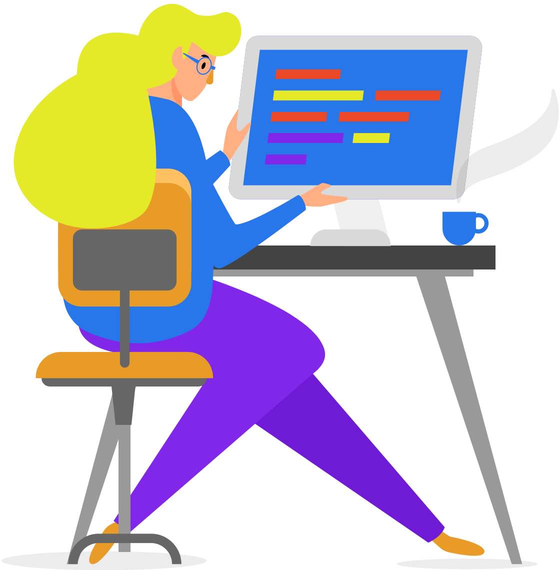
Recommendation systems

This is what happens when you use my engine to get music recommendations

You have been warned

A case study on the movielens dataset

Recommendation systems



01 **Introduction**
What are recommendation systems

02 **Popularity-based models**

03 **Content-based model**

04 **Collaborative-based models**

05 **Deep Learning models**

06 **conclusion**

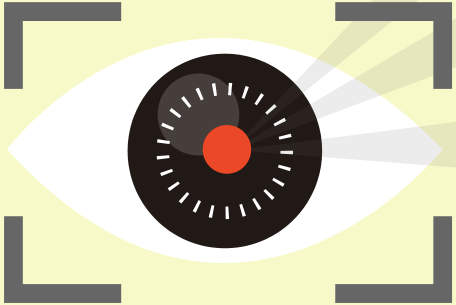
Introduction

Recommendation system for movielens

- We are given a huge dataset on multiple files that contains the ratings of 280,000 users to over 58,000 movies. We have different around 1128 tag per movie with relevance scores for each tag for each movie
- We want to make good recommendations to users for them to watch new movies

Popularity-based model

Recommend the most popular movies in the data



One-size fits all

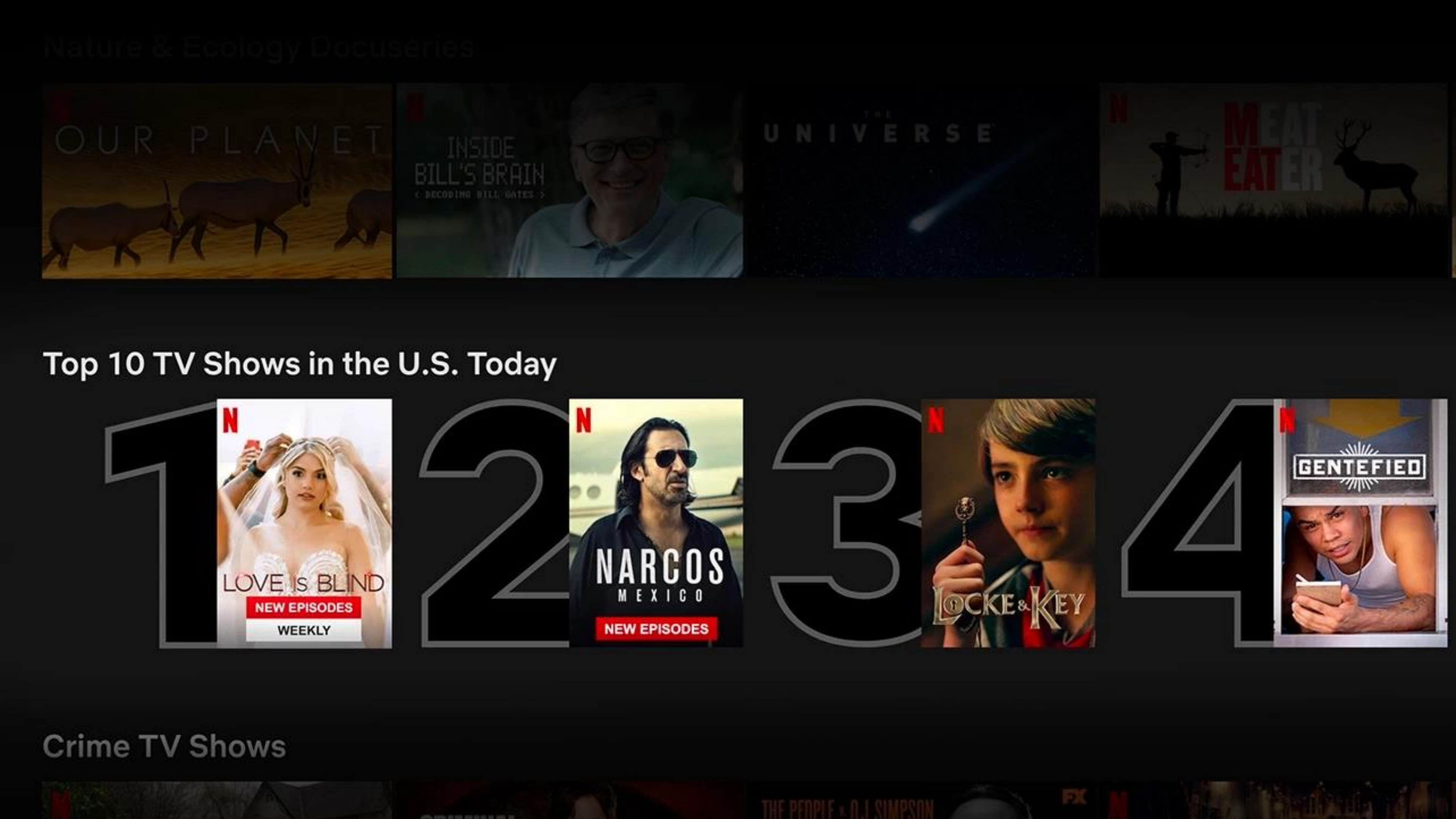
The system is not personalized at all

High-probability of being already watched

There is a big probability that the user has already watched the most popular movie in the dataset

