Not always easy to find interesting content — Picture Book Museum — Credit National Geographic

How Variational Autoencoders make classical recommender systems obsolete.

Quentin Bacuet
Apr 2 · 14 min read ★

With the increase of information overload, it's getting nearly impossible to acquire exciting items through a sea of content. That's why the recommendation system is here to the rescue. Recommender systems are the models that help a user base explore new content such as music and news by showing them what they will find potentially interesting.

Here at **Snipfeed**, we deal with thousands of content every day with a demanding user base: Gen Z. By leveraging state of the art deep-learning recommender systems, we help the users navigate through their favorite videos, news, quizzes, and podcasts.(The beta version of our app is coming soon, but you can try the messenger version for a sneak peek).

As McKinsey estimates,

> *"Already, 35 percent of what consumers purchase on Amazon and 75 percent of what they watch on Netflix come from product recommendations based on such algorithms."*

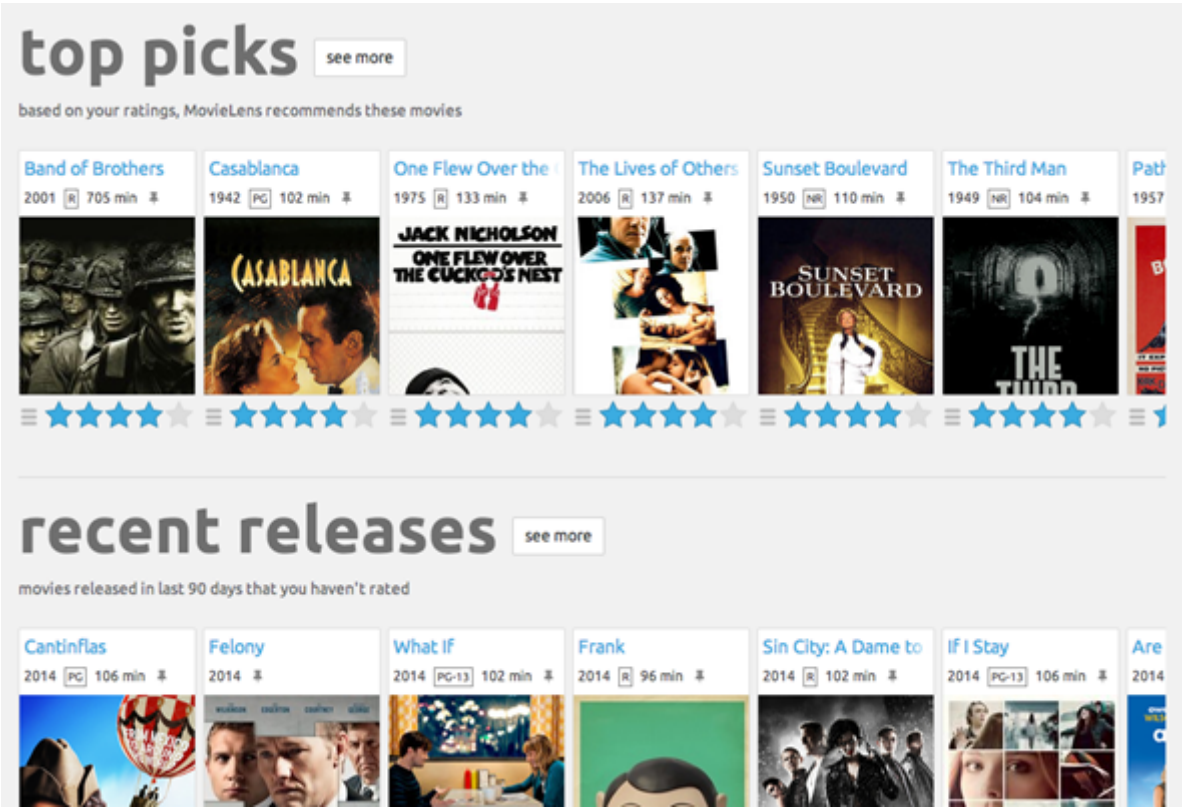
With the growing popularity of the recommendation system, the questions emerge: what new models and algorithms can bring the recommendation to a new level? How well can they perform compared with more classic methods like matrix factorization?

To answer the questions, I decided to compare nine methods and focus on two metrics: Normalized Discounted Cumulative Gain (NDCG) and the personalization index, using the infamous MovieLens dataset to conduct my experiments. I used TensorFlow and Keras to implement the models and trained them with the Google Colab Free GPU.

Dataset: MovieLens 20M

Initial Dataset

For the analysis we will use the well-known dataset MovieLens 20M.



Movielens

This dataset contains more than 20 million ratings from MovieLens, a movie recommendation service. Below, a sample of the dataframe:

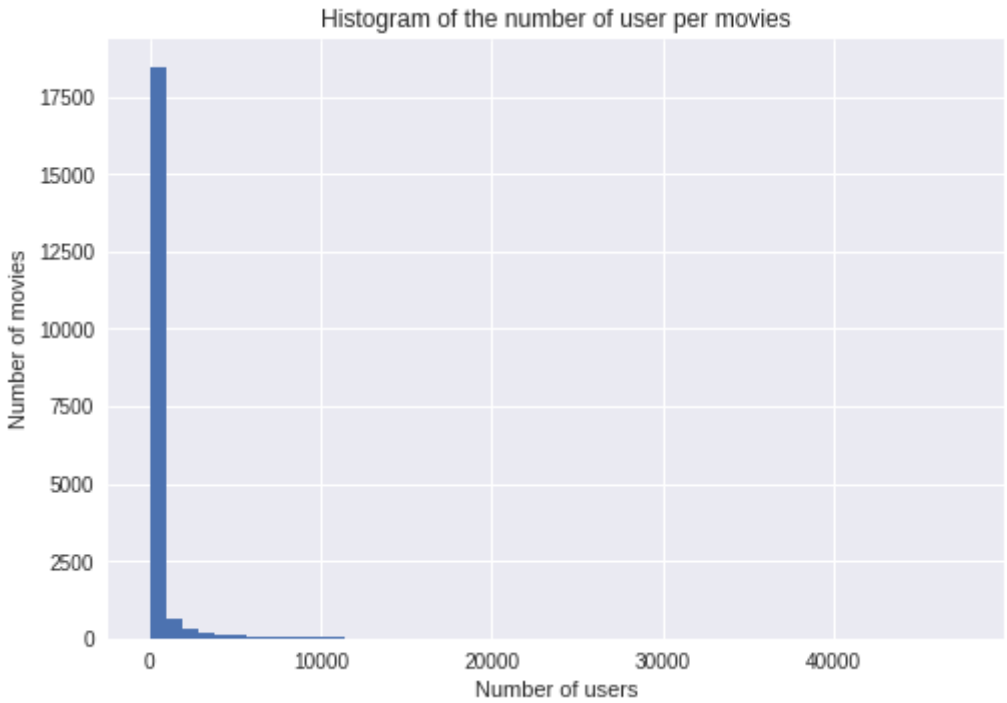
	userId	movieId	rating	timestamp
2352617	15915	2628	2.5	1205686051
810005	5393	33493	3.5	1119897675
1279964	8713	2011	4.0	945533027
9625498	66614	2881	2.0	993336001
14063276	97165	2291	4.0	985537534

