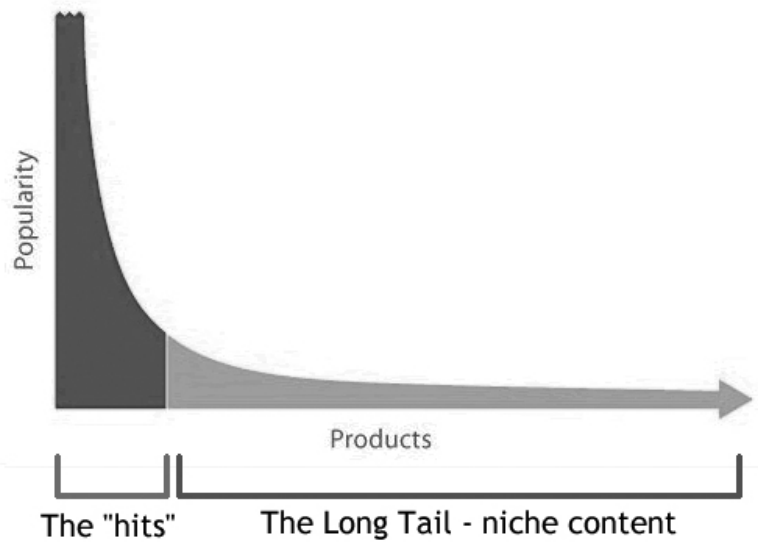
The topic of recommender systems is a very advanced topics that can get very complicated really fast. The key to approaching such a big topic is to break it down into smaller subproblems that gradually increase in complexity in exchange for better results.

Recommendation systems are currently used everywhere. Youtube and Tiktok recommend videos for you that are very personalized. Amazon recommends products to you based on your needs, purchase history, time of day and month. Even if you purchased baby products 5 years ago, they know your child is now older and they should stop recommending baby products, and recommend other products. Facebook recommends friends, and twitter recommends tweets and topics to follow. The possibilities are endless, you just have to be creative enough to see where your recommendation system fits.

The need for personalization arose with the start of the internet. Before Amazon for example, stores like Walmart or Carrefour had limited shelf space, and thus they were limited to selling the products that made the most profit, because it was not profitable to store less popular items.

You might think that the same applies to online retail, but let's look at a famous plot that tends to illustrate this idea.



https://miloszkrasinski.com/wp-content/uploads/2015/09/the-long-tail-effect.jpg
This plot is known as the long-tail plot, and it really fits almost everything around us. The graph means that very few products are "hits" and are the best-sellers or the top-viewed videos, etc. This is true, and these products or videos are very profitable to recommend, but looking at the long-tail of this plot shows us a very interesting idea. The area under the curve for most of these

niche products represents a very considerable amount of the “sales” or “profits” or “views” of the total area under the curve. This means that by neglecting them you will be losing on a huge opportunity that is enough to make you lose your edge in a very competitive market.

