vector embedding *(m)*. Similarly, the activation of the second tower's last layer will form the user's vector embedding *(u)*. The combined optimization function at the top aims to minimize the distance between the dot product of u and m (predicted feedback) and the actual feedback label.

$$min(|dot(u,m) - label|) \quad where \ u,m \in R^d, \ d = dimension \ of \ u \ and \ v's \ embedding$$

Let's look at the intuition behind this cost function. The actual feedback label will be positive when the media aligns with user preferences and negative when it does not. To make the predicted feedback follow the same pattern, the network learns the user and media embeddings in such a way that their distance will be minimized if the user would like this media and maximized if the user would not like the media.

> 📝 The vector representation of the user, which you would get as a result, will be nearest to the kind of movies that the user would like/watch.

Combined loss function

Minimize the distance between predicted feedback (u.m) and actual feedback label

Neural network generating media embedding

Loss Function
min(abs(dot(u,m) - label))

Neural network generating user embedding

This layer's activation is media embedding (d dimensions)

This layer's activation is user embedding (d dimensions)

dense features   sparse features

dense features   sparse features

content tags	genre	release country	tittle	movie id	...

Location	Age	Gender	user-actor histogram	user-genre histogram	...

Media features

User features

Generating user and movie embedding through neural networks with a combined loss function

📝 The feature *user-actor histogram* shows the percentage of media watched by the user with particular actors (cast) in it. For instance, let's say 2% of the content viewed by the user had Brad Pitt in it, and 5% had Leonardo DiCaprio in it.
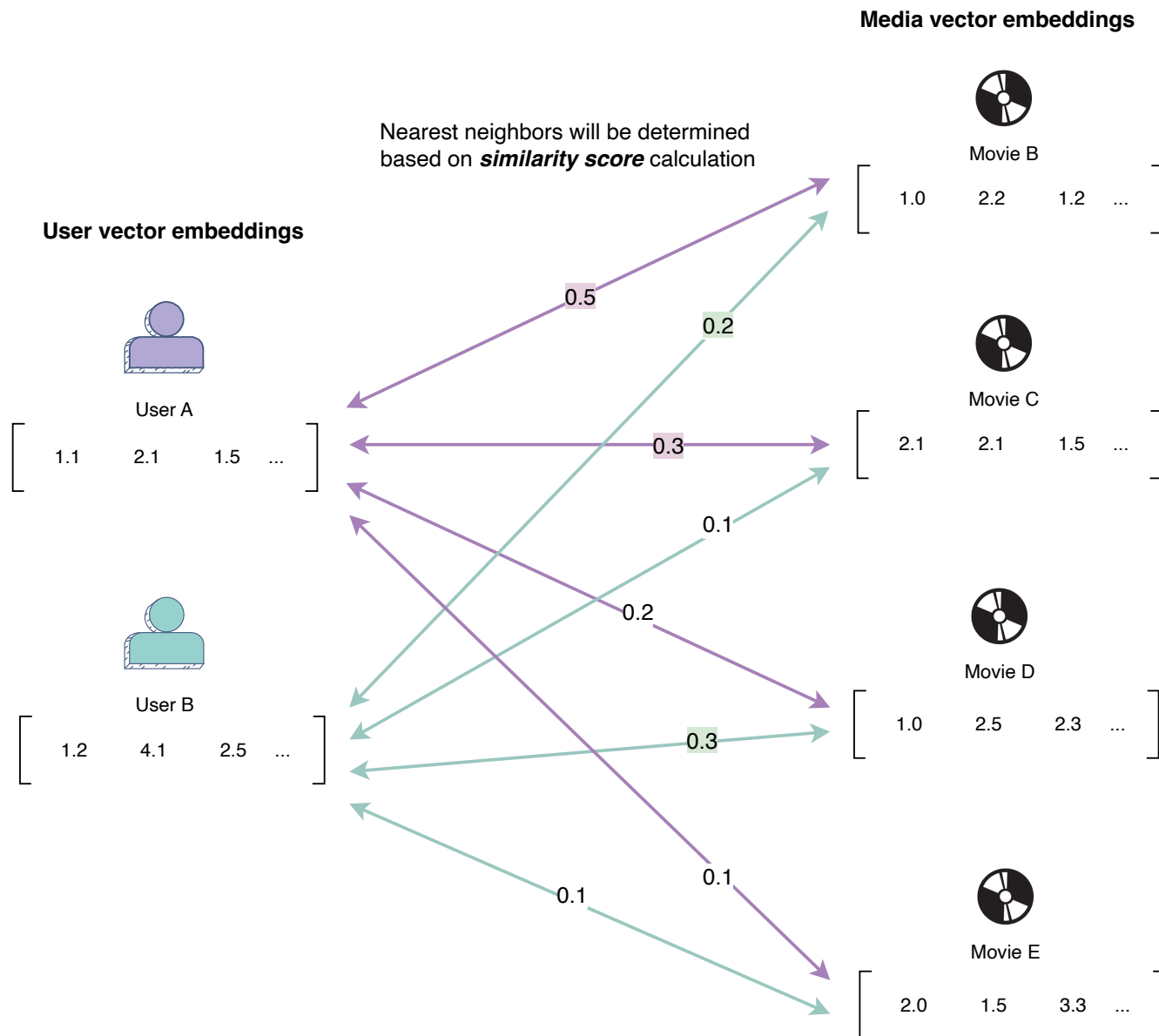
💡 Click to expand: Hyper parameters