Sample from the ratings file

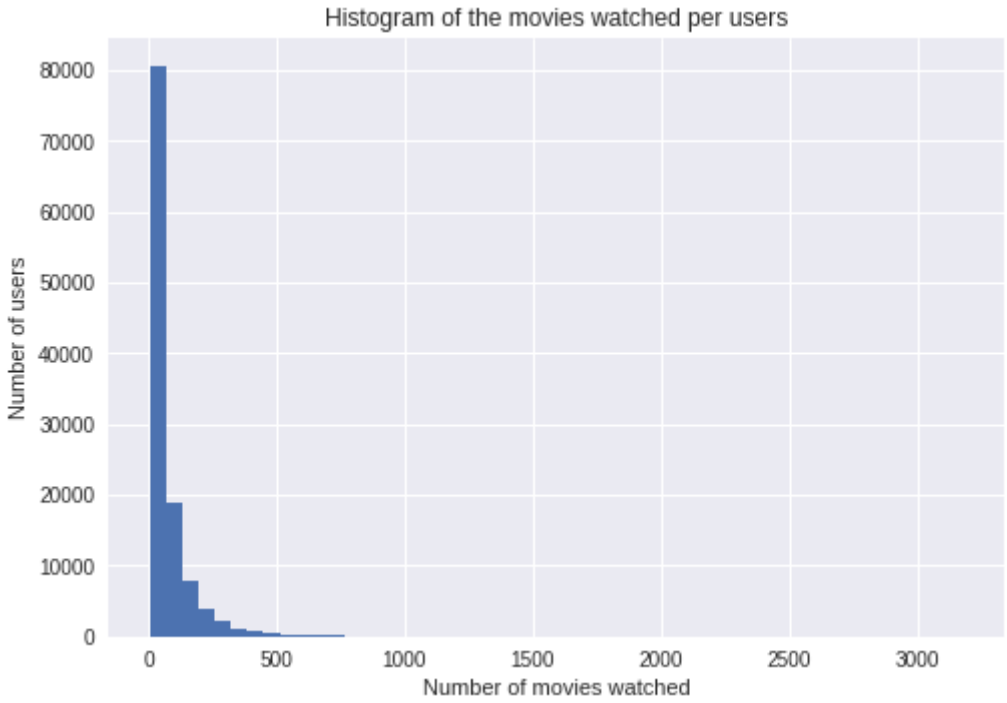
The dataset lists 138K users and over 27K movies. After cleaning and filtering (we only take the positive reviews) we have:

- 136K users
- 20K movies
- 10M interactions
- 99.64% sparsity

We can also see from the histograms below that the majority of movies have under 5,000 ratings…



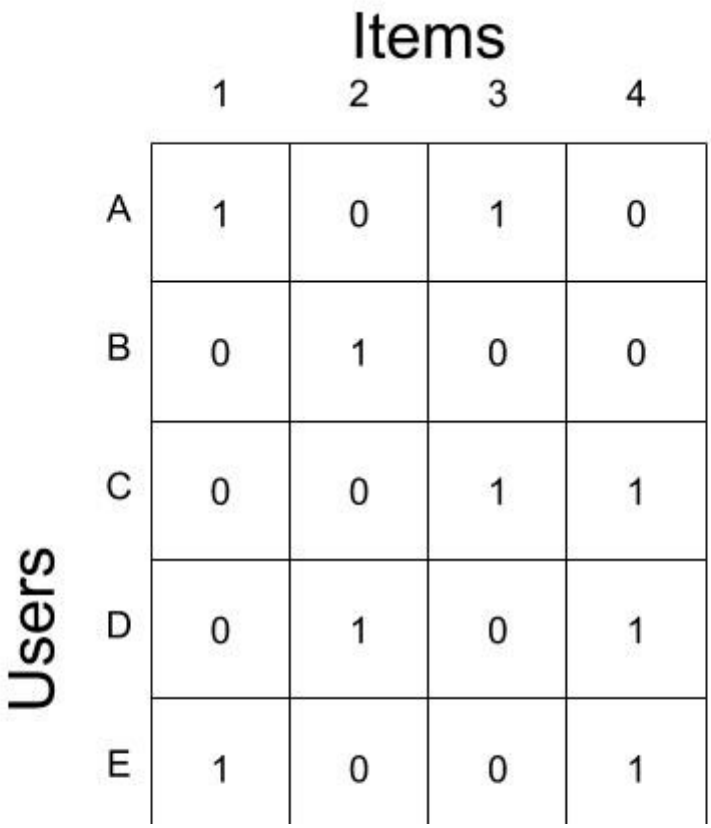
… and the majority of users have rated under 500 movies.



This is consistent with most of RS problems: very few users rated a lot of movies, and very few movies have a lot of ratings.

Training Dataset

We can build a click matrix from this data. A click matrix follows the format shown below. The cell on row *u* and column *i* contains a 1 if user *u has* interacted with item *i*, and 0 otherwise.



F	0	1	0	0
G	0	1	0	1

Example of a click matrix

We also define the click vector x_u as the vector obtained by taking the uth row of the click matrix.

Train-Validation-Test Dataset

To assess the quality of our models, we will split the dataset into 3 subsets, one for training, one for validation, one for testing. We'll use the first subset to train the model, the second to select the best model during the training, and the last one to get the final metrics.

The Metrics: NDCG and Personalization Index

NDCG

As previously stated, we'll be using two metrics to evaluate our models. The first will be the NDCG, which measures the quality and utility of the order of the items in our recommendations. We first need to define the Discounted Cumulative Gain (DCG). The higher the DCG is, the better. The DCG@p is defined as:

$$\mathrm{DCG@p} = \sum_{i=1}^{p} \frac{I(elem_i \in test)}{\log_2(i+1)}$$

DCG Formula

I being the indicator function, $elem_i$ the ith item in the ordered recommendation, and test the set of items that we wish to obtain in our recommendations (our target, if you will). To illustrate this abstract formula, here's a short example:

Recommendation expected: {A,B,C}

- Recommendation 1: [C,A,D] — DCG@3 = 1.63
- Recommendation 2: [D,B,A] — DCG@3 = 1.13

Note that the recommendations have an order. Hence we have: $DCG_1 > DCG_2$, as the first two items in our prediction 1 are items we were targeting, whereas these items are at the end of the list for our prediction 2.

The NDCG is the normalized cousin of the DCG, which means we're projecting the scores between 0 and 1 so they translate between models.

Personalization Index

The personalization index measures how unique recommendations are to a user. It computes the distance between each pair of recommendations, and then the average of that. To compare different personalization indices, we normalize them (like we did for the NDCG, we project the scores between 0 and 1). To illustrate this metric, let's look at the example below:

Recommendation 1:

- User 1: [A,B,C] / User 2: [D,E,F] — Personalization= 1

Recommendation 2:

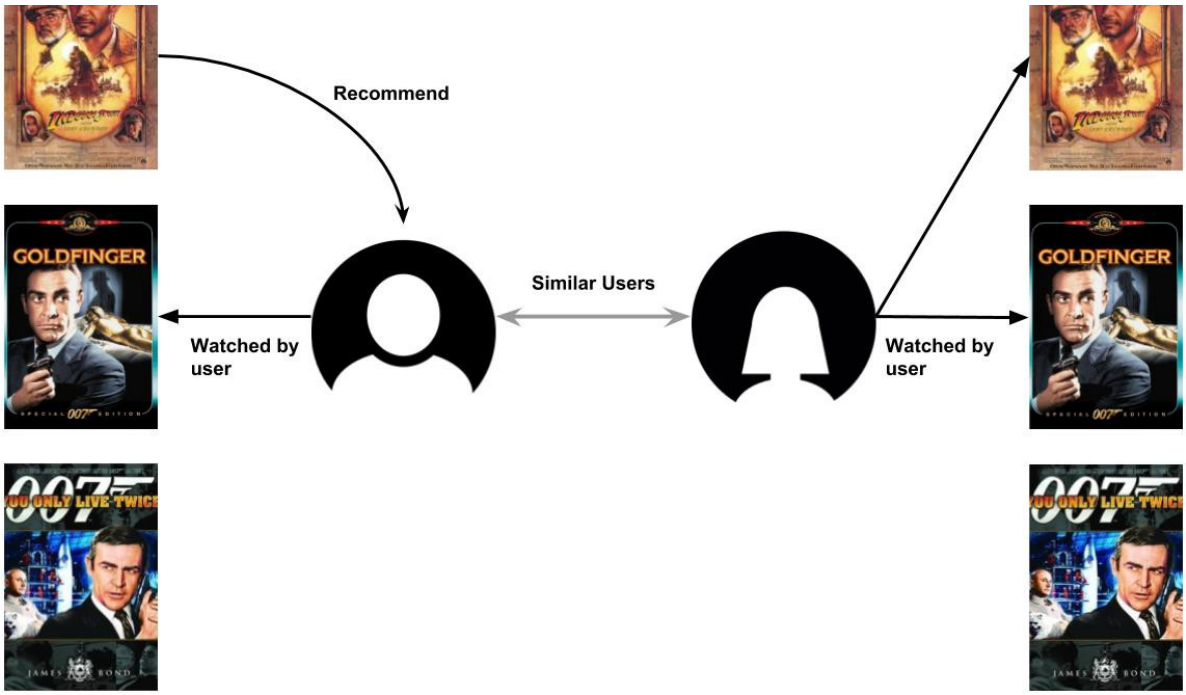
- User 1: [A,B,C] / User 2: [A,B,C] — Personalization= 0

Collaborative vs Content-based Filtering

Recommendation systems can be divided into 2 categories: collaborative, and content-based filtering.

Collaborative Filtering

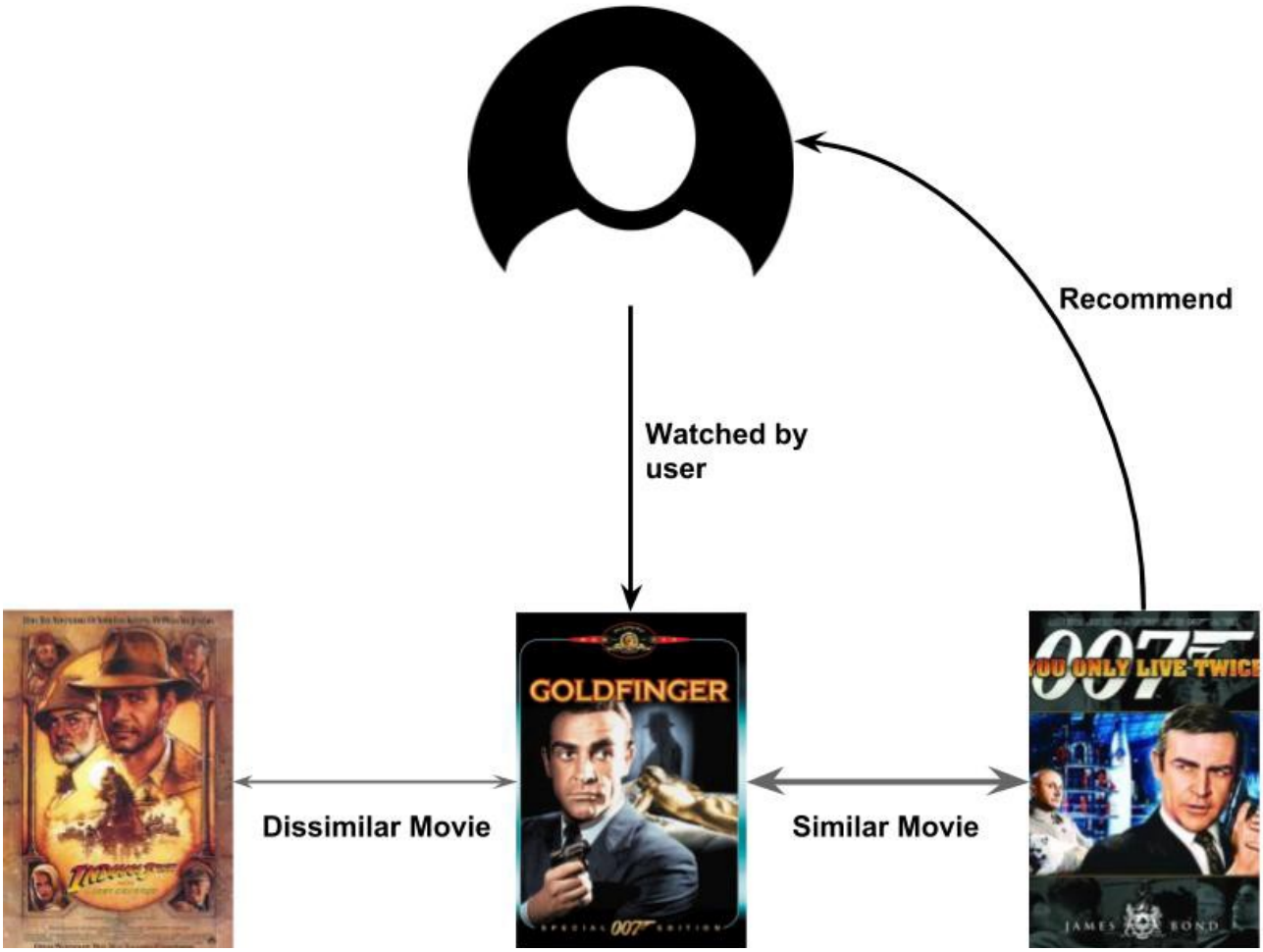
Collaborative filtering is a sub-family of RS based on user-similarity. It makes predictions on the interests of user *u* by analyzing the tastes of users which are close to *u*. It's based on the hypothesis that tightly related users are more likely to enjoy the same type of content than dissimilar users.

Collaborative filtering illustration

Content-based Filtering

Content-based filtering is another category of RS based on user preferences and content similarity, which means it's based on the idea that if you like item *i*, then you'll be more likely to like items that resemble *i* than items that are different from it.



Content-Based filtering illustration

Content Based

Definition

The content-based method, as described above, uses item descriptions to find the closest items to what a user has already seen. I implemented this method to be as exhaustive as I could, but a dataset with few features is always a limit for that method. The MovieLens dataset only provides the movies' genre.

However, we've developed a simple method, described in the pseudo-code below:

```
reco = zero-vector of size number of items

for i in items of user u:
   for j in the k closest items to i:
      reco[j] = max(reco[j],1 - dist(i,j))

output recommendation reco
```

With dist(i,j), the distance between two items i and j. I used the cosine distance between vectors of genres.

Results

- **NDCG@100:** 0.011
- **Personalization:** 0.958

The NDCG is very low, which was expected as the number of features for each item is very limited.

Advantages

- **No cold start with respect to the items:** One of the recurrent issue in RS is cold start. This issue arises when a new item or user is added, and recommendations are a bit stiff because you have no previous activity to infer from. In our scenario, the number of interactions an item has had doesn't affect its likelihood to end up in

recommendations, which means we don't have the cold start problem when it comes to items.

- **Easy implementation:** As shown above with the few lines of pseudo-code, the algorithm is fairly simple.