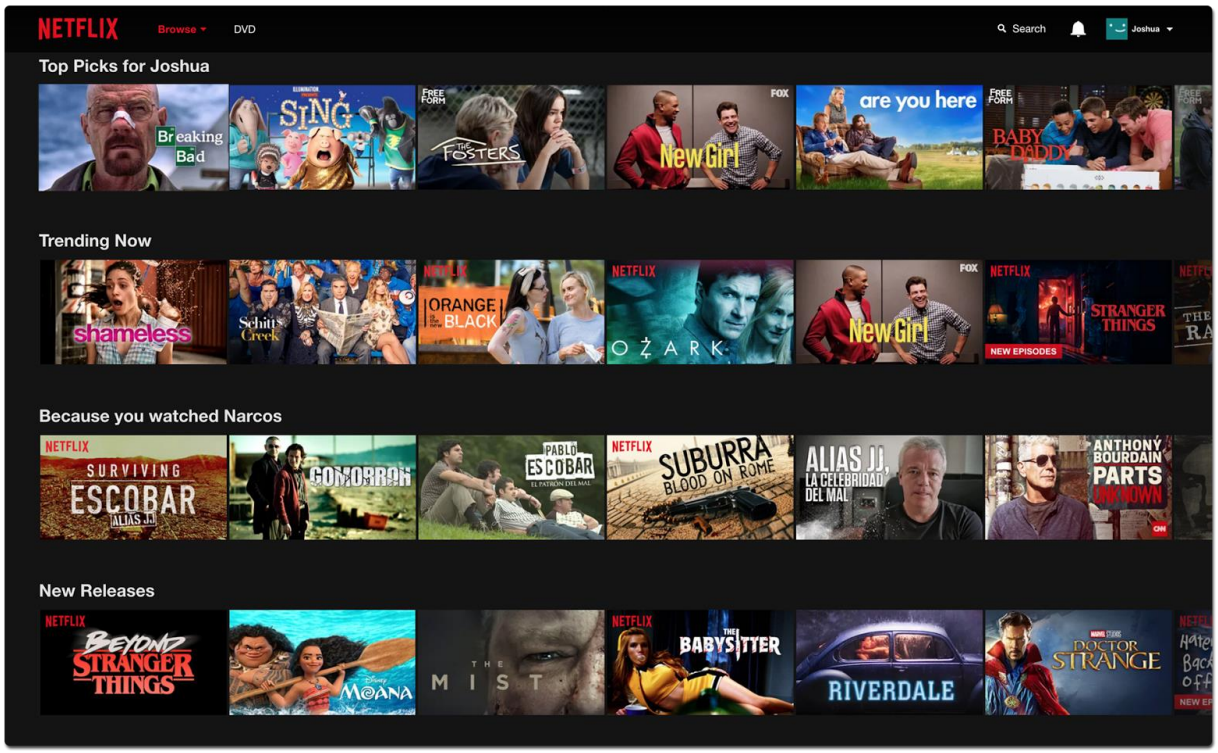
There are a lot of other concepts that are useful to know at this stage, but I will probably spend 1 week writing this article if I had to explain each one of them, so I will mention them throughout the notebook and you do the reading if you want to know more.

Here is a summary of most of the recommendation system types out there.

- Collaborative Filtering
- Content-Based Techniques
- Knowledge-Based Techniques
- Stereotype-Based Recommender Systems
- Ontologies and Semantic Web Technologies for Recommender Systems
- Community Based Recommender Systems
- Demographic Based Recommender Systems
- Context Aware Recommender Systems
- Conversational/Critiquing Recommender Systems
- Hybrid Techniques

Let’s start our notebook by mentioning the most basic way of recommending things:

1. The popularity based approach
   This approach is really simple, you look at the most popular items in your pool of items, and you recommend them to users. Very easy to build, but has two problems.



The first problem is that they don't offer any kind of personalization. All users will get the same recommendation to watch the shawshank redemption because it is the most popular movie, even if the user was a 5-year-old child who wants to watch baby shark.

The second problem is that their is a big chance that since this item is the most popular in your pool of items that the user has already seen it before or bought that book before, etc. you don't want to do this or this will increase the customer churn-off rate.

To deal with this, companies have moved to the second type of recommendation engines which is

2. The "trending now" model
   Basically, it is the same as the popularity based approach but it has a sliding window that makes sure the recommendations are all from new content only that is currently trending. A trending video doesn't have to be the most viewed, but rather a video that is getting views at an alarming rate (high slope), so you start recommending it.

Now, we shift from the old era of non-personalized one-size fits all to the era of personalized recommendations.

Before starting, we have to understand two types of features that recommendation engines use in they analysis:

- Explicit features: these are the features that provided by the user directly.
  They are divded into two parts:

    - User-related explicit feature: these are the features that are explicitly supplied by the user during the signup process for example such as his age, gender, languages, country, religion, etc. these features can be extremely useful, but they are very few. These are vital to the functionality of your recommendation system. You cannot recommend a french video to a user who only speaks mandarin, or a chinese video to a south-african man. You generally want to avoid recommending cartoons to grown-ups, unless the grown-up is Mina Ashraf, so it is a different story here. Anyway, I am sure you get the idea.