Evaluate with P@K

Achieves around 14% p@15

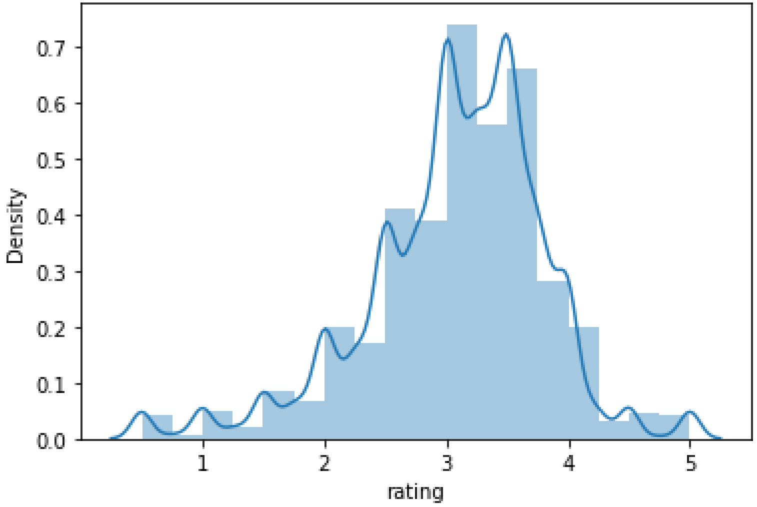

Nature & Ecology Docuseries
OUR PLANET
INSIDE BILL'S BRAIN
DECODING BILL GATES
THE UNIVERSE
MEAT EATER
Top 10 TV Shows in the U.S. Today
LOVE IS BLIND
NEW EPISODES
WEEKLY
NARCOS
MEXICO
NEW EPISODES
LOCKE & KEY
GENTEFIED
Crime TV Shows

In our data, these are the top 10 popular movies

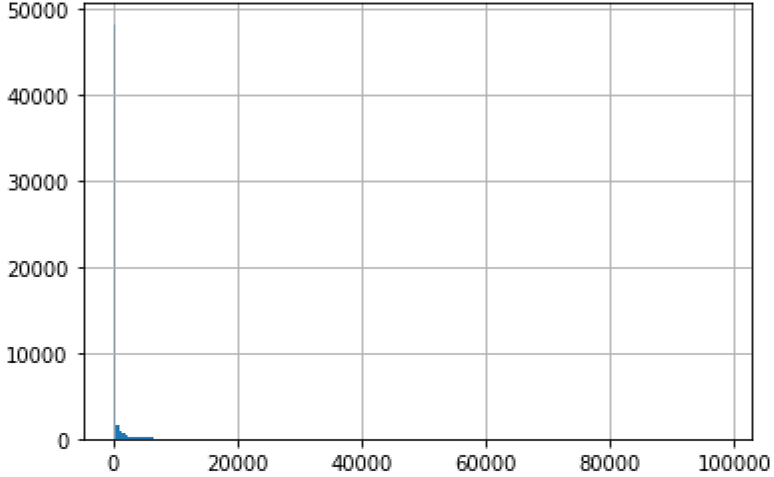
```
top10Watched.reset_index(inplace=True)
for index, row in top10Watched.iterrows():
    print(getMovieTitle(moviesDf, 'movieId', row))
```

```
Shawshank Redemption, The
Forrest Gump
Pulp Fiction
Silence of the Lambs, The
Matrix, The
Star Wars: Episode IV - A New Hope
Jurassic Park
Schindler's List
Braveheart
Toy Story
```

Normal distribution for ratings with most movies getting rated higher than the average rating



The thin blue line on the left shows the long-tail plot that most movies will probably end getting very low views, and very few movies would get very high views



With this simple policy, we got around 538577 out of the 3.76 million top 15 preferences correctly.
This is 14.3% precision at 15.

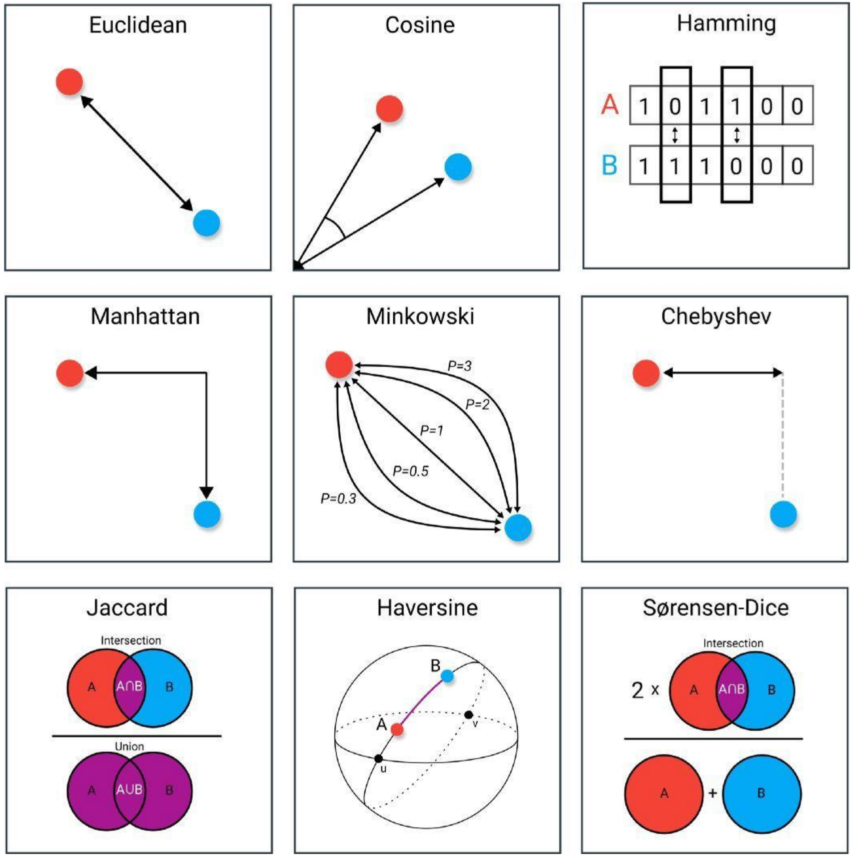
```
print("We made ", counter, " correct recommendations out of the ", top15Copy.shape[0], " we had to make")

print("Our precision when recommending 15 movies @ 15 top likes of each user is: ", counter/top15Copy.shape[0])
```

```
We made  538577  correct recommendations out of the  3768097  we had to make
Our precision when recommending 15 movies @ 15 top likes of each user is:  0.14293076850197858
```

Distances measures

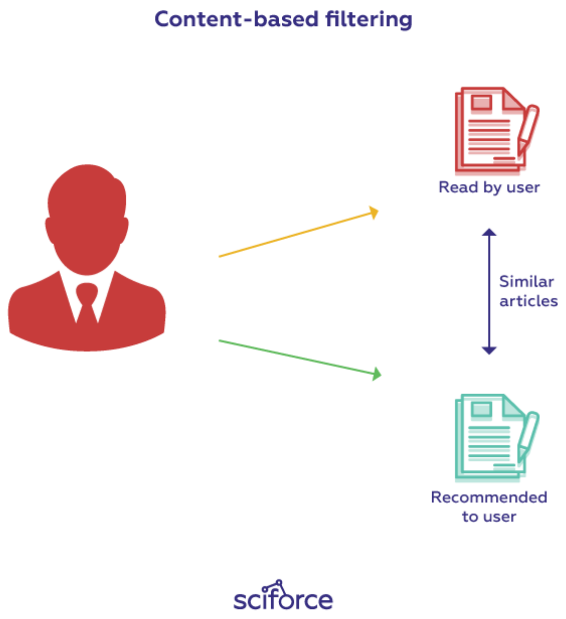
In the notebook, I used Jaccard similarity and cosine similarity to measure similarity between users and movies

Check the notebook for this, it has some really nice examples and implementation tricks



Content-based recommendation system

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.