Disadvantages

- **Query time is O(#items × #features):** We have to be careful with scalability. Without preprocessing, each time our system is asked to recommend new content to a user, it has to find the k closest items to each of the items our user has interacted with. As there are #items to compare them to, and each distance takes #features computations to measure, this whole thing takes O(#items × #features). With pre-processing, we can kill this query time, but then we need to store the k closest items per item, which means k × #items in memory.

- **Only works if there are enough features for the items:** As seen in the results this doesn't work if there is not enough features for the items. If a description of the plot of the movies was available, we could have better results.

Memory-Based

Definition

Memory-based recommendation is a simple method to compute the similarity between users and items. As opposed to model-based methods, memory-based recommendation has no parameters to optimize. It's a very simple algorithm that can be summarized into those few lines of code:

```
input user u:

Find the k users that are the closest to u using a dist function
Aggregate the vectors of the k closest users in a new vector v_u

output recommendation v_u
```

In our case, we implemented the algorithm with:

- For the distance function we used the Hamming distance:

$$D(x, y) = \Sigma_i |x_i - y_i|$$

Hamming Distance

- For the aggregation function we used:

$$v_u = \Sigma_v \frac{1 - \text{dist}(u, v)}{\Sigma_v 1 - \text{dist}(u, v)} * x_v$$

Aggregation Function

Results

- **NDCG@100**: 0.173
- **Personalization:** 0.715

Advantages

- **Simple to implement:** As shown above with the little pseudo code, the algorithm is fairly simple, making it easy to implement.
- **Interpretability:** This is an important feature that some algorithms have. This allows to explain to the user why a specific content was recommended to them. This can have the form of: “We recommended you movie A because you saw movie B”.

Disadvantages

- **Scaling complexity:** The main problem with this method is it can make it harder to obtain something scalable. Our best friends on this are Locality Sensitive Hashing (LSH) and nearest neighbor search algorithms.
- **Query time is O(#users × #items):** Query time without pre-processing is high per user, as you need to compute #users distances with a cost of O(#items) to get the distances to all other users. Then we need to find the k closest users, which is O(#items). With pre-processing, we can kill this query time, but then we need to store the k closest users to each user, which means k × #users in memory.

Non-negative Matrix Factorization

Definition

Non-negative matrix factorization (NMF) is a well-known algorithm for recommendation systems that arose during the Netflix contest. The idea is to decompose the click matrix into two lower-dimension rectangular matrices, one for the users and one for the items, “embedded” into vectors of computable dimensions (we call that a latent space). Multiplying these two matrices back together results in a new matrix, with values close to the original click matrix where they existed, and all the gaps filled in with (hopefully) good predictions.

Results

- **NDCG@100:** 0.315
- **Personalization:** 0.800

Advantages

- **Simple to implement:** Some librairies such as Surprise or sklearn implement matrix factorization!
- **Potential interpretability:** Using some clustering and some analysis on them (find common actors, genres, ect…); it is technically possible to obtain explicable results.
- **Fast Query time:** To get the recommendation for a user we only need to multiply a vector and a matrix.

Disadvantage

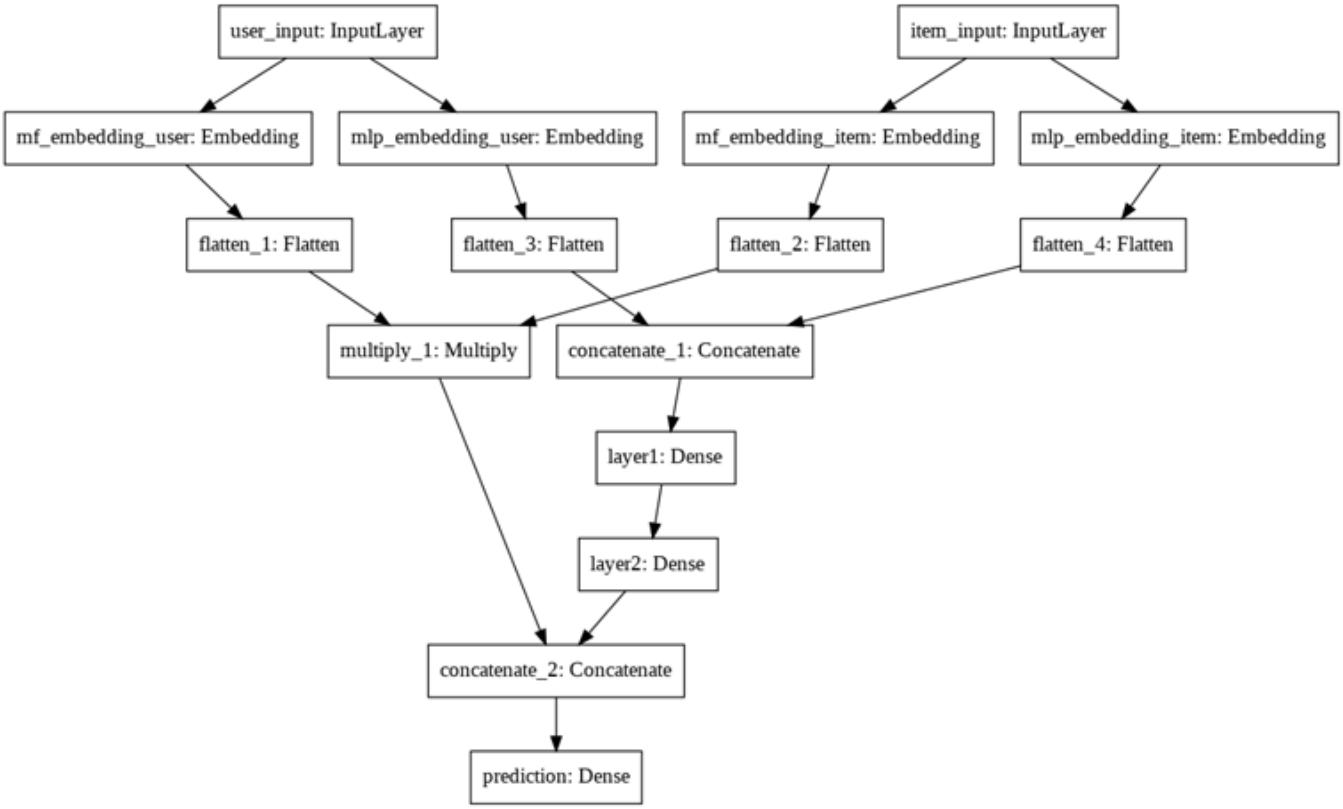
- **Linear Model:** One of the main limitations of the matrix factorization is that it is a linear model, hence it cannot capture more complex relations in the data. Even though it is linear, we see that it give good results in terms of NDCG.

Neural Matrix Factorization

Definition

Neural Matrix Factorization (NeuMF) is a new approach that attempts to generalize the classic NMF seen above. It was developed in this paper. The model takes two integers (two indices) as inputs representing the item i and user u, and output a number between 0 and 1. The output represents the probability that the user u will be interested in item i. The architecture of the Neural Net(NN) can be split into two parts: the Matrix

Factorization part and the fully-connected part. These parts are concatenated, and then passed on to a sigmoid layer.