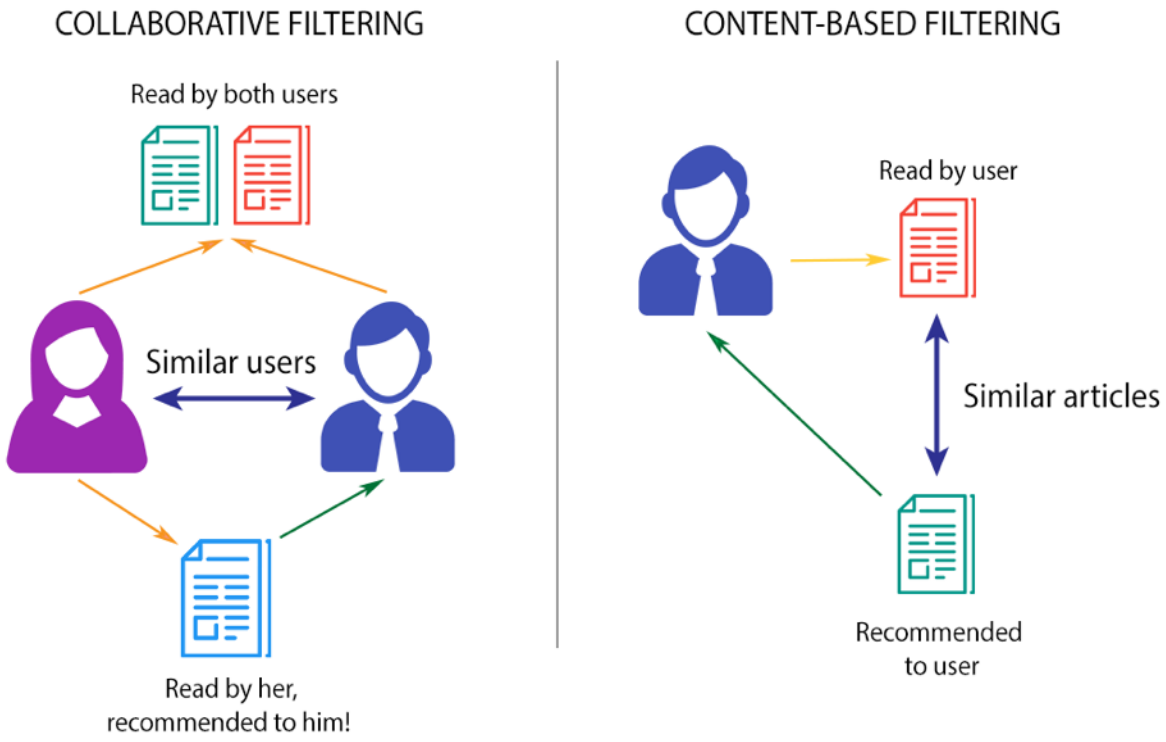
    - item-related explicit features:
      These are the features related to the product itself. Its color, weight, length in minutes for videos, size and dimensions, indoor or outdoor product, etc. these are quite rare. For example, out of the billions of views generated on YouTube everyday, only less than 0.01% of it is converted into likes, and even less into comments or shares.

- Implicit features:
  This is where it gets really interesting. These are the features that the user don't tell to you, but if you look closely, you will learn them. For example, the products in the wishlist on amazon, the pages you follow on Facebook, the items you add to your cart and never buy, the flights you viewed their prices but never booked a ticket, the watchtime of a video on YouTube (both by the user or for the video itself). generally, these features are the hardest to extract

One final concept to introduce before jumping in is the sparsity of the our user-items matrix. YouTube has billions of users, and millions, if not billions, of videos, but how much video do you think each of the users watch during his lifetime? Thousands at most.
Thus, we get a very very sparse interaction matrix where most of the matrix is empty. Filling these values by zeroes will explode the largest supercomputer on earth beside not being the right solution to begin with. If you are rating products from 0-10, a 0 for null values implies that the user will not like the video so don't recommend it to them. Filling it with any other value is equally bad, we need to make educated guesses.