On our dataset

```
Getting recommendations for:  Toy Story
['Toy Story',
 'Monsters, Inc.',
 'Toy Story 2',
 "Bug's Life, A",
 'Finding Nemo',
 'Ratatouille',
 'Toy Story 3',
 'For the Birds',
 'Toy Story That Time Forgot',
 'Winnie the Pooh and Tigger Too']
```

We get some really nice recommendations for Toy story 1, including the two other toy story movies, nemo, bug's life, etc

Even better with Batman

```
getRecommendations(10, 58559)
```

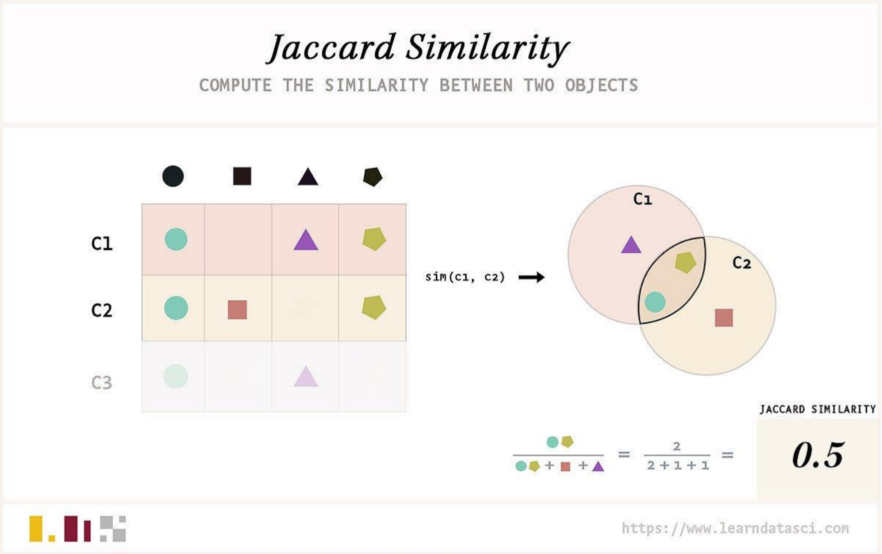
```
Getting recommendations for:  Dark Knight, The
['Dark Knight, The',
 'Batman Begins',
 'Dark Knight Rises, The',
 'Batman: The Dark Knight Returns, Part 2',
 'Batman',
 'Batman Beyond: Return of the Joker',
 'Superman/Batman: Apocalypse',
 'Batman: Mystery of the Batwoman',
 'Batman: Under the Red Hood',
 'Justice League: Crisis on Two Earths']
```

That was based on the top 60 tags similarity using Jaccard distance

Each movie had 1128 tags associated with it, each tag has a relevance score strength

I ranked the top 60 tags per movie, and measured the jaccard similarity between all movies and got the highest scores to be our content-based recommendations

It works really well!



TF-IDF on Genres

Our dataset had 21 genres in total, each movie could have any possible combination of those genres. I used the tf-idf score that is commonly used to measure document similarity to measure movies similarities based on their tf-idf scores of the genres

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right)$$

TF-IDF

Term x within document y

$tf_{x,y}$ = frequency of x in y

df_x = number of documents containing x

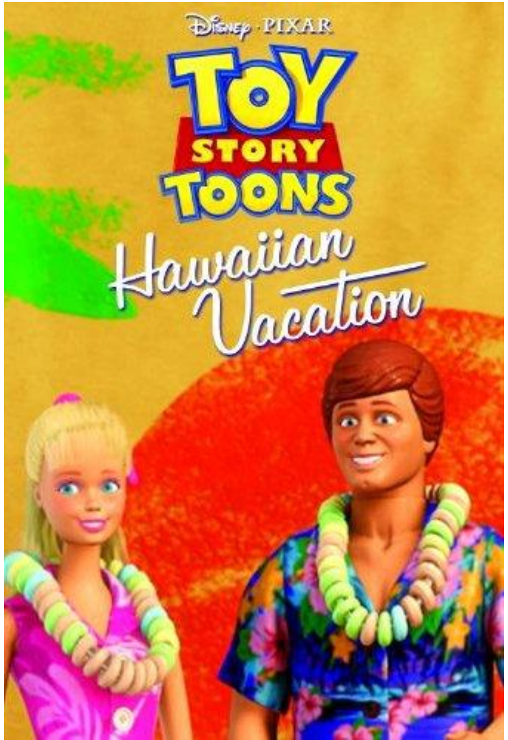
N = total number of documents

This time, we had continuous scores, so I used cosine similarity

It recommended two toy story short movies that I never heard of, and other famous cartoon movies like Moana and the good dinosaur. This metric alone is not perfect, it should be combined with the previous tags score

```
get_top_k_cosinSim(movieId=1, tfidf_matrix=tfidf_matrix, k=15)
```

```
Toy Story
Toy Story Toons: Hawaiian Vacation
Toy Story Toons: Small Fry
The Magic Crystal
Asterix and the Vikings (Astérix et les Vikings)
Puss in Book: Trapped in an Epic Tale
Moana
Tale of Despereaux, The
Brother Bear 2
The Dragon Spell
Tangled: Before Ever After
Boxtrolls, The
Aladdin
The Good Dinosaur
Adventures of Rocky and Bullwinkle, The
```