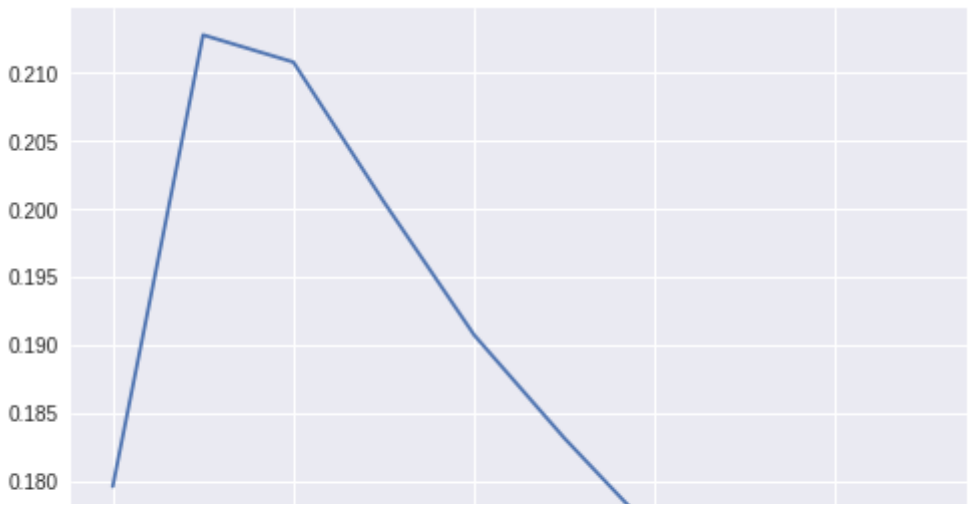


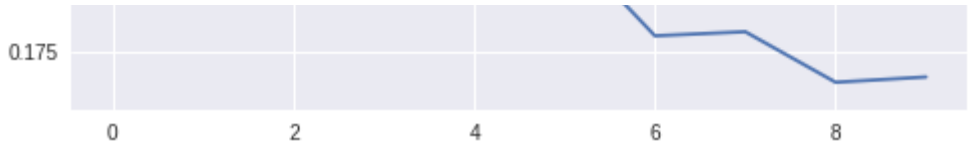
NeuMF architecture

Results

- **NDCG@100:** 0.173
- **Personalization:** 0.017

Below, we added the learning curve of the validation set of the NDCG@100 against the epochs. Even though I tried to regularize with lots of different parameters, over-fitting was unavoidable.





NDCG@100 against epochs on the validation set

Advantage

- **Neural net (non-linear model):** One of the main advantages of the NeuMF is that it is a non-linear model, so it can capture more complex patterns in the data. However, we can see our NDCG is lower here than it was with regular NMF.

Disadvantages

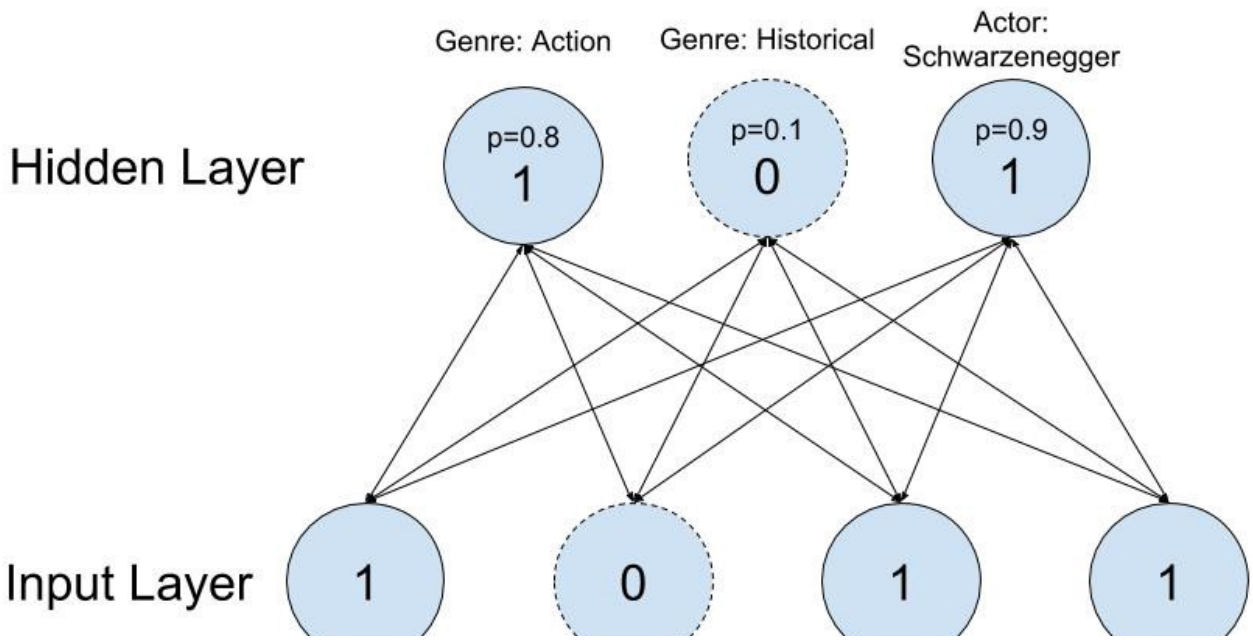
- **Overfitting for big datasets:** In the original paper, the NeuMF improves the NMF model, but it is for a smaller dataset. We can deduce that for bigger datasets this method tends to overfit.

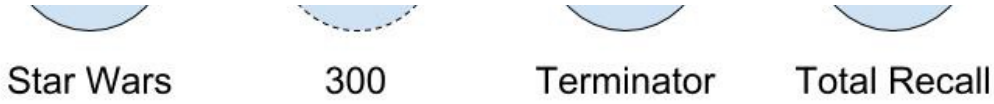
- **Query time is O(#items):** One of the problems with this method is that, for a given user, we need to parse all the items. This can become a scalability problem when the number of items increases.

Restricted Boltzmann machine

Definition

Restricted Boltzmann machines (RBM) are a generative stochastic artificial neural network with a very simple architecture (one input layer and one hidden layer) that can be used to learn a probability distribution over the inputs, in our case the click vectors.



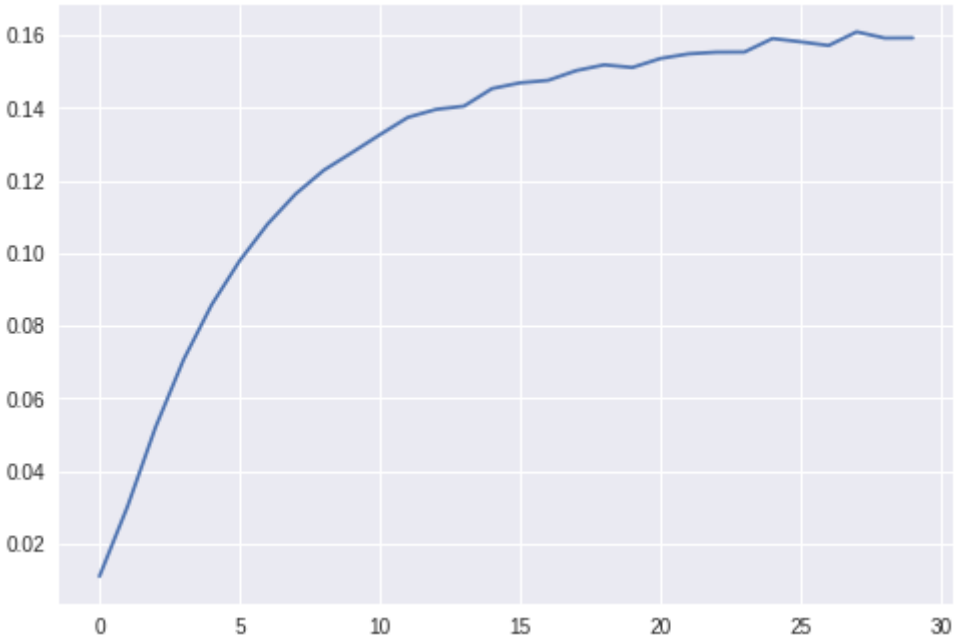


Example of an RBM

Results

- **NDCG@100:** 0.155
- **Personalization:** 0.959

Below, we added the learning curve of the validation set of the NDCG@100 along the epochs.