If you are interested, the topic of sparse matrices is really amazing and you can learn a lot about computer architecture and linear algebra just by learning the different types of sparse matrices and their uses. Read more about CSC, COO, CSR, CSC, LIL matrices.

Here we can divide our recommendation systems into 2 big categories.



- The content-based recommendation systems
  you only recommend items similar to other items in your data. If your user likes toy-story, recommend to him Toy story 2 and 3. If he likes Marvel avengers, recommend Dr strange to him, etc. you measure similarities between content.

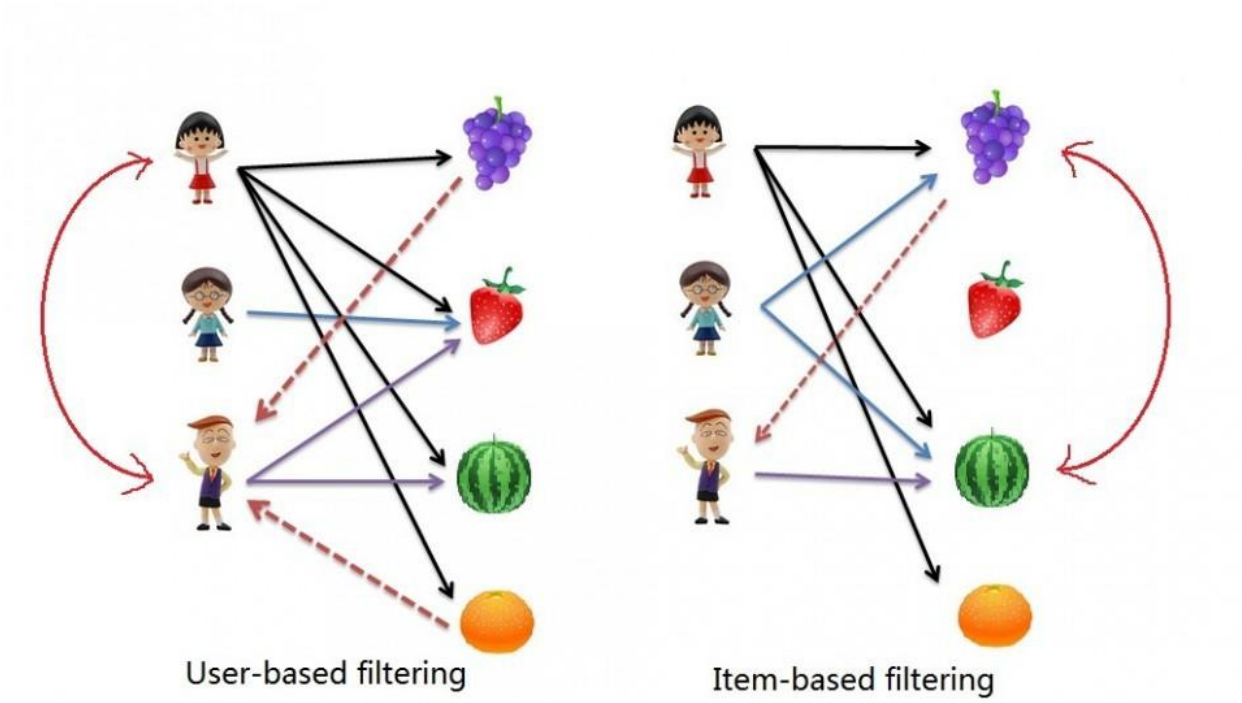
  The collaborative filtering recommendation systems
  - Memory-based recommendation systems
    - **User-User Collaborative Filtering**
      Let's say you noticed that mina interacts with the content very similarly to fawzy, then you recommend what fawzy likes to mina, and the other way around. It is like the KNN we used before, but since the dimensions of the data are huge, you need to be clever while making your computations.

    - **Item-item Collaborative Filtering**
      Here, instead of looking at the users, you look at the items (columns instead of rows), and do the same operation. You basically observe if item x gets very similar interactions to item Y. recommend item x to users who liked item Y, and vice versa. It gets very challenging when the dimensions of the data increase.