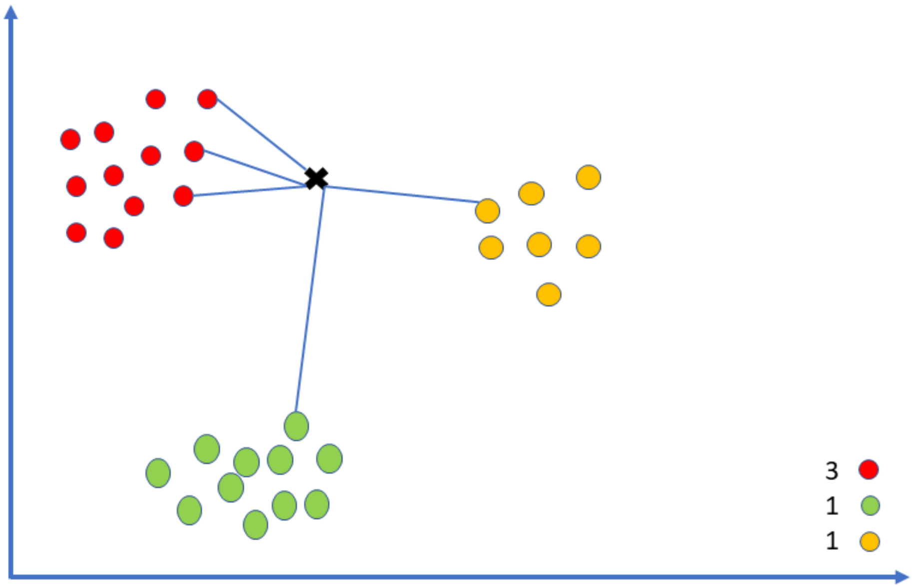


Modeling it as a KNN problem

```
#knn classifier for movie recommendation
from sklearn.neighbors import NearestNeighbors

knn = NearestNeighbors(n_neighbors=11, algorithm='auto', metric='cosine')
x = movieTagsDictDf.iloc[:,0:].values
knn.fit(x)
```

```
ids = movieTagsDictDf.iloc[toyStoryNeighbors[0]].index.values
for id in ids:
    print(moviesDf[moviesDf['movieId'] == id]['title'].values[0])
```

```
Toy Story
Monsters, Inc.
Toy Story 2
Bug's Life, A
Toy Story 3
Finding Nemo
Ratatouille
Incredibles, The
Up
Ice Age
Shrek
```



This time, I used the cosine distance between the value associated with each tag of all 1128 tags

```
[ ] movieTagsDictDf.head()
```

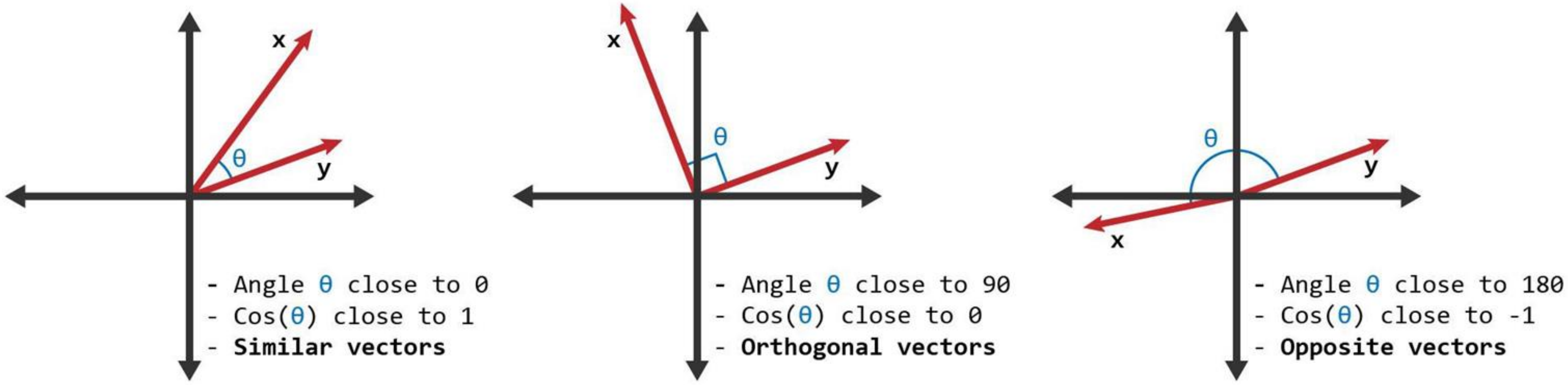
movieId	tag_007	tag_007 (series)	tag_18th century	tag_1920s	tag_1930s	tag_1950s	tag_1960s	tag_1970s	tag_1980s	tag_19th century	...
1	0.02900	0.02375	0.05425	0.06875	0.16000	0.19525	0.07600	0.25200	0.22750	0.02400	...
2	0.03625	0.03625	0.08275	0.08175	0.10200	0.06900	0.05775	0.10100	0.08225	0.05250	...
3	0.04150	0.04950	0.03000	0.09525	0.04525	0.05925	0.04000	0.14150			
4	0.03350	0.03675	0.04275	0.02625	0.05250	0.03025	0.02425	0.07475			
5	0.04050	0.05175	0.03600	0.04625	0.05500	0.08000	0.02150	0.07375			

5 rows × 1128 columns

Refresher on cosine similarity

movie 1 {“drama”: 10%, Horror: “90%”}
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

It can generalize to any dimensions, the 1128 dimensions is similar to the 3 dimensions. It is math that is hard to imagine



I tested it with the Godfather and it was just perfect

```python
godfather = movieTagsDictDfvery.iloc[777,:]
godfatherNeighbors = knn.kneighbors([godfather], return_distance=False)
ids = movieTagsDictDf.iloc[godfatherNeighbors[0]].index.values
for id in ids:
    print(moviesDf[moviesDf['movieId'] == id]['title'].values[0])
```

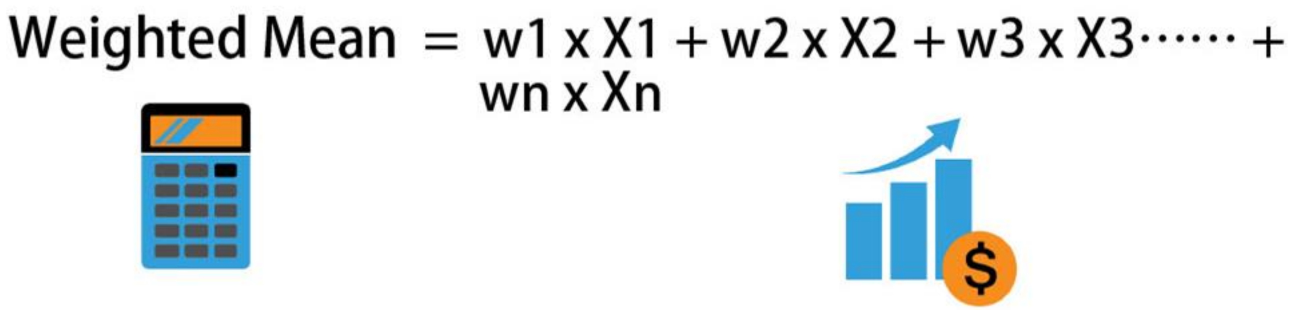
```
Godfather, The
Godfather: Part II, The
The Godfather Trilogy: 1972-1990
Goodfellas
Departed, The
On the Waterfront
Road to Perdition
Scarface
Untouchables, The
Unforgiven
Good, the Bad and the Ugly, The (Buono, il brutto, il cattivo, Il)
```

The problem is: Users are not movies.

How do you make recommendations to users if you only know how to measure similarity between movies?