NDCG@100 against epochs on the validation set

Advantages

- **Neural net (non-linear model):** As the RBM is a NN, it is a non-linear model, so it can capture more complex patterns in the data.
- **Potential interpretability:** The RBM learns complex features from the data that are represented by the hidden layer. By doing some analysis (in terms of genre, actors) one could technically manage to explain the results.

Disadvantages

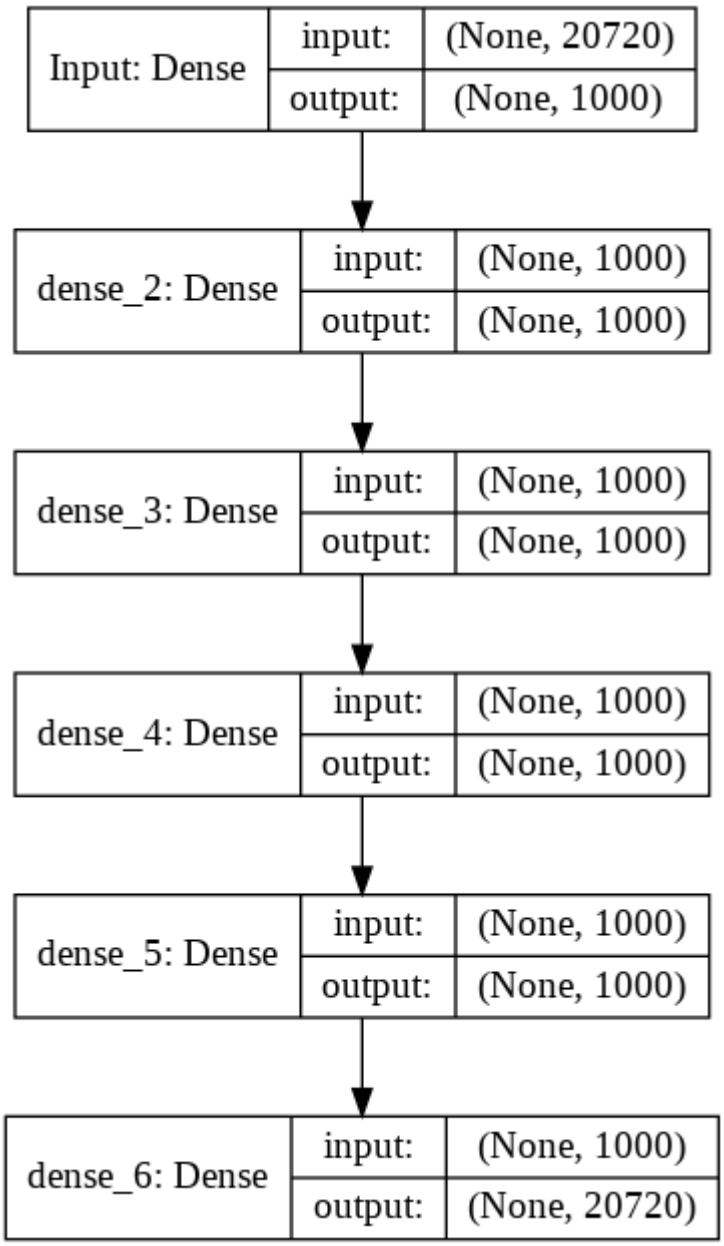
- **Long to train:** The training of this model revolves around a method called Gibbs sampling. This method implies a lot of sampling and this is computationally

intensive.

Deep Collaborative

Definition

Deep collaborative is a straight-forward collaborative model that aims to predict the most useful items for an user. The input is a user's click vector, and the raw output is our recommendation. To train this model, I used 70% of the user's click vector as input (the 30% left is replaced with 0s), and the rest as output.



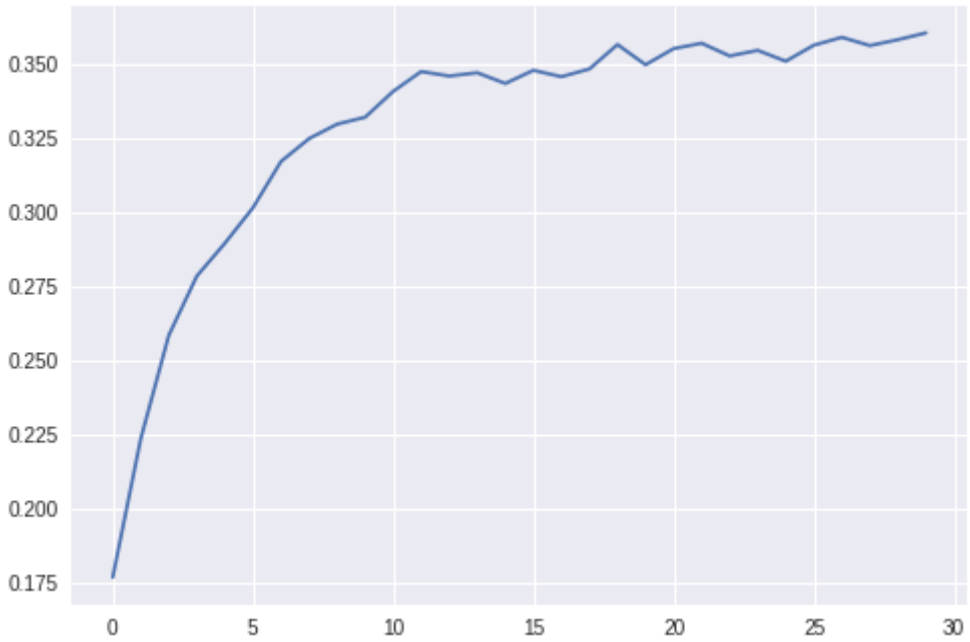
Deep Collaborative architecture

The architecture is simple. There's an input and an output of the same size (#items), and multiple hidden layers of the same size (1000 neurons).

Results

- **NDCG@100:** 0.353
- **Personalization:** 0.087

Below, same as usual (NDCG@100 on the validation set during learning):



NDCG@100 against epochs on the validation set

Advantages

- **Neural net (non-linear model):** Deep Collaborative is a non-linear model, so it can capture more complex patterns in the data.
- **Fast query time:** The main advantage of this model is that in one forward pass, we can obtain the recommendation for a given user, which gives us a short query time. One can see that the number of parameters of the model increases with the number of the items, but even with this it remains faster than NeuMF.