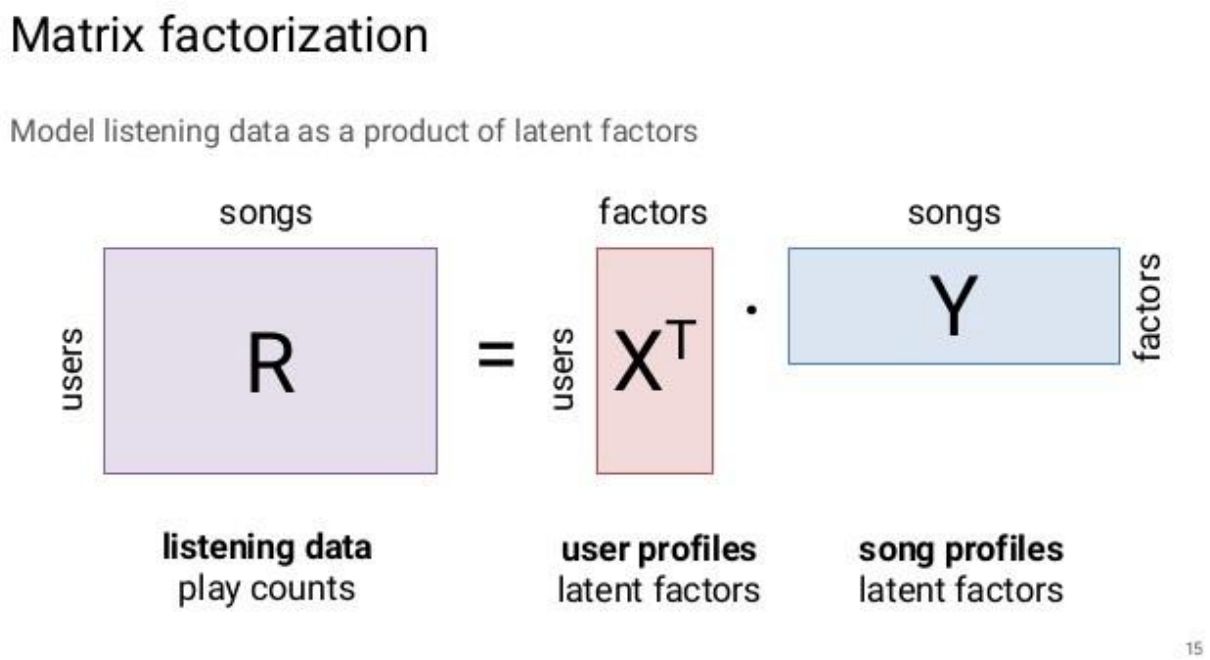
- Model-based recommendation systems
    - Matrix-factorization models (Latent-factor models using SVD)
      Very simply, we have an interaction matrix between users and movies that is mostly sparse. We need to factorize this matrix into two parts. A user representation matrix that is generally much smaller (number_of_users x user_embeddings_size)
      And another matrix for movies (movies_embeddings_size x number_of_movies)

      Multiplying these two matrices together would give you the full-interaction matrix, but it will not be sparse this time. You can look up the highest ratings that were not in the original matrix and recommend them to users. This is considering that you learned a good representation of the users and movies in your latent factors.

      The main problem here is interpretability. Similar to NLP embeddings, we don't know how they work, but we can only guess and interpolate.