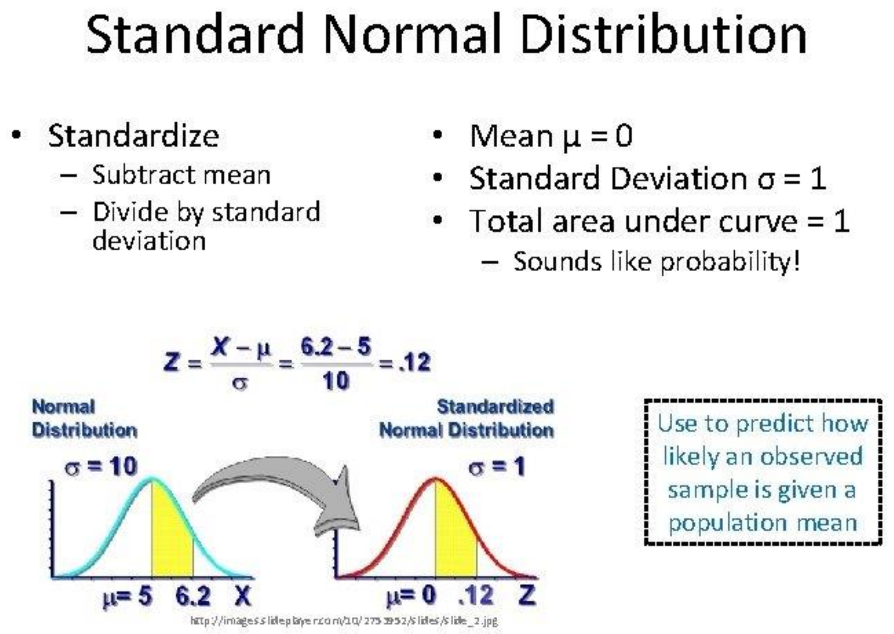
I calculated a weighted-mean for each user based on how he rated each movie times the movie vector.

Now if he really dislikes cartoon, we multiply his rating to each cartoon movie by its vector and them all.



Standardize first so low rating become negative and high ratings becomes positive

It gave much better recommendations



	userId	movieId	rating	avg	std	rating_n
14237562	145719	1270	4.0	3.990642	0.666545	0.014040
18664753	190349	1961	4.0	3.333333	1.154701	0.577350
19097314	194856	8961	5.0	3.750000	1.163090	1.074724
24256100	248042	6890	5.0	3.710177	1.134835	1.136573
40199	384	51540	4.0	3.162338	1.301145	0.643789
15296924	156277	3633	5.0	3.589984	1.003580	1.404986
13843550	141682	4306	5.0	4.637255	0.459113	0.790101
20590309	210228	168	3.0	2.927798	1.050459	0.068734
24883036	254219	1036	4.0	3.340637	1.071078	0.615607
1992358	20399	688	4.0	1.888545	0.912283	2.314475

Watch and learn how it is done 😌

```
#you can see that user 5 watched in general 72 movies
print("User 5 watched: " , len(ratingsDf[ratingsDf['userId'] == 5]))
#print intersection of liked and not liked movies
print("and out of the 10 recommendation we made, he would have liked: " , len(set(u5Liked).intersection(u5_recommendations)))
print(set(u5Liked).intersection(u5_recommendations))
```

```
User 5 watched:  72
and out of the 10 recommendation we made, he would have liked:  10
{'Goodfellas', 'Memento', 'Fight Club', 'Eternal Sunshine of the Spotless Mind', 'City of God (Cidade de Deus)', 'American Beau
```

10/10 ! perfect for this user

The problem is that this is very very very heavy computationally. It took me around 30 seconds to make the recommendations to user 5, can you imagine recommending to the 280,000 users in this dataset? Let alone the billions of users on real platforms.

For the rest of the users (sampled), an average of 20-30% P@10

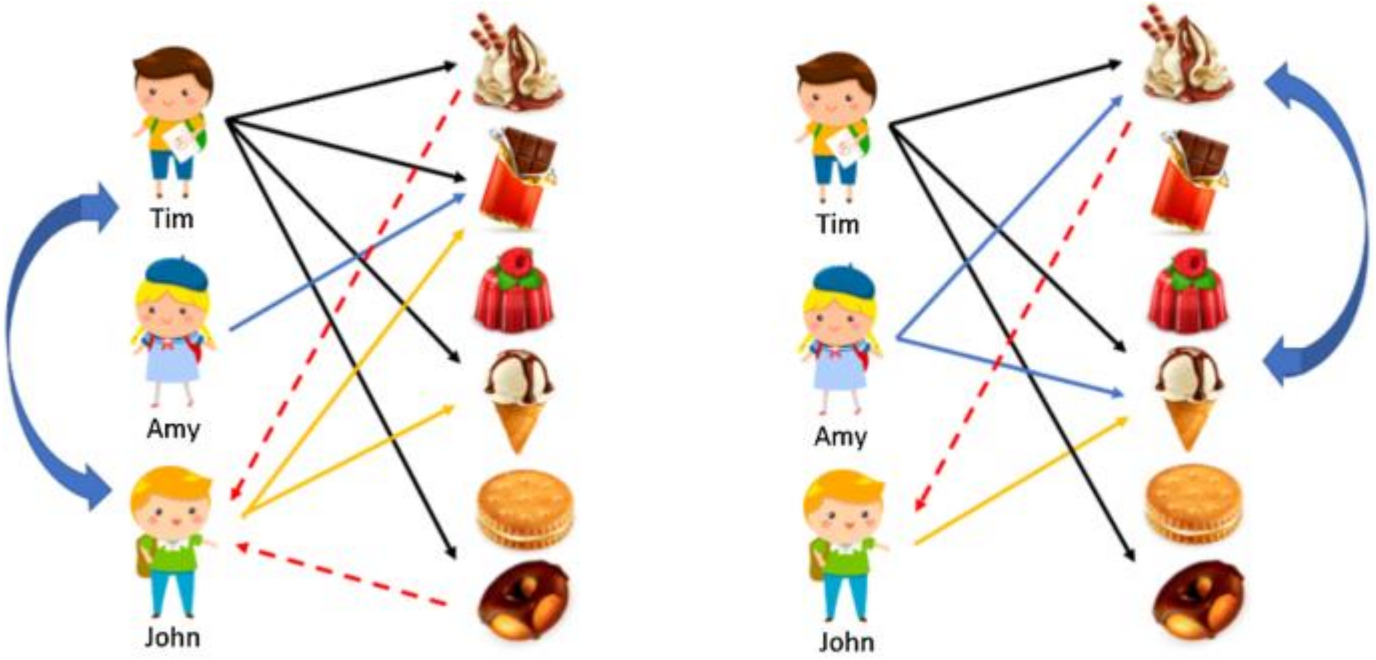
```
[ ]  uid = 30
     k = 10 #precision @ k
     urec = recommend_for_user(uid)
     jaccard_similarity(uid, k, urec, verbose=True)
```

```
User:  30
User liked:  {'Ronin', 'Some Like It Hot', 'Maltese Falcon, The', 'Godfather: Part III, The', 'Manchurian Candidate, The', 'Ver
Our recommendations:  {'Maltese Falcon, The', 'Manchurian Candidate, The', 'Vertigo', 'Laura', 'Double Indemnity', 'Big Sleep,
Intersection:  {'Maltese Falcon, The', 'Vertigo', 'Manchurian Candidate, The'}
3
```

```
[ ]  uid = 1007
     k = 10 #precision @ k
     urec = recommend_for_user(uid)
     jaccard_similarity(uid, k, urec, verbose=True)
```

```
User:  1007
User liked:  {'Forrest Gump', 'Pretty Woman', 'Sleepless in Seattle', 'Aladdin', 'Dave', 'Night to Remember, A', 'Steel Magnoli
Our recommendations:  {'On Golden Pond', 'Pretty Woman', "You've Got Mail", 'Notebook, The', 'Sleepless in Seattle', 'Magic of
Intersection:  {'Pretty Woman', 'Steel Magnolias', 'Sleepless in Seattle'}
3
```