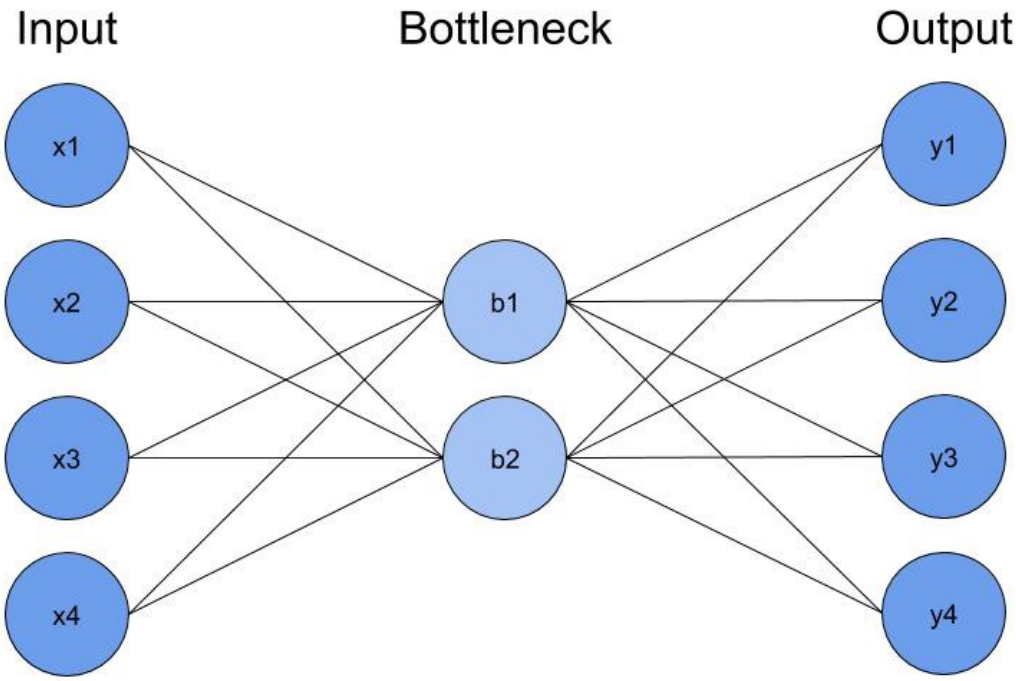
Disadvantages

- **No interpretability:** This kind of deep NN makes it impossible to explain the results.

Autoencoder

Definition

Autoencoders (AE) were initially used to learn a representation of the data (encoding). They decompose into two parts: the encoder, which reduces the shape of the data with a bottleneck, and the decoder, that transforms the encoding back into its original form. As there is a dimension reduction, the NN will need to learn a representation in lower dimension of the input (the latent space) to be able the reconstruct the input. In the context of RS, they can be used to predict new recommendation. To do so, the input and the output are both the click vector (it is usual for AE that the input and the output are the same) and we will use dropout after the input layer. This means that the model will have to reconstruct the click vector as some element from the input will be missing, hence learning to predict the recommendation for a given click vector.

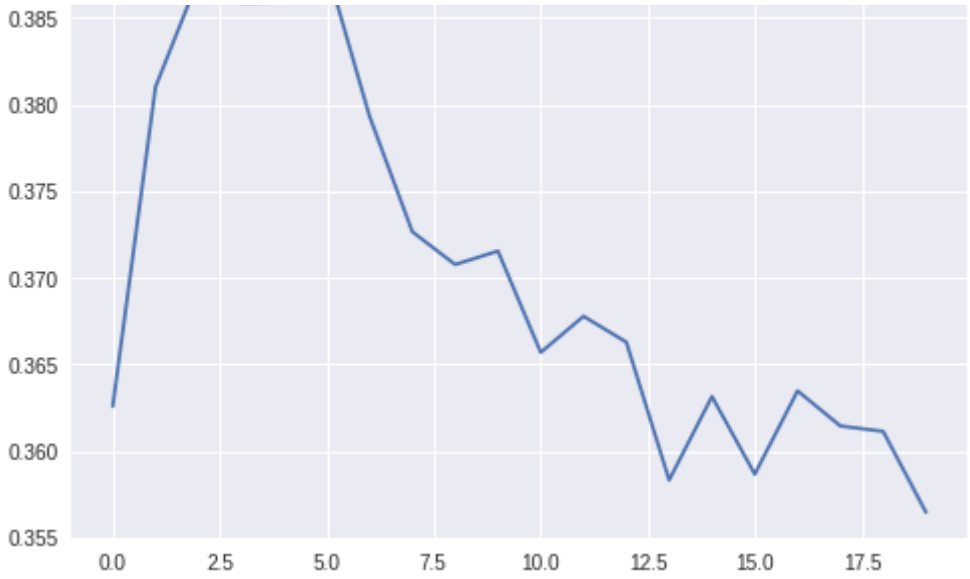


AE architecture

Results

- **NDCG@100:** 0.382
- **Personalization:** 0.154

Below, we added the learning curve of the validation set of the NDCG@100 along the epochs. Even though we tried to regularize with a lot of different parameters, it quickly overfit.



NDCG@100 against epochs on the validation set

Advantages

- **Neural net (non-linear model):** The AE is a non-linear model, which means it can capture more complex patterns in the data.
- **Fast query time:** One forward pass is sufficient to get the recommendation for a given user. Which means the query time is fast.

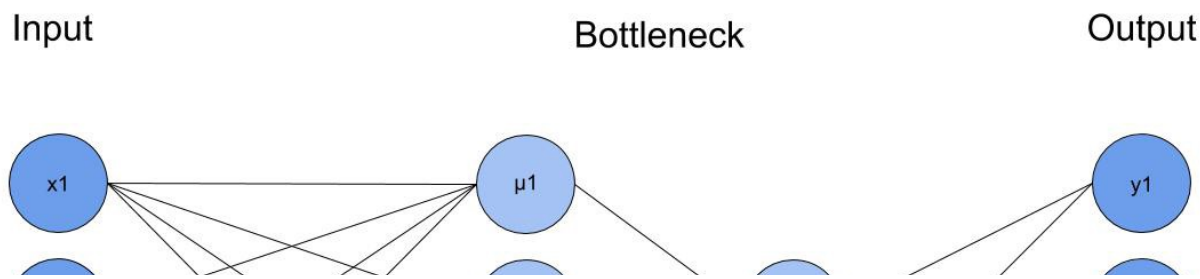
Disadvantages

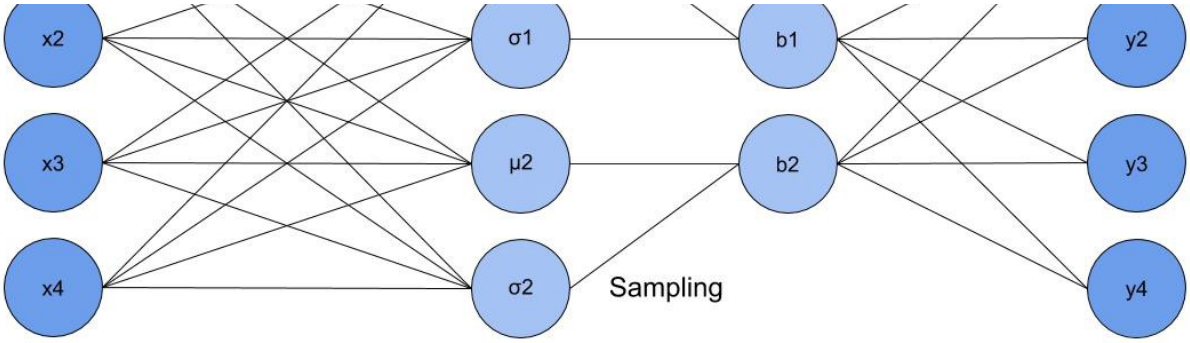
- **No interpretability:** This kind of deep NN makes it impossible to explain the results.

Variational Autoencoder

Definition

Variational Autoencoders (VAE) are an extension of AE. Instead of having a simple dense layer for the bottleneck, it will have a sampling layer. This layer will use the mean and variance from the last layer of the encoder to get a Gaussian sample and use it as input for the decoder. Same for the AE we use dropout for the first layer.