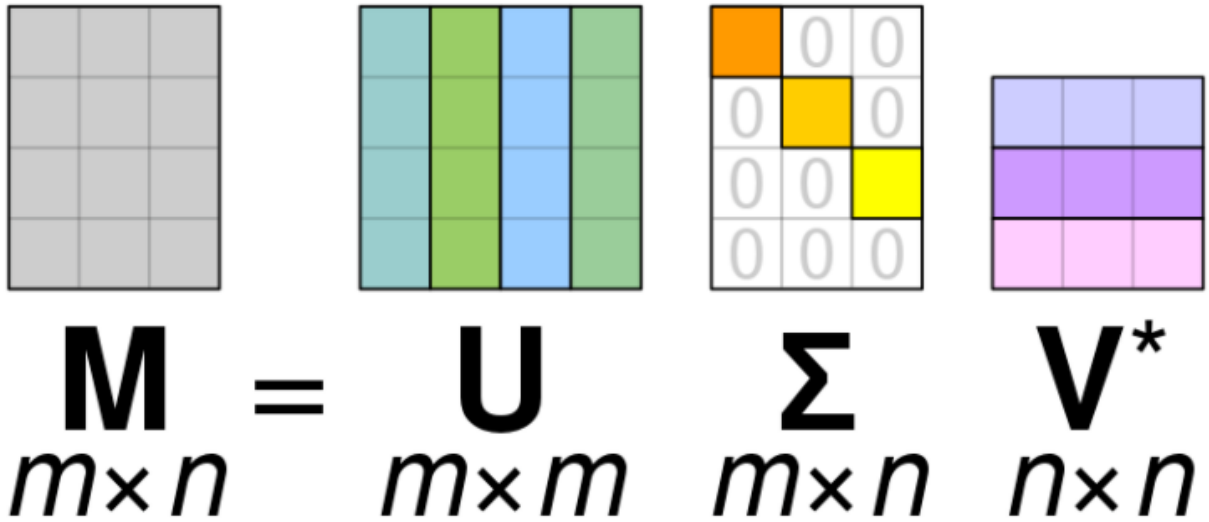


https://miro.medium.com/max/1200/1*QvhetbRjCr1vryTch_2HZQ.jpeg



https://miro.medium.com/max/1276/1*pNXo4Zj_QOZLS0a2mu5OSg.jpeg

In my implementation, I used the SVD (support vector decomposition), which results in 3 matrices (one for user, one for movies, and an attention-like matrix for weights)



https://miro.medium.com/max/1400/1*mo8loFarEKeNeVX49205-g.png

- Finally, the deep learning era where we try to learn an embedding based on implicit and explicit features to start making recommendations.

For another notebook

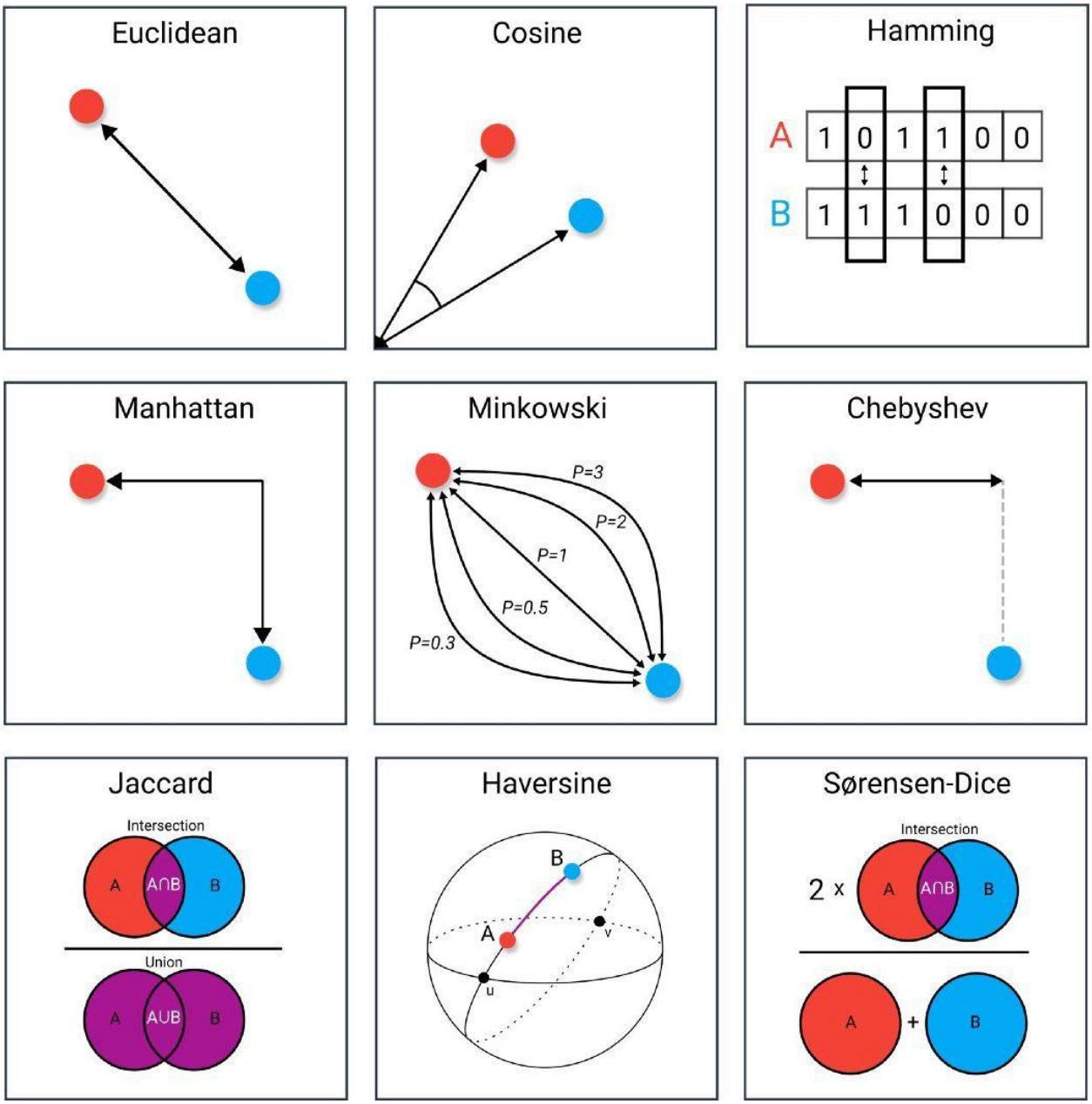
Finally, we need to discuss evaluation of our recommendation models.