Collaborative filtering



(a) User-based filtering (b) Item-based filtering

Memory-based models (memory killers)

```
872 block_values.fill(fill_value)
873 return new_block_2d(block_values, placement=placement)

MemoryError: Unable to allocate 57.2 TiB for an array with shape (283228, 27753444) and data type float64
```

the easy way to go here is to fill null values with zeroes, and calculate the pearson correlation between items to get item-item CF, or between users to get user-user CF, which performs much worse, but this is not our problem now.

The problem is that this doesn't scale as it needs terabytes of memory to store 58000 * 58000 (movies-movies)

to know exactly how much memory we would need, 58,000 * 58,000 * 8 bytes (float) = 26912000000 bytes = 26.912 Gigabytes

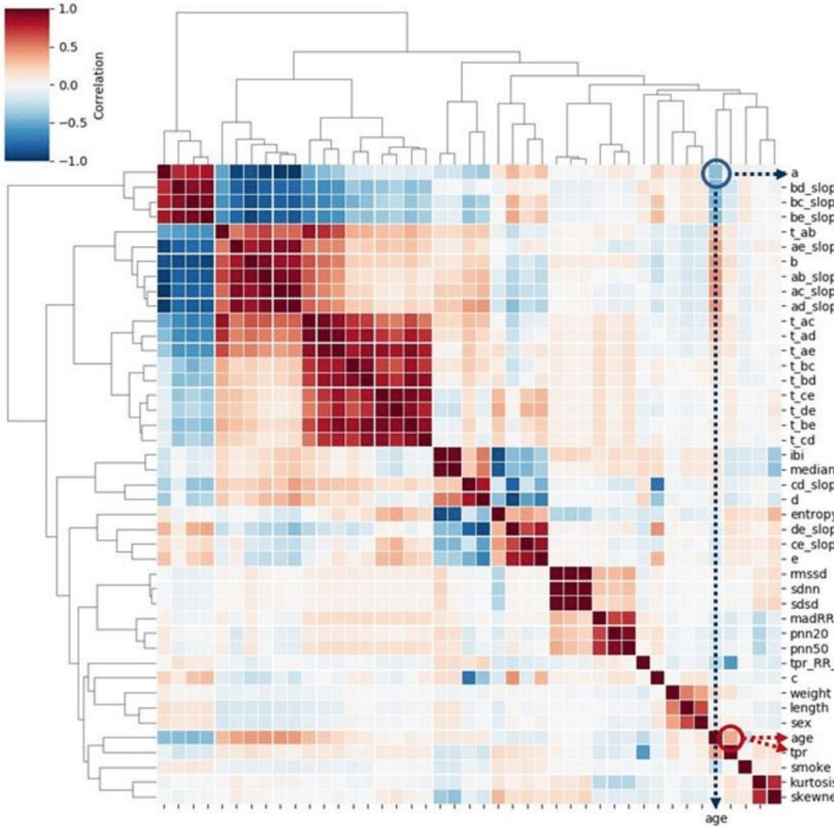
I don't know about you, but my computer doesn't have this amount of memory, so I will do it in a more memory efficent approach at the price of speed and computation.

Sparse matrices and cosine similarity

Our data is 95.5% sparse, with very few user-movie interactions in general

Ideally, you should just compute the pearson correlation between users to get user-user CF model and between items to get item-item CF model

This needs aaaaaaaaaaaaa LLOOOOOOOOOT of memory (280,000 * 280,000 floats in memory), let alone the computation itself



Approximate nearest neighbor search

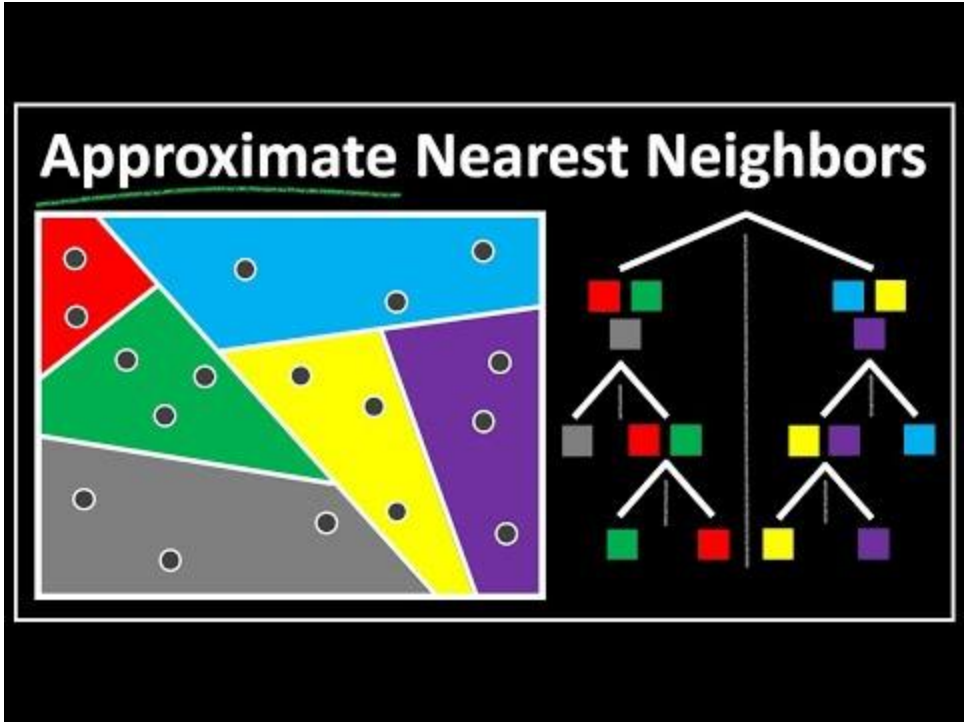
Instead of precomputing all 280,000 * 280,000 interactions, we will search for the nearest K neighbours in the data for each user

This doesn’t scale, so we need a heuristic for finding the nearest neighbours approximately.