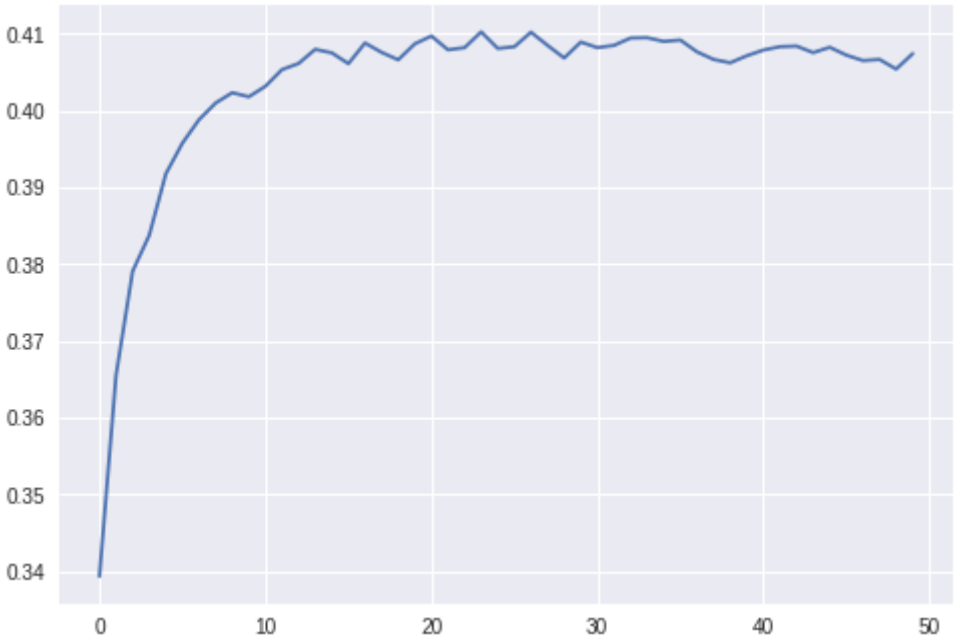


VAE architecture

Results

- **NDCG@100:** 0.403
- **Personalization:** 0.117

Below, that same old NDCG@100 on the validation set during learning.



NDCG@100 against epochs on the validation set

Advantages

- **Neural net (non-linear model):** The VAE is a non-linear model, so it can capture more complex patterns in the data.
- **Fast query time:** One forward pass is sufficient to get the recommendation for a given user. Hence the query time is fast.

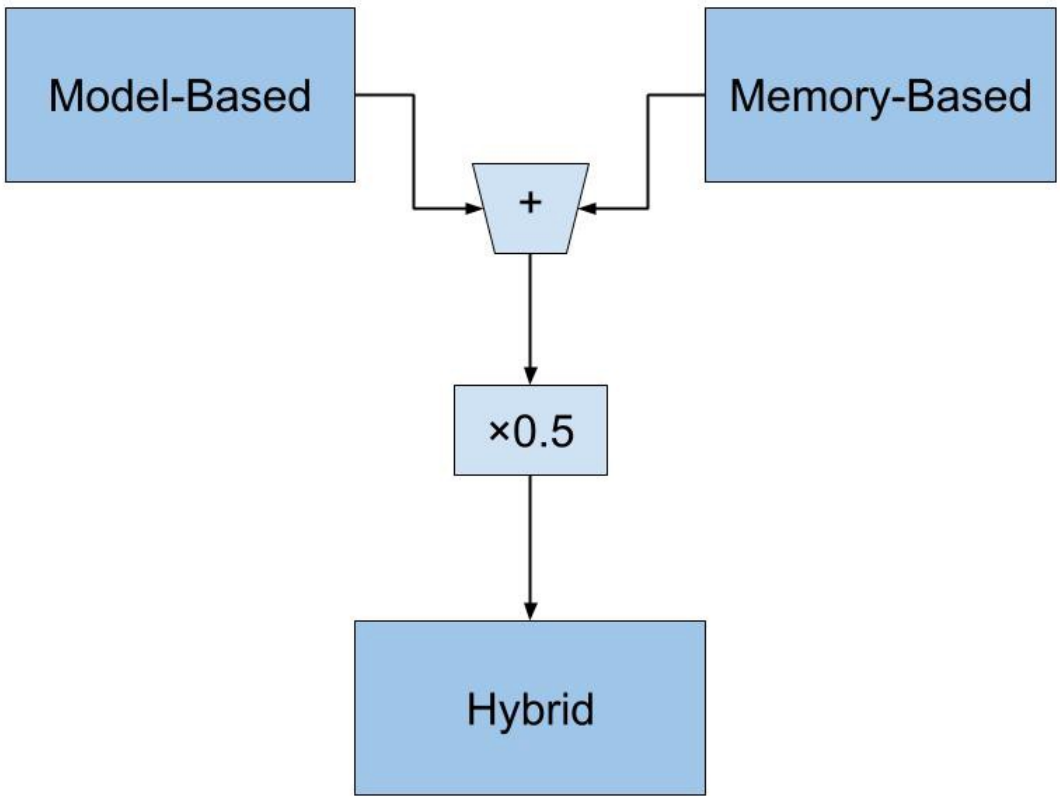
Disadvantages

- **More complex to implement:** the sampling layer makes it difficult to compute a gradient descent with back-propagation. The reparameterization trick makes it possible to overcome this problem by using the equation $z = \varepsilon \times \sigma + \mu$, with $\varepsilon \sim N(0,1)$. We can now compute the gradient safely.
- **Non-explicable results:** This kind of deep NN makes it not feasible to explain the results.

Hybrid

Definition

Hybrid models offer the best of both worlds (memory-based and model-based approaches), and are therefore very popular in RS.



Hybrid architecture

To implement the hybrid method, I chose to use a VAE and then average its results with memory-based results.

Results

- **NDCG@100:** 0.334
- **Personalization:** 0.561

Advantages

- **Part of it is a NN:** As a part of the method is a VAE, it can capture more complex patterns in the data.
- **Interpretability:** As a part of the method is Memory-based, we get the interesting property that we can explain to the user why we recommended them a specific item.

Disadvantages

- **Query time is O(#users × #items):** The bottleneck in terms of computational time is the memory-based part. As shown above the query time for it is O(#users × #items) without preprocessing.

Comparison

We can now compare all our models. The best model when looking at the NDCG@100 is the VAE. For the personalization index, it's the RBM.

Modèle	Memory Based	NMF	NeuMF	VAE	AE	RBM	Content Based	Deep Colla	Hybrid Model
NDCG	0.173	0.315	0.173	**0.403**	0.382	0.155	0.011	0.353	0.334
Perso	0.715	0.800	0.017	0.117	0.154	**0.959**	0.958	0.087	0.561

Conclusion

New methods like VAE, AE, or Deep Collaborative outperform classical methods like NMF on the NDCG metric. Non-linear probabilistic models such as variational autoencoders enable us to go beyond the limited modeling capacity of linear factor models.

In my next post, I will do a deep dive into the VAE implementation for recommender systems with code and illustrations, so stay tuned !

References

1. Xiangnan He, Lizi Liao, Hanwang Zhang, Liqiang Nie, Xia Hu, Tat-Seng Chua, Neural Collaborative Filtering, 2017

2. Dawen Liang, Rahul G. Krishnan, Matthew D. Hoffman, Tony Jebara,Variational autoencoders for collaborative filtering, 2018.

Thanks to Pierre-Habté Nouvellon, Rédouane Ramdani, and Chloé Lagrue.

Machine Learning Recommendation System Variational Autoencoder Collaborative Filtering

AI

About Help Legal