Candidate selection (KNN)

Let's assume that you have to generate two candidates for each user, as shown in the diagram below. After the user and media vector embeddings have been generated, you will apply KNN and select candidates for each user. It can be observed in the diagram below, that user A's nearest neighbors are movie B and movie C, based on high *vector similarity*. Whereas, for user B, movie D and movie B are the nearest neighbors.

Finding k nearest neighbors(media) for the users

Media vector embeddings

Nearest neighbors will be determined
based on *similarity score* calculation

User vector embeddings

User A

$$\begin{bmatrix} 1.1 & 2.1 & 1.5 & ... \end{bmatrix}$$

User B

$$\begin{bmatrix} 1.2 & 4.1 & 2.5 & ... \end{bmatrix}$$

Movie B

$$\begin{bmatrix} 1.0 & 2.2 & 1.2 & ... \end{bmatrix}$$

Movie C

$$\begin{bmatrix} 2.1 & 2.1 & 1.5 & ... \end{bmatrix}$$

Movie D

$$\begin{bmatrix} 1.0 & 2.5 & 2.3 & ... \end{bmatrix}$$

Movie E

$$\begin{bmatrix} 2.0 & 1.5 & 3.3 & ... \end{bmatrix}$$

0.5   0.2   0.3   0.1   0.2   0.3   0.1   0.1

Finding k=2 nearest neighbor (movies) for users based on their vector similarity (dot product)

Techniques' strengths and weaknesses#

Let's look at some strengths and weaknesses of the approaches for candidate generation discussed above.

Collaborative filtering can suggest candidates based solely on the historical

interaction of the users. Unlike content-based filtering, it *does require*

domain knowledge to create user and media profiles. It may also be able to capture data aspects that are often elusive and difficult to profile using content-based filtering. However, collaborative filtering suffers from the *cold start problem*. It is difficult to find users similar to a new user in the system because they have less historical interaction. Also, new media can't be recommended immediately as no users have given feedback on it.

The **neural network technique** also suffers from the *cold start problem*. The embedding vectors of media and users are updated in the training process of the neural networks. However, if a movie is new or if a user is new, both would have fewer instances of feedback received and feedback given, respectively. By extension, this means there is a lack of sufficient training examples to update their embedding vectors accordingly. Hence, the cold start problem.

Content-based filtering is superior in such scenarios. It does require some initial input from the user regarding their preferences to start generating candidates, though. This input is obtained as a part of the onboarding process, where a new user is asked to share their preferences. *Once we have the initial input, it can create and then match the user's profile with media profiles*. Moreover, new medias' profiles can be built immediately as their description is provided manually.

← **Back**

Feature Engineering

Next →

Training Data Generation

✔ Completed