Generally, we cannot use the normal precision and recall for this exercise, because for most cases we don't have the ground truth, we want to make new predictions that the user has not seen before, we should add some variety to our recommendation to avoid being boring and repetitive to the user.

For this, we use a modified version of the precision which we call precision@K. Very simply, it will make a recommendation, and if it was in the top K preferences of the user, we consider it a good model. An 80% precision@10 score means that 8 out of the 10 recommendations I made to the user were relevant to him. Similarly, we have the recall@K.

For the SVD problem, we can use RMSE between the predicted rating and the actual rating.

Finally, the only topic that remains undiscussed is how to measure similarity between two users or two items.

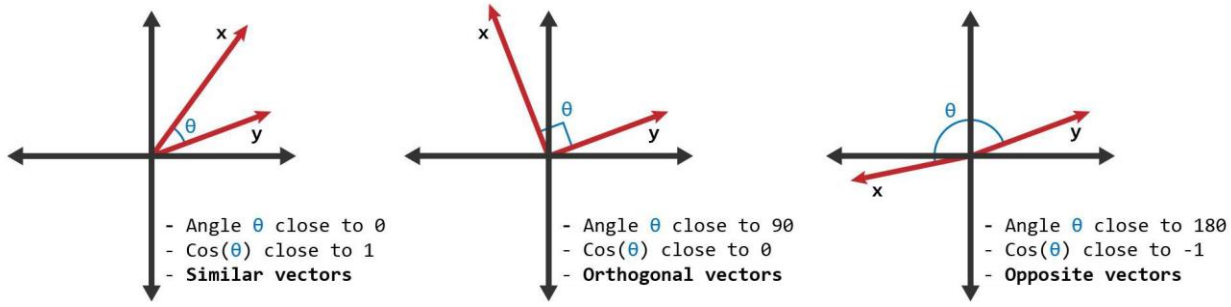


[1650370473474 (1050×1058) (licdn.com)]

In the notebook, I used Jaccard and cosine.
The jaccard similarity is simply measuring the similarity in terms of the overlap.
For example, let's say we are measuring the similarity between two movies in terms of the genres.