This is the ANN.

I tried using pysparnn from Facebook, but it didn’t work.

I also used sklearn NearestNeighbors with a sparse matrix (which is supported with the minoweski distance only), but I also couldn’t get it to work



My fails 😥

The classes in `sklearn.neighbors` can handle either NumPy arrays or `scipy.sparse` matrices as input. For dense matrices, a large number of possible distance metrics are supported. For sparse matrices, arbitrary Minkowski metrics are supported for searches.

I found this module (pysparnn) from facebook research that does approximate nearest neighbours search on sparse matrices, but it was very very slow, and I couldn't get it to work as intended.

So I implemented my own, because my mama didn't raise a quitter

[ ]

```
'''
#knn model on the sparse user_movie_matrix
import pysparnn.cluster_index as ci

data_to_return = range(1)
cp = ci.MultiClusterIndex(user_movie_matrix, data_to_return)

cp.search(user_movie_matrix[:0], k=1, return_distance=False)
'''
```

So I implemented it 😈

The only problem that it takes around 15 minutes to make k recommendations for 1 user in the user-user CF

And around 2 hours to find top 10 similar movies in the item-item CF model
So use my implementation if you want to die waiting 😌

```python
def get_top_similar_cf(model, id, matrix, k):
    similarities = []
    if model == 'item':
        our_item = matrix.getcol(id).toarray()[0].T
        for i in tqdm_notebook(range(ratingsDfcopy['movieId'].nunique())):
                comparison_item = matrix.getcol(i).toarray()[0].T
                similarity = 1 - scipy.spatial.distance.cosine(our_item, comparison_item)
                similarities.append((similarity))
        topK = np.argpartition(similarities, -k)[-k:]
        return topK
    elif model == 'user':
        our_user = matrix.getrow(id).toarray()[0]
        print(our_user)
        for i in tqdm_notebook(range(ratingsDfcopy['userId'].nunique())):
                comparison_user = matrix.getrow(i).toarray()[0]
                similarity = 1 - scipy.spatial.distance.cosine(our_user, comparison_user)
                similarities.append((similarity))
        topK = np.argpartition(similarities, -k)[-k:]
        return topK
    else:
        return []
```

Very small subset test correlation matrix

```
(17762, 3)
(4441, 3)
```

```
[ ] from sklearn.metrics.pairwise import pairwise_distances

    user_correlation = 1 - pairwise_distances(train_data, metric='correlation')
    user_correlation[np.isnan(user_correlation)] = 0
    print(user_correlation[:4, :4])
```

```
[[1.         0.79283325 0.8088982  0.79887067]
 [0.79283325 1.         0.99964005 0.99995029]
 [0.8088982  0.99964005 1.         0.99985787]
 [0.79887067 0.99995029 0.99985787 1.        ]]
```

```
[ ] # Item Similarity Matrix
    item_correlation = 1 - pairwise_distances(train_data_matrix.T, metric='correlation')
    item_correlation[np.isnan(item_correlation)] = 0
    print(item_correlation[:4, :4])
```

```
[[1.00000000e+00 1.15296649e-02 7.86728896e-03]
 [1.15296649e-02 1.00000000e+00 9.21387985e-04]
 [7.86728896e-03 9.21387985e-04 1.00000000e+00]]
```

And its bad results (very high RMSE)

```
[ ]  user_prediction = predict(train_data_matrix, user_correlation, type='user')
     item_prediction = predict(train_data_matrix, item_correlation, type='item')
     print('User-based CF RMSE: ' + str(rmse(user_prediction, test_data_matrix)))
     print('Item-based CF RMSE: ' + str(rmse(item_prediction, test_data_matrix)))
```

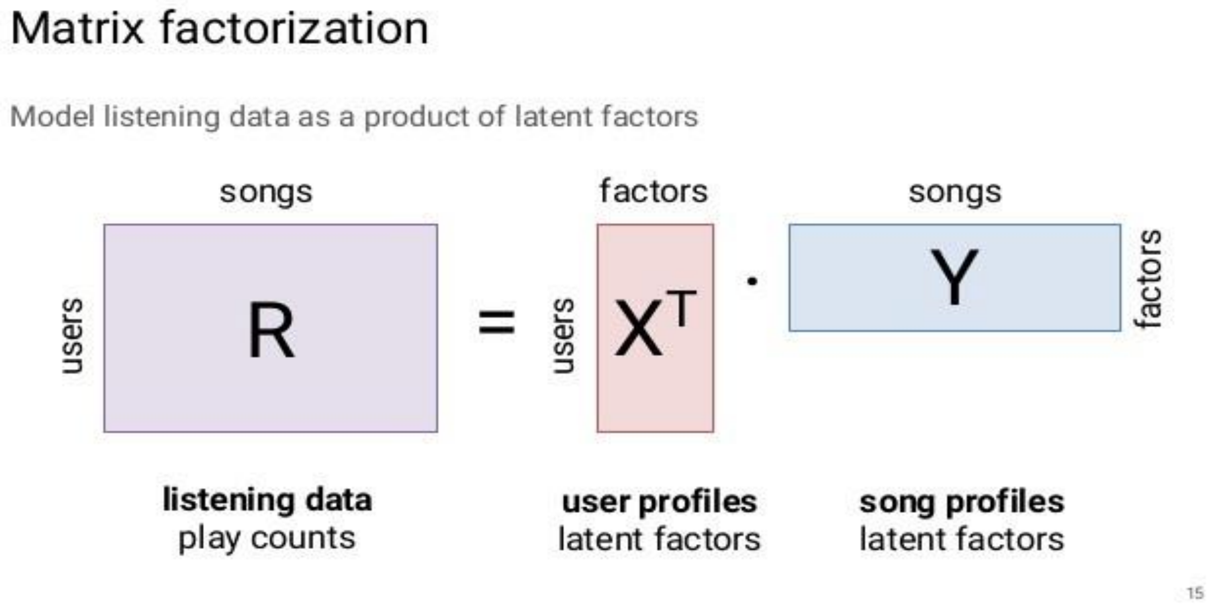
```
User-based CF RMSE: 61186.32388574864
Item-based CF RMSE: 71106.22453843542
```

```
[ ]  # RMSE on the train data
     print('User-based CF RMSE: ' + str(rmse(user_prediction, train_data_matrix)))
     print('Item-based CF RMSE: ' + str(rmse(item_prediction, train_data_matrix)))
```

```
User-based CF RMSE: 38813.92267804584
Item-based CF RMSE: 2114.5877116130873
```