If movie 1 genres were: comedy, drama, romance
And movie 2 genres were: drama and mystery
Then the Jaccard similarity between them is (1 - drama) / (5) = 0.2 only.
It will be 100% if the two users had the same genres.

The cosine similarity measure the angle between two vectors, and it can generalize to any dimension.



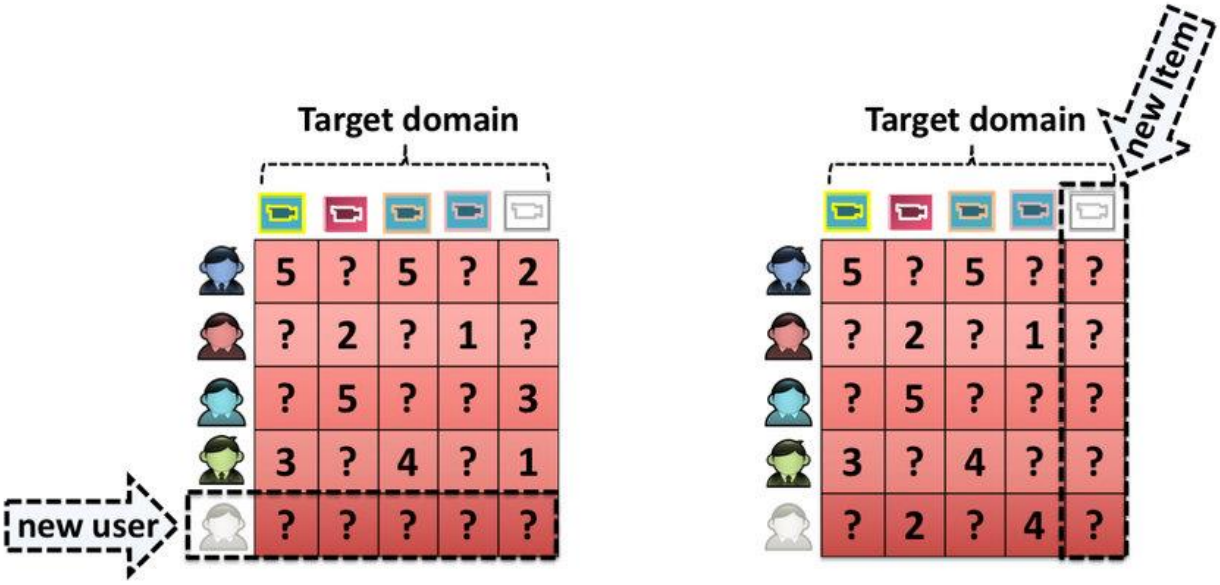
https://storage.googleapis.com/lds-media/images/cosine-similarity-vectors.original.jpg

It is very useful as it can capture the similarity between two movies or users while taking into consideration the strength of each component in the vector

For example:
If movie 1 had this vector {“drama”: 10%, Horror: “90%”}
And movie 2 {“drama”: 99%, “horror”: 1%}
The Jaccard similarity between them is 100% when they are clearly not. This is why we need the cosine distance

There are a few problems here that are related to tough raters, and easy raters, and the centered cosine need, etc.

Finally, there is a very famous problem in recommendation engines which is the cold-start problem.



https://www.researchgate.net/profile/Mehdi-Elahi-2/publication/332511384/figure/fig2/AS:749057106530306@1555600539374/Illustration-of-Cold-Start-problem-in-recommender-systems-New-user-problem-left-and.jpg

What if you had a new movie or user, you don't have any ratings to them to make your similarity measures, or any type of model. Generally, people would do an ensemble of models in this stage like popularity-based models, trending models, content-based recommendations based on the initial interactions of the new user, or Doc2Vec for analyzing the new book to get an idea about its topic to seed its vector, etc.

For the items, you can do a lot of clever methods to find the representation of it like deep learning sentiment analysis for videos, speech recognition, or tf-idf or Doc2Vec on books or movie scripts, etc. the possibilities are endless, I just wanted to introduce you to this concept.