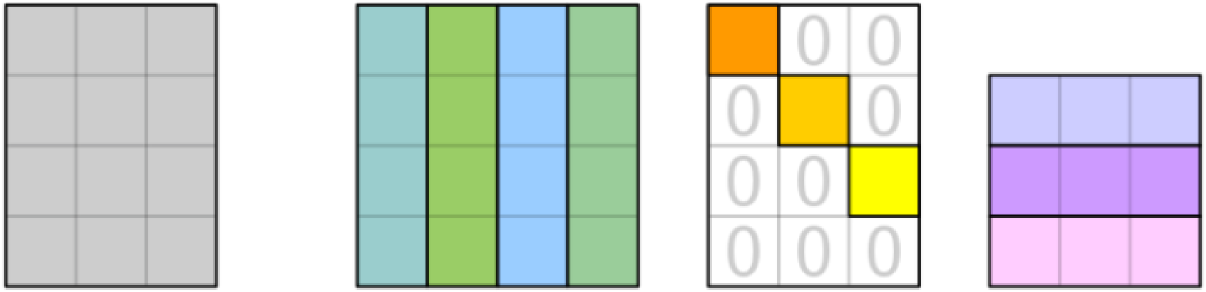
Finally, the MF (matrix factorization) model

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)



Use Singular Value Decomposition with a choice of embedding size

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)



$$\underset{m\times n}{\mathbf{M}} = \underset{m\times m}{\mathbf{U}}\;\underset{m\times n}{\boldsymbol{\Sigma}}\;\underset{n\times n}{\mathbf{V}^*}$$

Higher K → overfitting
Choose wisely (hyperparameter)

Multiply to get predictions, pick highest ratings

```
User 158 has already rated 60  movies.
Recommending highest 15 predicted ratings movies not already rated.
```

	userId	movieId	rating	title
30	158	2080	5.0	Lady and the Tramp
42	158	3408	5.0	Erin Brockovich
22	158	1304	5.0	Butch Cassidy and the Sundance Kid
25	158	1721	5.0	Titanic
27	158	1967	5.0	Labyrinth
29	158	2023	5.0	Godfather: Part III, The
31	158	2194	5.0	Untouchables, The
37	158	2687	5.0	Tarzan
40	158	2991	5.0	Live and Let Die
43	158	3871	5.0	Shane
20	158	1221	5.0	Godfather: Part II, The
44	158	4212	5.0	Death on the Nile
47	158	4896	5.0	Harry Potter and the Sorcerer's Stone (a.k.a. ...

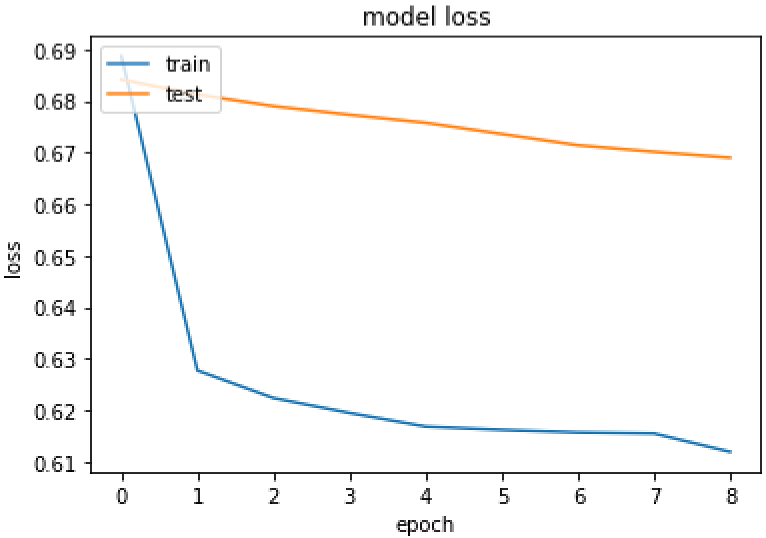
```
predictions
```

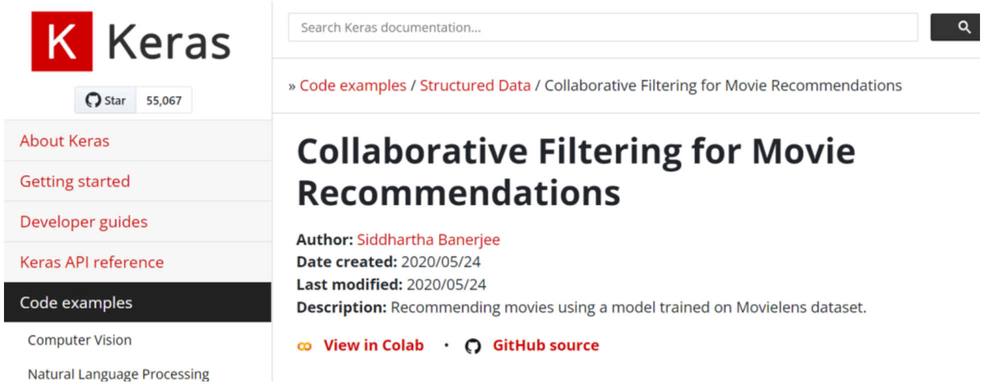
	movieId	title
1150	1193	One Flew Over the Cuckoo's Nest
27	32	Twelve Monkeys (a.k.a. 12 Monkeys)
1241	1291	Indiana Jones and the Last Crusade
103	111	Taxi Driver
1155	1200	Aliens
1057	1097	E.T. the Extra-Terrestrial
2175	2291	Edward Scissorhands
2987	3114	Toy Story 2
578	597	Pretty Woman
2871	2997	Being John Malkovich
4736	4878	Donnie Darko
1094	1136	Monty Python and the Holy Grail

A deep learning approach

Keras website had a nice guide with an embedding model

I just modified it to work on my dataset





Results were slightly bad as I trained it on My CPU for 5 epochs only on a very small subset of the data as a POC

Sample prediction of the deep learning model

```
Showing recommendations for user: 48426
====================================
Movies with high ratings from user
--------------------------------
Father of the Bride Part II : Comedy
Rashomon (Rashômon) : Crime Drama Mystery
--------------------------------
Top 10 movie recommendations
--------------------------------
Léon: The Professional (a.k.a. The Professional) (Léon) : Action Crime Drama Thriller
Pulp Fiction : Comedy Crime Drama Thriller
Shawshank Redemption, The : Crime Drama
Silence of the Lambs, The : Crime Horror Thriller
Godfather, The : Crime Drama
Saving Private Ryan : Action Drama War
American Beauty : Drama Romance
Amelie (Fabuleux destin d'Amélie Poulain, Le) : Comedy Romance
Spirited Away (Sen to Chihiro no kamikakushi) : Adventure Animation Fantasy
Lord of the Rings: The Return of the King, The : Action Adventure Drama Fantasy
```

Read more

`An amazing source that is worth every minute you spend on it`

`https://colab.research.google.com/github/google/eng-edu/blob/main/ml/recommendation-systems/recommendation-systems.ipynb`

Thank you